Written by Dr. Thapanapong Rukkanchanunt

06 Web Server

OUTLINE

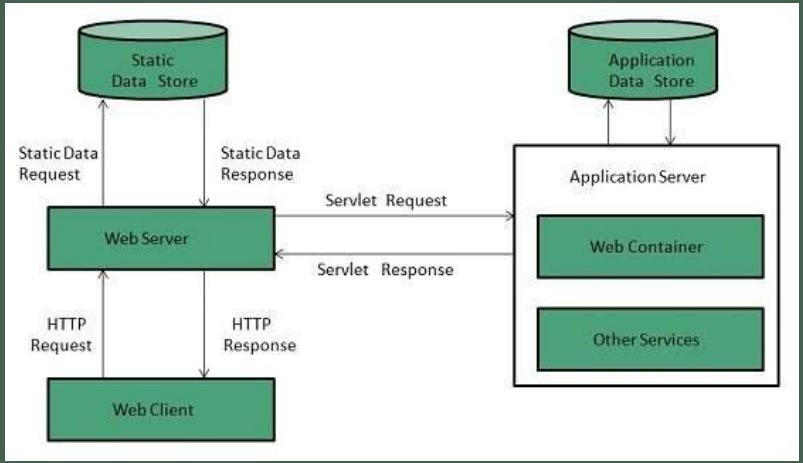
- The Basic
- Web Server Architecture
- Web Server in Action
- Web Content
- HTTP Requests/Responses
- Evolution of HTTP

The Basic

- Web Server เป็นตัวอย่างการนำเอาสถาปัตยกรรม Client-Server มาใช้งาน โดยผู้ใช้จะเชื่อมต่อเข้ากับ Web Server ผ่านทาง Web Browser หรือ Web Client
- Web Server จะเป็นที่เก็บ Web Content หรือ Web Site ดังนั้นการเชื่อมต่อระหว่าง Web Server และ Web Client จะเป็นการโหลด Web Content ที่ต้องการมาแสดงผล
- ในปัจจุบันการเขียน Web Server เองเป็นสิ่งที่ไม่ค่อยมีใครทำแล้ว ส่วนมากจะใช้ Framework (Middleware) เป็นสื่อกลางระหว่าง Web Client และ Web Server
  - Framework ที่นิยมใช้ได้แก่ ASP.NET, Apache, Ruby on Rails, NodeJS, และอื่น ๆ อีก

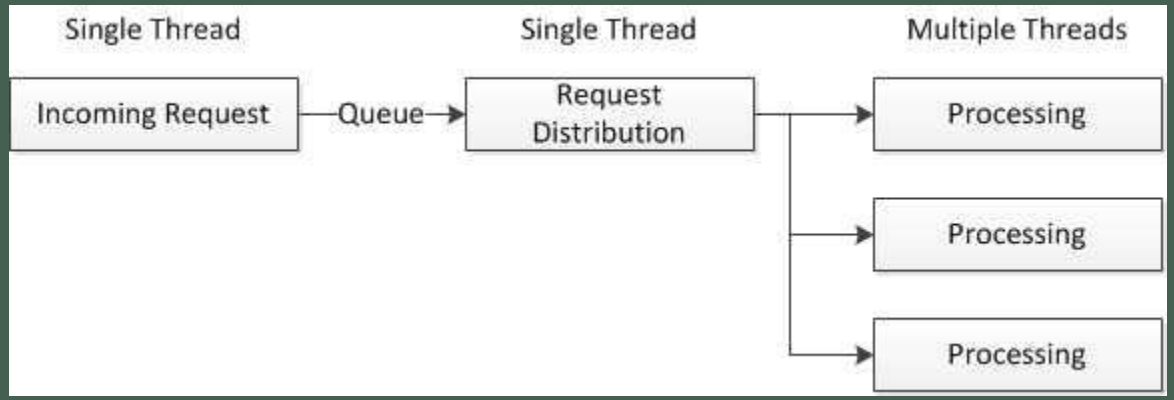
Web Server Architecture

- Web Server ตอบรับ Request จาก Web Client สองทางคือ
  1. ส่งไฟล์ที่เกี่ยวข้องกับ URL ที่ Request
  2. สร้าง HTTP Response ที่เกิดจากการประมวลผลบน Database
- เราสามารถออกแบบ Web Server ให้รองรับการเชื่อมต่อได้สองแบบ
  1. Concurrent Approach
  2. Event-Driven Approach



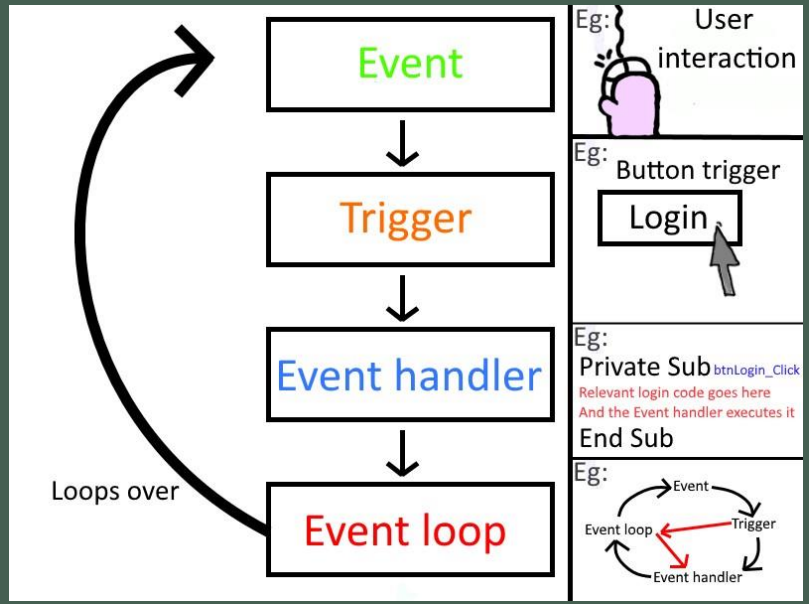
Concurrent Approach

- Multi-Process – Parent Process ส่ง Request ให้ Child Process ทำ ถ้าจำนวน Request น้อยก็จะสั่ง kill แต่ถ้าเยอะก็จะสั่ง fork ซึ่ง Child Process จะทำงานทีละ Request ให้จบก่อนรับ Request ใหม่
- Multi-Thread – ในหนึ่ง Process สามารถสร้างหลาย Thread มารับ Request ได้ แต่หน่วยความจำของ Thread ใน Process เดียวกันจะใช้ร่วมกัน



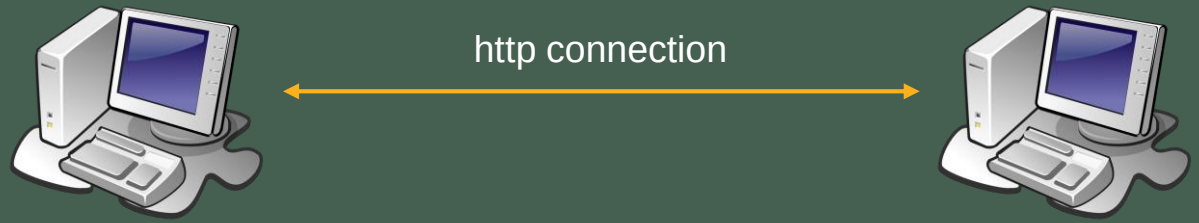
Event–Driven Approach

- Single Thread Single Process สร้าง Event Handler ขึ้นมาเพื่อรอการตอบรับจากผู้ใช้
- เมื่อมีอีเว้นท์เกิดขึ้นก็จะทำตามคำสั่งของ Callback Function ที่กำหนดไว้ใน Event Handler (Trigger)
- เมื่อทำเสร็จแล้วก็ทิ้ง Event Handler เดิม แล้วสร้าง Event Handler มาใหม่



Web Server in Action

- Client เชื่อมต่อกับ DNS (Domain-Name Server) เพื่อแปลงชื่อของ Server เป็น IP Address
- Client สร้างการเชื่อมต่อกับ Server แบบ HTTP (HyperText Transfer Protocol) ผ่าน TCP/IP (Transmission Control Protocol / Internet Protocol)
- Client ส่งชื่อของ Web Content ที่ต้องการไปยัง Server
- Server ส่งกลับ Web Content ที่เรียกมา



Web Content

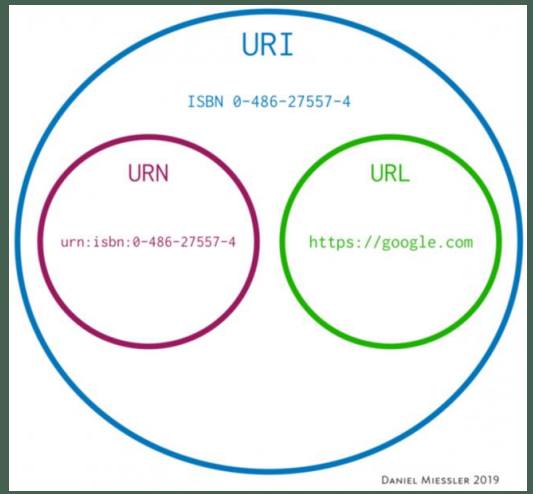
- Web Content คือข้อมูลที่อยู่ในรูปแบบของ MIME (Multipurpose Internet Mail Extensions) โดยที่ Header ของ Web Content จะต้องระบุประเภทของข้อมูล
  - Text/html — เอกสาร HTML
  - Text/plain — ข้อความที่ไม่มีการจัดระเบียบ
  - Application/postscript — เอกสาร Postscript (คล้าย PDF)
  - Image/gif — Binary image ในฟอร์แมต GIF
  - Image/jpeg — Binary image ในฟอร์แมต JPEG
  - และอื่น ๆ อีกมากมาย
- Client เลือกแอพเปิดข้อมูลตาม MIME type ที่ได้รับ (ในปัจจุบันเราเรียกว่า Media type)

Static and Dynamic Content

- Content ที่ส่งกลับมาจาก Server ในรูปแบบของ HTTP Response สามารถแบ่งได้เป็นสองแบบคือ Static และ Dynamic
  - Static Content คือ Content ที่ถูกบันทึกลงในไฟล์และจะถูกเรียกใช้ได้ทันทีตาม HTTP Request ตัวอย่างเช่นไฟล์ HTML รูปภาพ คลิปเสียง
  - Dynamic Content คือ Content ที่ถูกสร้างขึ้นตาม HTTP Request ที่ได้รับในขณะนั้น ตัวอย่างเช่นข้อมูลที่ได้จากการประมวลโดย Server จากความต้องการของ Client
- Search Engine โฟกัสเฉพาะ Static Content เมื่อทำการจัดลำดับผลลัพธ์ของเว็บ

HTTP Requests

- HTTP Requests ประกอบไปด้วย Request line ตามด้วย 0 หรือ Request headers
- Request line : <method> <uri> <version>
  - <version> คือเวอร์ชั่นของ HTTP ปัจจุบัน Version 3.0 เป็นเวอร์ชั่นล่าสุด เปิดใช้งานปี 2018 แต่หลายเว็บยังคงใช้ Version 2.0 อยู่
  - <uri> (Uniform Resource Identifier) คือตัวบ่งชี้ทรัพยากรใน Server สามารถเป็นได้ทั้ง URL (Uniform Resource Locator) ตัวบ่งชี้การเข้าถึงตำแหน่งของทรัพยากรใน Server หรือ URN (Uniform Resource Name) ตัวบ่งชี้ชื่อหรือชื่อ Attribute ของทรัพยากรนั้น ๆ



HTTP Requests’ Methods

- <method> เป็นการกระทำบนข้อมูลที่อ้างอิงใน <uri> เป็นไปได้ 7 แบบ
  - GET – เรียกดู Resource ที่อยู่ใน Server (ทั้ง Static และ Dynamic)
  - POST – ส่งข้อมูลไปแก้ไขสถานะที่เกี่ยวข้อง Resource นั้น
  - OPTIONS – กำหนดวิธีการเชื่อมต่อกับ Resource
  - HEAD – คล้าย GET แต่ดูเฉพาะ Header ของ Resource
  - PUT – เปลี่ยนค่าทั้งหมดใน Resource ด้วยค่าใหม่
  - DELETE – ลบ Resource
  - TRACE – ใช้สำหรับ Debug

HTTP Responses

- HTTP Responses ประกอบไปด้วย Response line ตามด้วย 0 หรือ Response header
- Response line : <version> <status code> <status msg>
  - <version> คือ เวอร์ชั่นของ HTTP Response อาจจะไม่เหมือนกับ HTTP Request ที่ส่งเข้ามา
  - <status code> คือตัวเลขที่ระบบสถานะของ Response
  - <status msg> คือข้อความภาษาอังกฤษที่อธิบาย <status code> เช่น
    - 200 OK ไม่มี Error
    - 403 Forbidden Server ไม่สามารถเข้าถึง Resource ได้
    - 404 Not found ไม่เจอ Resource ที่กำหนด

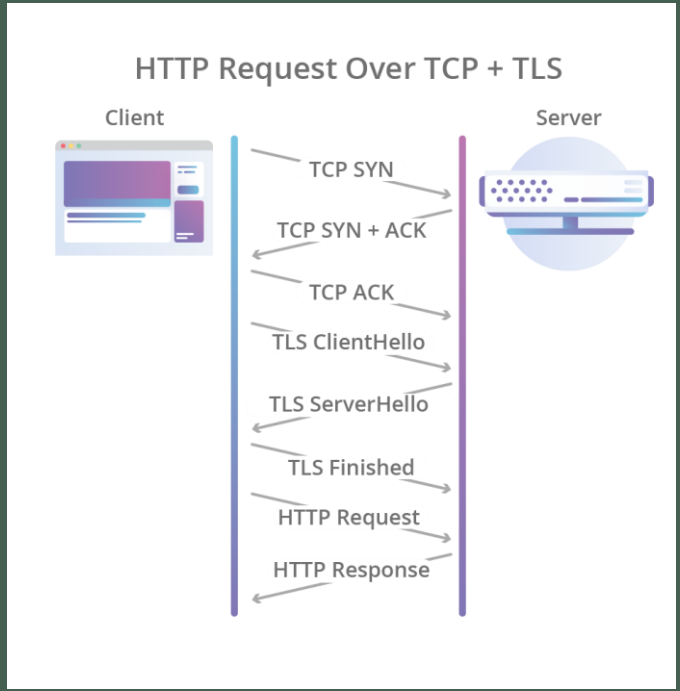
Response Header

- Response header : <header name>: <header data>
- ให้ข้อมูลเพิ่มเติมจากข้อมูลใน Response line
- Field ที่สำคัญใน <header data> ได้แก่
  - Content-Type คือ MIME Type ของ Response body

    Content-Type: text/html; charset=utf-8
  - Content-Length คือ ความยาวของ Response body (หน่วยเป็น Bytes)

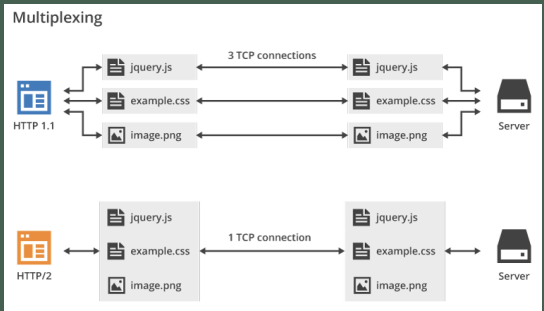
Evolution of HTTP

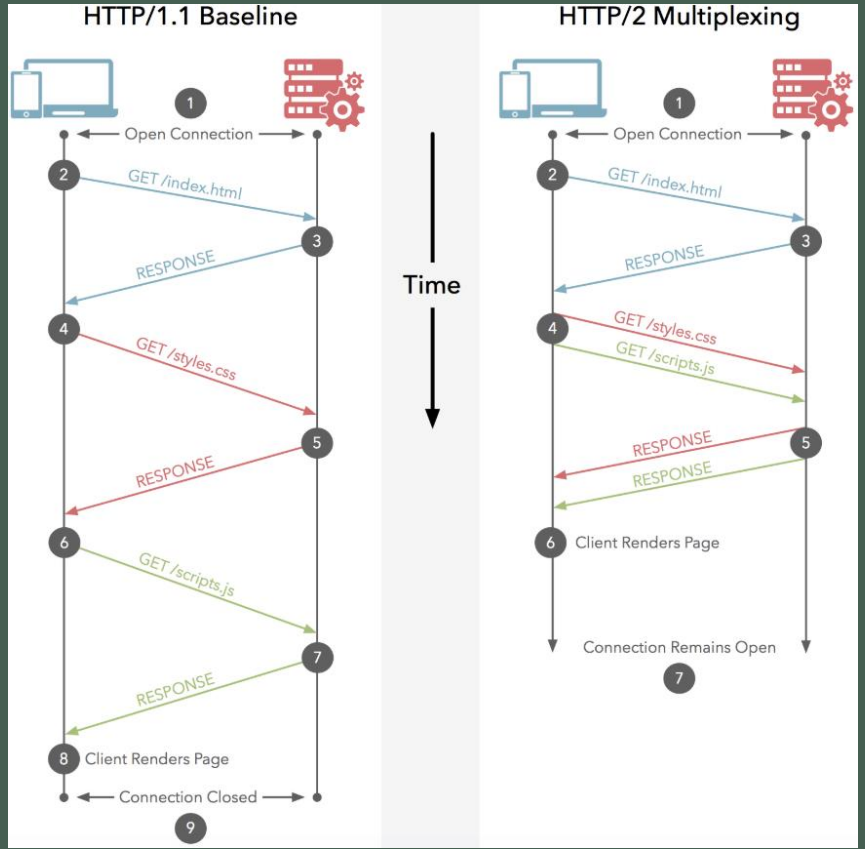
- HTTP 1.0 เริ่มต้นด้วยการสร้าง TCP Connection ทุกครั้งที่มีการ Request หรือ Response ข้อจำกัดคือกว่าจะรับส่งข้อมูลจริง Client-Server จะต้องผ่านการ Warm-up ก่อน ซึ่งในช่วงนี้ Bandwidth จะไม่ถูกใช้ได้เต็มที่
- HTTP 1.1 แก้ปัญหานี้ด้วยการ Keep-alive นั่นคือ Request ต่อเนื่องได้โดยไม่ต้องสร้าง Connection ใหม่



Evolution of HTTP (2)

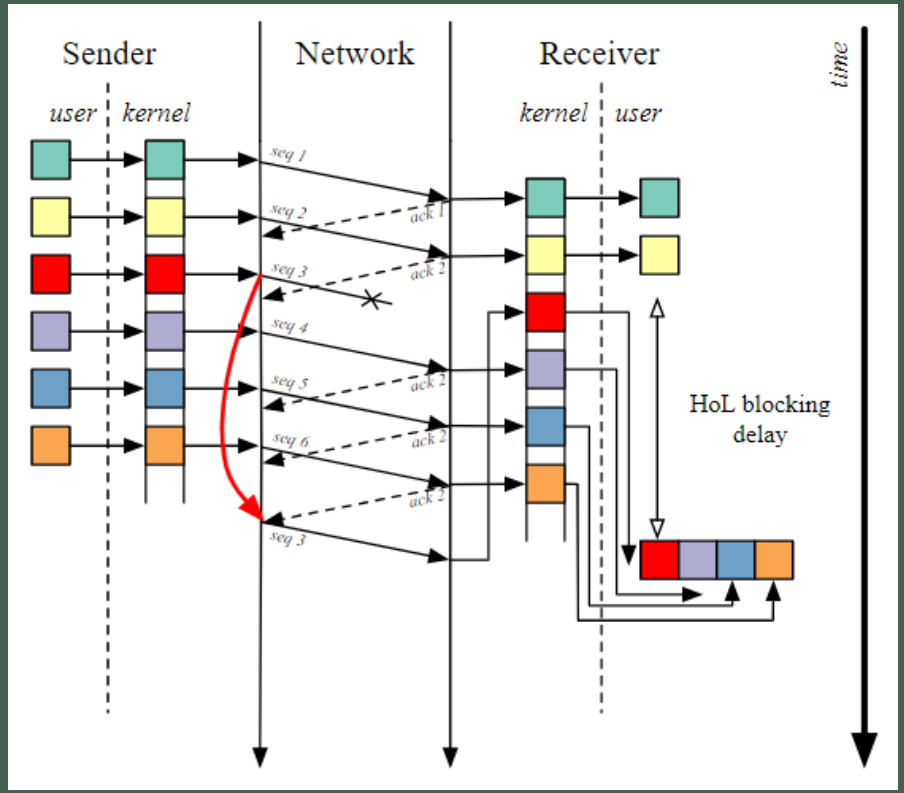
- พอเว็บมีการพัฒนา ทำให้มี Resource ที่ต้องดึงมาจาก Server มากขึ้น การทำงานแบบ Concurrency จึงมีความจำเป็นอย่างมาก HTTP 2.0 จึงอนุญาตให้ส่ง Request ได้หลายครั้งโดยไม่ต้องรอ Response





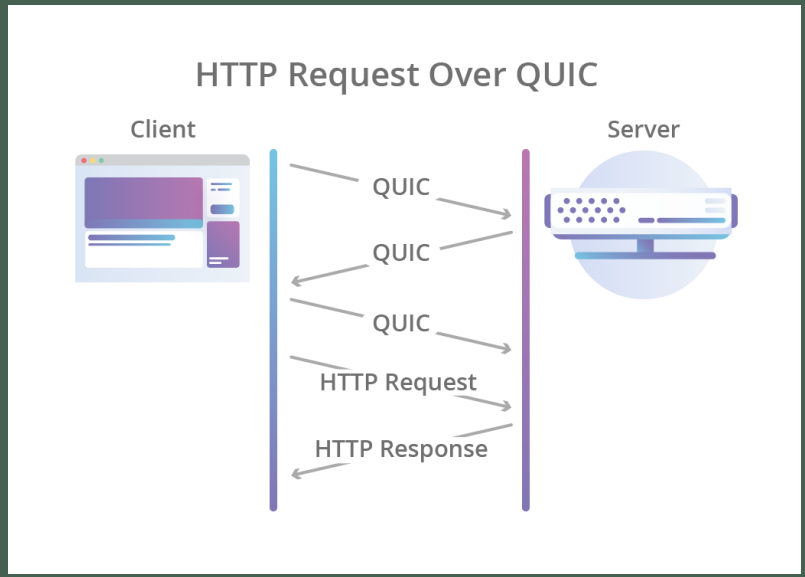
Head-of-line blocking

- การส่งข้อมูลผ่าน TCP มีข้อดีอยู่ที่การตรวจสอบว่า Packet ที่ส่งไปครบถ้วนหรือไม่ แต่นั่นก็ส่งผลให้การส่งมีความล่าช้าหากข้อมูลตกหล่น (Packet loss)
- เมื่อมีข้อมูลตกหล่นทำให้เกิดช่องว่างใน Data Stream ทำให้ TCP ต้องส่งข้อมูลที่หายไปอีกครั้งก่อนที่จะส่งข้อมูลที่เหลือตามมา ทั้ง ๆ ที่ข้อมูลที่เหลืออาจไม่มีส่วนเกี่ยวข้องกับปัญหาที่เกิดขึ้น
- ปัญหานี้มีชื่อเรียกว่า Head-of-line blocking



HTTP 3.0

- HTTP 3.0 แก้ปัญหาดังกล่าวด้วยการใช้การเชื่อมต่อแบบใหม่ QUIC
- QUIC (Quick UDP Internet Connections) พัฒนาโดย Google มีเป้าหมายทำให้การเชื่อมต่อรวดเร็วและปลอดภัยมากขึ้น
- QUIC Stream แชร์ QUIC Connection เลยไม่ต้องมีช่วง Warm-up อีกทั้ง QUIC Packet พัฒนามาจาก UDP Datagram ทำให้ส่งข้อมูลแยกกันได้และไม่ส่งผลต่อกันและกัน
- การเชื่อมต่อแบบ TCP ไม่มีการ Encrypt ทำให้สามารถดักสัญญาณได้ แต่การเชื่อมต่อแบบ QUIC มีการ Encrypt ตั้งแต่ต้น จึงมีความปลอดภัยมากกว่า



Implementation

- ในปัจจุบัน Chrome และ Firefox รองรับการใช้งาน HTTP 3.0
  - Chrome Canary และ Firefox Nightly Build เริ่มทดลองใช้งานครั้งแรกในเดือนกันยายน 2019
  - Chrome 79 เป็นเวอร์ชั่นแรกที่รองรับ HTTP/3 แบบเต็มตัวในเดือนธันวาคม 2019
  - Firefox 72.0.1 ที่ปล่อยให้ดาวน์โหลดในเดือนมกราคม 2020 สามารถรองรับ HTTP/3 ได้
- Open source libraries สำหรับ Developer ที่ต้องการพัฒนา Server โดยการใช้ QUIC และ HTTP/3 มีดังนี้ quiche ของ Couldflare, neqo ของ Mozilla, proxygen ของ Facebook และ Google, lsquic ของ LiteSpeed และอื่น ๆ อีกมากมาย
- จริง ๆ แล้ว QUIC มีมาสักพักแล้วและมีความพยายามที่จะนำเอา QUIC มาใช้งานร่วมกับ HTTP/2 แต่พบข้อจำกัดและใช้งานยาก จึงพัฒนาแยกออกมาเป็น HTTP/3