# Nagle's algorithm

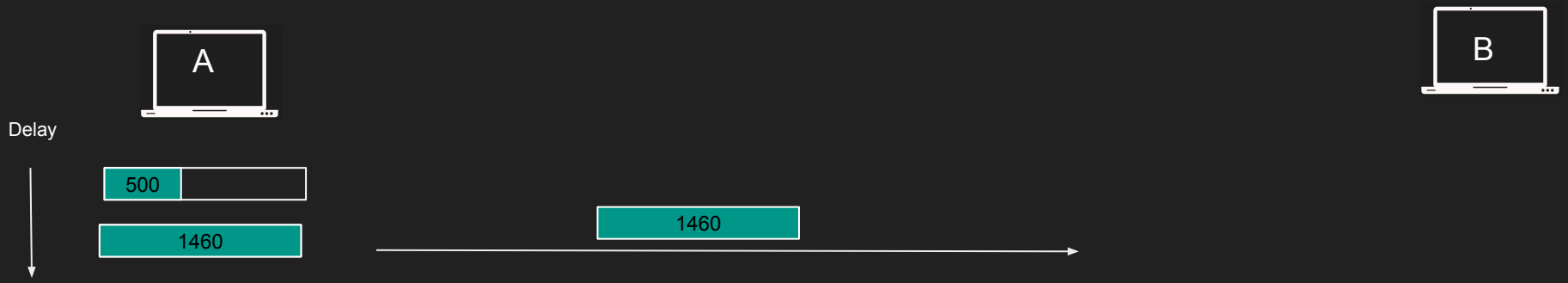
Delay in the client side

# Nigel Algorithm

- In the telnet days sending a single byte in a segment is a waste
- Combine small segments and send them in a single one
- The client can wait for a full MSS before sending the segment
- No wasted 40 bytes header (IP + TCP) for few bytes of data

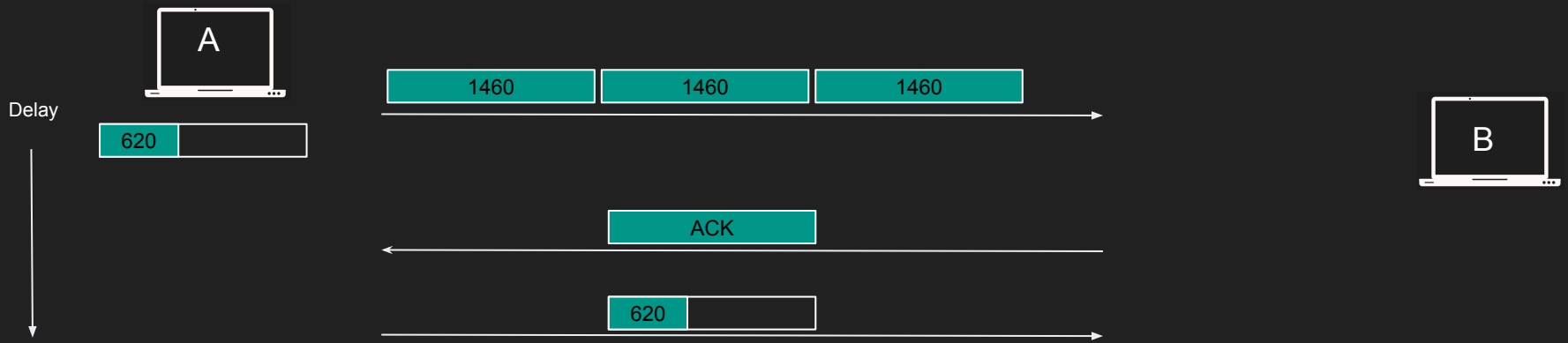
# Nagle's algorithm

- Assume MSS = 1460, A sends 500 bytes
- 500 < 1460 client waits to fill the segment
- A sends 960 bytes, segment fills and send
- If there isn’t anything to ACK data will be immediately sent



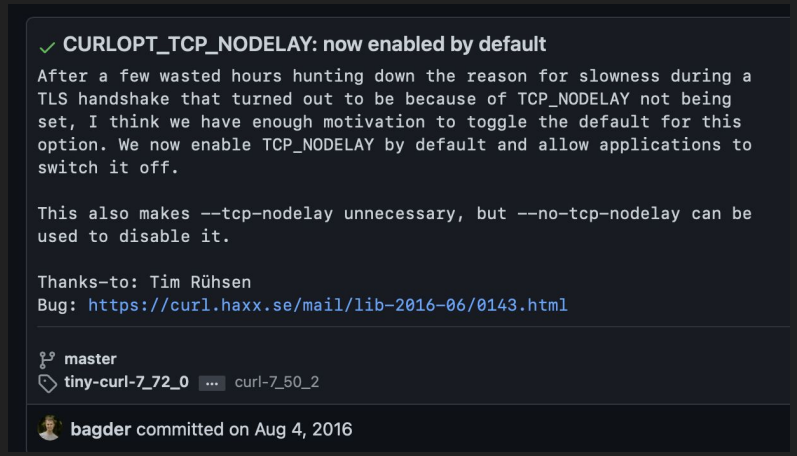
# Problem with Nagle's algorithm

- Sending large data causes delay
- A want to send 5000 bytes on 1460 MSS
- 3 full segments of 1460 with 620 bytes
- 4th segment is not sent!
- 4th not full segment are only sent when an ACK is received



# Disabling Nagle's algorithm

- Most clients today disable Nagle's algorithm
- I rather get performance than small bandwidth
- TCP_NODELAY
- Curl disabled this back in 2016 by default because TLS handshake was slowed down
- https://github.com/curl/curl/commit/4732ca5724072f132876f520c8f02c7c5b654d9



# Delayed Acknowledgement

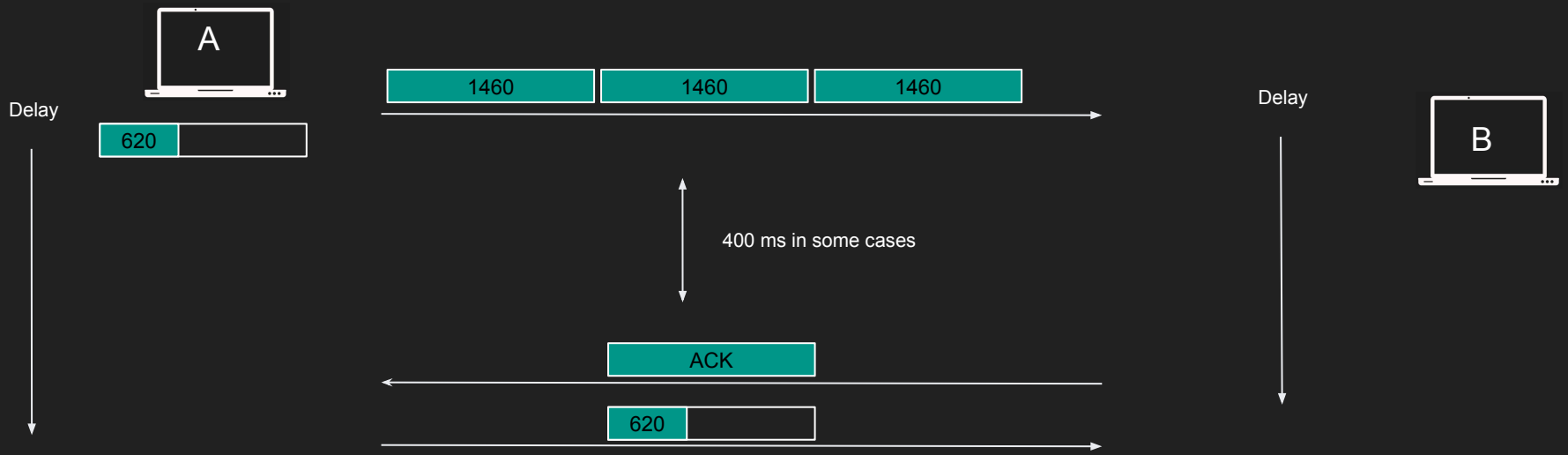
Less packets are good but performance is better

# Delayed Acknowledgment algorithm

- Waste to acknowledge segments right away
- We can wait little more to receive more segment and ack once

# Problem with delayed ACK

- Causes delays in some clients that may lead to timeout and retransmission
- Noticeable performance degradation
- Combined with Nagle's algorithm can lead to 400ms delays!
- Each party is waiting on each other



# Disabling Delayed acknowledgement algorithm

- Disable delayed ack algorithm can be done with TCP_QUICKACK option
- Segments will be acknowledged “quicker”

# The Cost of Connections

Understanding the cost of connections

# Connection establishment is costly

- TCP three way handshake
- The further apart the peers, the slower it is to send segments
- Slow start keeps the connection from reaching its potential right away
- Congestion control and flow control limit that further
- Delayed and Nigel algorithm can further slow down
- Destroying the connection is also expensive

# Connection Pooling

- Most implementation database backends and reverse proxies use pooling
- Establish a bunch of TCP connection to the backend and keep them running!
- Any request that comes to the backend use an already opened connection
- This way your connections will be “warm” and slow start would have already kicked in
- Don’t close the connection unless you absolutely don’t need it

# Eager vs Lazy Loading

- Depending on what paradigm you take you can save on resources
- Eager loading -> Load everything and keep it ready
  - Start up is slow but requests will be served immediately
  - Some apps send warm up data to kick in the slow start but be careful of bandwidth and scalability
- Lazy Loading -> only load things on demand
  - Start up is fast but requests will suffer initially