HTTP/3 Explained

by Daniel Stenberg

차례

HTTP/3 해설

이 책은 2018년 3월에 작성하기 시작했다. HTTP/3와 그 기반 프로토콜인 QUIC이 왜, 어떻게 동작하는지와 프로토콜의 상세내용, 그 구현체 등을 설명할 계획이다.

이 책은 완전히 무료이므로 돕고자 하는 사람은 누구나 참여해서 같이 만들 수 있다.

사전 요구사항

이 책의 독자는 TCP/IP 네트워크, HTTP의 기본, 웹을 어느 정도 이해하고 있다고 가정한다. HTTP/2에 대해서 더 알고 싶다면 http2 explained를 읽어보기를 권장한다.

저자

이 책은 Daniel Stenberg가 시작해서 만들어졌다. Daniel은 세계에서 가장 널리 사용되는 HTTP 클라이언트 소프트웨어인 curl을 만든 사람이자 curl의 리드 개발자다. Daniel은 20년 넘게 HTTP와 인터넷 프로토콜로 일했고 http2 explained의 저자이기도 하다.

홈페이지

이 책의 홈페이지는 daniel.haxx.se/http3-explained다.

도움 요청

이 문서에서 실수, 누락, 오류, 속 보이는 거짓말 등을 발견한다면 해당 문단의 수정 버전과 함께 보내주면 수정하고 도움 준 사람에게 적절한 크레딧을 줄 예정이다. 이 문서가 점점 나아지기를 기대한다.

가능하면 이 책의 github 페이지에 이슈를 생성하거나 풀 리퀘스트를 제출해주기 바란다.

라이센스

왜 QUIC인가

QUIC은 약어가 아니라 이름이다. 영어단어 "quick"과 똑같이 발음한다.

QUIC은 많은 부분에서 HTTP 같은 프로토콜에 적합하다. TCP와 TLS에서 동작하는 HTTP/2의 단점으로 알려진 문제를 해결하면서 안정적이고 안전한 새 전송 프로토콜로 볼 수 있다.

QUIC은 HTTP 전송에만 국한되지 않는다. 최종 사용자에게 웹과 데이터를 더 빨리 전달하려는 바람이 초기 이 새로운 전송 프로토콜을 만들게 된 가장 큰 이유이자 원동력이다.

그러면 새로운 전송 프로토콜은 왜 만들고 왜 UDP 상에서 동작하게 했는가?

HTTP/2를 기억하는가?

HTTP/2 명세인 RFC 7540는 2015년 5월에 발행되었고 이후 인터넷과 월드 와이드 웹에 널리 구현되고 배포되었다.

2018년 초 상위 1,000개의 웹사이트 중 거의 40%가 HTTP/2로 동작하고 있으며 Firefox가 보낸 모든 HTTPS 요청의 70% 정도가 HTTP/2 응답을 받았고 주요 모든 브라우저와 서버, 프락시가 이를 지원하고 있다.

HTTP/2는 HTTP/1의 수많은 결점을 수정했고 HTTP의 두 번째 버전을 도입함으로써 사용자들이 수많은 우회법을 사용하지 않게 되었다. 그 중 일부는 웹 개발자에게 상당한 부담이었다.

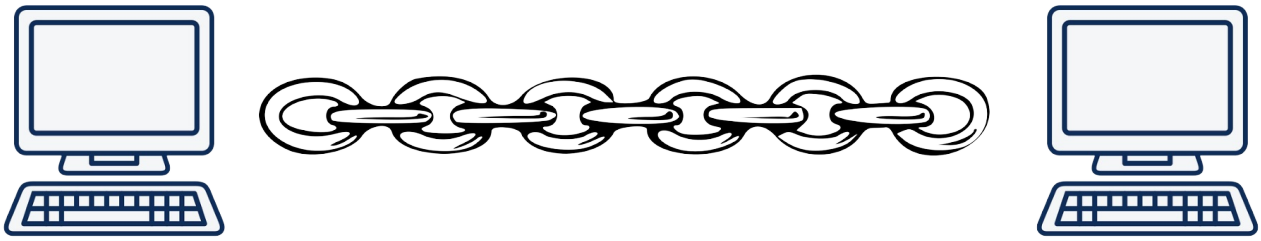
HTTP/2의 주요 기능 중 하나인 멀티플렉싱을 사용하여 같은 물리 TCP 연결을 통해 다수의 논리 스트림을 보낼 수 있게 된 것이다. 이는 많은 것을 더 좋고 빠르게 만들었다. 또한, 혼잡 제어 작업을 훨씬 낫게 해주어 사용자가 TCP를 훨씬 잘 사용하게 되면서 대역폭을 적절하게 가득 채워서 사용해 TCP 연결을 더 오래 유지되도록 만들었다. 이는 이전보다 더 자주 최대 속도를 낼 수 있기 때문에 좋아진 것이다. 헤더 압축은 대역폭을 적게 사용하게끔 해준다.

이전에는 브라우저가 호스트당 *6*개의 TCP 연결을 사용했지만, HTTP/2를 사용하면 보통 하나의 TCP 연결을 사용한다. 사실 HTTP/2에서 연결 병합과 "desharding" 기술을 사용하면 연결 수를 훨씬 더 줄일 수도 있다.

HTTP/2는 클라이언트가 다음 요청을 보내기 전에 첫 요청이 끝나기를 기다려야 하는 HTTP HOL(head of line) 블로킹 문제를 고쳤다.

TCP HOL(head of line) 블로킹

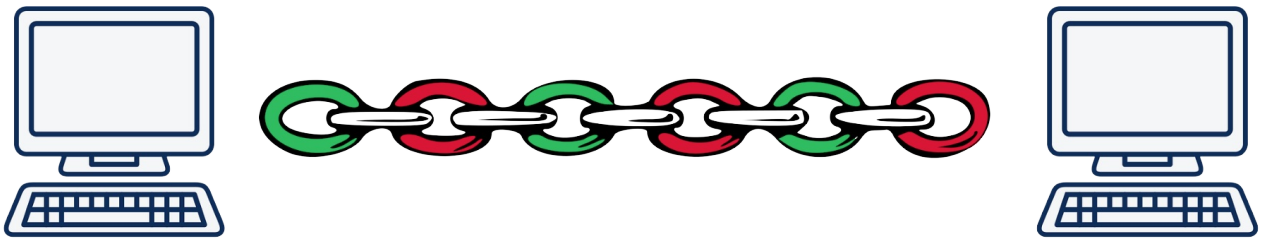
HTTP/2는 TCP를 사용하며 이전 HTTP 버전을 사용할 때 보다 더 적은 TCP 연결을 사용한다. TCP는 신뢰할 수 있는 전송 프로토콜이고 기본적으로 두 머신 간의 가상 체인으로 생각해도 된다. 네트워크의 한쪽 끝에 넣은 것이 최종적으로 다른 쪽 끝에 같은 순서로 나올 것이다.(아니면 연결이 끊어진다.)



HTTP/2를 사용하는 일반적인 브라우저는 TCP 연결 한개로 수십, 수백 개의 병렬 전송을 한다.

HTTP/2로 통신하는 두 엔드포인트 사이 네트워크 어딘가에서 하나의 패킷이 빠지거나 없어진다면 없어진 패킷을 다시 전송하고 목적지를 찾는 동안 전체 TCP 연결이 중단되게 된다. 즉, TCP는 "체인"이기 때문에 한 링크가 갑자기 사라지면 그 링크 이후에 와야 하는 모든 것들이 기다려야 한다는 뜻이다.

체인 메타포를 사용해서 이 연결을 통해 두 스트림을 보내는 경우를 그린 그림을 보자. 빨간색 스트림과 녹색 스트림이 있다.



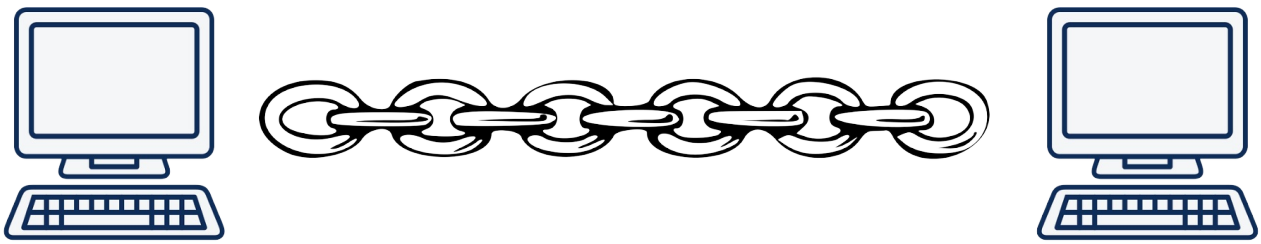
이는 TCP에 기반을 둔 head of line 블로킹이 된다!

패킷 손실률이 증가하면 HTTP/2의 성능도 저하된다. 테스트를 통해 패킷 손실률이 2%(상기하자면, 이는 끔찍한 네트워크 품질이다) 일때 HTTP/1 사용자가 더 나은 것으로 입증했다. 그 이유는 HTTP/1은 보통 손실된 패킷을 분배하는데 6개의 TCP 연결을 갖고 있어서 손실된 패킷이 없는 다른 연결은 계속 사용할 수 있기 때문이다.

TCP를 사용하는 한 이 이슈를 고치는 것은 (가능할 수도 있지만) 쉽지 않다.

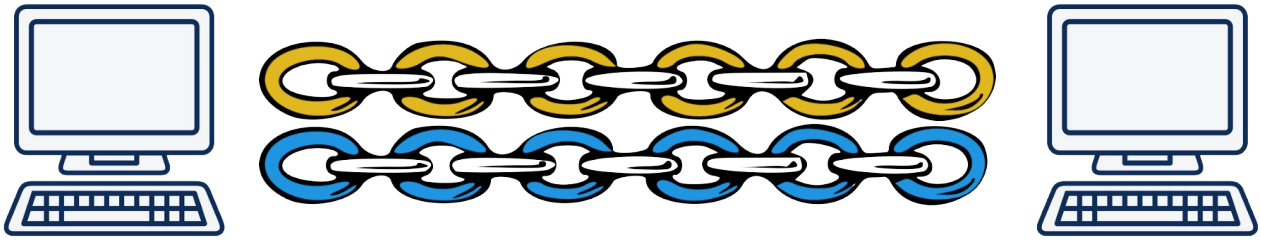
차단을 피하는 독립 스트림

QUIC에는 두 엔드포인트간에 연결을 안전하게 하고 데이터 전달을 신뢰할 수 있게 하는 연결 설정이 있다.



이 연결을 통해 두 가지 다른 스트림을 설정했을 때 이들을 독립적으로 다루므로 스트림 중 하나에서 어떤 링크가 사라지더라도 해당 스트림만(특정 체인) 멈추고 재전송될 없어진 링크를 기다란다.

다음은 두 엔드포인트간에 노란색과 파란색 스트림을 보내는 그림이다.



TCP 혹은 UDP

TCP에서 head-of-line 블로킹을 해결할 수 없다면 이론적으로 네트워크 스택에서 UDP와 TCP 옆에 새로운 전송 프로토콜을 만들 수 있다. 아니면 RFC 4960에서 IETF가 표준화하고 여러 가지 원하는 특성이 있는 전송 프로토콜인 SCTP를 사용할 수도 있다.

하지만, 최근 수년간 새로운 전송 프로토콜을 만들려는 노력은 인터넷에서 그것을 배포하는 어려움 때문에 완전히 멈춰 있다. 새 프로토콜 배포는 그 프로토콜이 도달해야 하는 사용자와 서버 사이에 있는 TCP와 UDP만 허용하는 방화벽, NAT, 라우터 등의 미들박스에 의해 방해받고 있다. 다른 전송 프로토콜의 도입은 UDP 또는 TCP 가 아닌 경우 이를 악의적이거나 뭔가 잘못되었다고 판단하고 차단해 버리는 박스들에 의해 차단되므로 연결의 N%가 실패한다. 노력을 기울이기에 N% 실패율은 너무 높다고 여겨진다.

게다가 보통 네트워크 스택의 전송 프로토콜 계층에서 뭔가를 바꾼다는 것은 보통 운영체제 커널에서 구현한 프로토콜을 말한다. 새로운 운영체제 커널을 갱신하고 배포하는 것은 상당한 노력이 필요한 느린 과정이다. IETF가 표준화한 수많은 TCP 개선사항도 광범위하게 지원되지 않아서 널리 배포되거나 사용되지 않고 있다.

왜 SCTP-over-UDP가 아닌가

SCTP는 스트림을 가진 신뢰성 있는 전송 프로토콜이고 WebRTC처럼 UDP 위에서 이 프로토콜을 사용하는 기존의 구현체도 있다.

다음의 몇 가지 이유로 QUIC의 대체재로 충분하지 않다고 생각했다.

- SCTP는 스트림에 대해 head-of-line 블로킹 문제를 고치지 못한다
- SCTP는 연결을 설정할 때 스트림의 수를 결정해야 한다
- SCTP에는 견고한 TLS/보안에 대한 언급이 없다
- SCTP는 4단계 핸드쉐이크(4-way handshake)를 사용하고 QUIC은 0-RTT를 제공한다
- QUIC는 TCP 같은 바이트스트림(bytestream)이고 SCTP는 메시지 기반이다
- QUIC 연결은 IP 주소 사이에서 마이그레이션을 할 수 있지만 SCTP는 할 수 없다

자세한 차이점이 알고 싶다면 A Comparison between SCTP and QUIC을 참고하라.

고착화(Ossification)

인터넷은 네트워크의 네트워크다. 이 네트워크의 네트워크가 의도대로 동작하게 하는 장비가 인터넷 여러 곳에 설치되어 있다. 우리는 이러한 기기(네트워크에 분포된 박스)를 미들박스라고 부른다. 이 박스는 전통적인 네트워크 데이터 전송의 주요 요소인 두 엔드포인트 사이에 있다.

박스는 여러 가지 다른 목적이 있지만 무언가 동작하게 하려면 이 박스가 해당 위치에 있어야 한다고 누군가 생각했기 때문에 곳곳에 깔린 것이라 생각한다.

미들박스는 네트워크 사이에서 IP 패킷을 라우팅하고, 악성 트래픽을 차단하고, NAT(Network Address Translation) 역할을 하고, 성능을 향상시키고, 통과하는 트래픽을 감시하는 등의 역할을 한다.

이러한 박스는 의무를 다하려면 자신들이 모니터링하고 수정하는 네트워크 및 프로토콜에 대해 반드시 알아야 한다. 박스는 이런 목적으로 소프트웨어를 실행한다. 그 소프트웨어를 항상 자주 업그레이드하는 것은 아니다.

박스는 인터넷이 유지되도록 연결하는 구성 요소이지만 종종 최신 기술을 따라잡지 못하는 경우가 있다. 보통 네트워크 중간 영역은 전 세계의 클라이언트나 서버 같은 엣지 만큼 빠르게 바뀌지 않는다.

어떤 프로토콜을 검사하기를 원하는지, 그리고 어떤것이 괜찮고 안 괜찮은지 알고있는 박스들은 과거에 배포되었고 그 당시의 프로토콜 기능 셋을 가지고 있다. 이전에는 몰랐던 새로운 기능이나 동작의 변경 사항이 추가되면 박스가 이를 잘못된 것이나 허용하지 않는 것으로 판단할 위험이 있다. 그래서 이러한 트래픽은 사용자가 해당 기능을 사용하고 싶지 않을 정도로 지연되거나 중단될 수 있다.

이를 "프로토콜 고착화(ossification)"라고 부른다.

TCP를 변경하는 것도 고착화 문제를 겪는다. 클라이언트와 원격 서버 사이에 있는 박스 중 일부는 알지 못하는 새로운 TCP 옵션을 발견하고 이 옵션이 무엇인지 몰라서 해당 연결을 차단해버릴 것이다. 프로토콜의 상세 내용을 탐지하도록 허용한 경우 시스템은 프로토콜이 보통 어떻게 동작하는지 배우게 되고 시간이 지나면 프로토콜을 변경할 수 없게 된다.

유일하게 고착화를 "방지하는데" 효과적인 방법은 미들박스가 지나가는 프로토콜에서 많은 것을 볼 수 없도록 통신을 최대한 암호화하는 것이다.

안전

QUIC은 항상 안전하다. 이 프로토콜에는 일반 텍스트 버전이 없으므로 QUIC 연결과 협상하려면 TLS 1.3을 사용해서 암호화 및 보안을 수행해야 한다. 위에서 언급했다시피, 이는 다른 차단이나 특별한 처리 같은 고착화 문제를 방지할 뿐만 아니라 QUIC이 웹 사용자가 기대하고 원하는 HTTPS의 모든 보안 속성을 가지게 만든다.

암호화 프로토콜 협상이 이뤄지기 전 초기 핸드쉐이크 패킷에 일반 텍스트 메시지가 아주 조금 있다.

더 이른 데이터

QUIC는 0-RTT, 1-RTT 핸드쉐이크 둘 다 제공하는데 이는 새로운 연결을 협상하고 설정하는데 걸리는 시간을 줄여준다. TCP의 3단계 핸드쉐이크(3-way handshake)와 비교해 보면 된다.

여기에 추가로 QUIC은 더 많은 데이터를 허용하는 "이른 데이터(early data)"를 처음부터 지원하고 이는 TCP Fast Open보다 더 쉽게 사용할 수 있다.

스트림 개념을 사용해서 기존에 존재하는 연결이 끝나기를 먼저 기다릴 필요 없이 같은 호스트로의 또 다른 논리 연결도 동시에 처리될 수 있다.

문제가 있는 TCP Fast Open

TCP Fast Open는 2014년 12월 RFC 7413로 발행되었고 이 명세는 이미 전달된 첫 번째 TCP SYN 패킷에 서버로 보낼 데이터를 전달하는 방법을 설명한다.

현실에서 이 기능의 실제 지원은 시간이 걸렸고 2018년인 오늘날까지도 문제로 가득하다. TCP 스택을 구현하는 사람이 문제를 이미 겪었으므로 애플리케이션이 이 기능의 이점을 취하려고 시도했다. 이를 활성화하기 위해 OS 버전이 무엇인지 알아야 할 뿐만 아니라 문제가 발생하면 그레이스풀하게 철회하고 문제를 다루는 방법을 찾아내야 한다. 여러 네트워크가 TFO 트래픽을 방해하는 것으로 밝혀졌고 이러한 TCP 핸드쉐이크를 적극적으로 망쳤다.

과정

Google의 Jim Roskind가 초기 QUIC 프로토콜을 설계하고 2012년 처음 구현했으며 Google의 실험을 확대한 2013년 전 세계에 공개적으로 발표했다.

그 당시에는 QUIC이 "Quick UDP Internet Connections"의 약자라고 주장했었지만, 이후에는 없어졌다.

Google이 프로토콜을 구현하고 이어서 널리 사용되는 자신들의 브라우저(Chrome)와 서버사이드 서비스(Google 검색, gmail, youtube 등)에 배포했다. Google은 프로토콜 버전을 꽤 빠르게 올리면서 이 개념이 시간이 지남에 따라 엄청난 수의 사용자에게 신뢰할 수 있게 동작한다는 것을 증명했다.

2015년 6월 표준화를 위해 QUIC의 첫 번째 인터넷 드래프트 버전을 IETF에 제출했지만 2016년 후반이 되어서야 QUIC 워킹 그룹이 승인되어 시작되었다. 하지만 이후 바로 많은 단체의 엄청난 관심을 받으면서 시작되었다.

2017년 Google의 QUIC 엔지니어가 인용한 수치에 따르면 전체 인터넷 통신량의 약 7%가 이미 QUIC을 사용한다고 한다. 이는 QUIC 프로토콜의 Google 버전이다.

IETF

IETF에서 프로토콜을 표준화하려고 만든 QUIC 워킹 그룹은 QUIC 프로토콜이 "단순히" HTTP가 아닌 다른 프로토콜을 전송할 수 있어야 한다고 빠르게 결정했다. Google-QUIC은 오로지 HTTP만 전송했고 실제로 HTTP/2 프레임 문법을 사용해서 HTTP/2 프레임을 효과적으로 전송했다.

IETF-QUIC은 Google-QUIC이 사용한 "커스텀" 접근 방법이 아니라 TLS 1.3의 암호화와 보안을 기반으로 두어야 한다고도 발표했다.

HTTP보다 더 많이 보내야 한다는 요구를 만족시키기 위해 IETF QUIC 프로토콜 아키텍처를 두 가지 별도의 계층인 전송 QUIC과 "HTTP over QUIC" 계층(후자는 종종 "hq"라고도 한다)으로 분할했다.

무해할 것처럼 들렸던 이 계층 분할로 인해 IETF-QUIC은 원래의 Google-QUIC과 많이 달라졌다.

하지만 워킹 그룹은 곧 적절하게 집중해서 제때 QUIC 버전 1을 제공할 수 있도록 HTTP를 제공하는 데 집중하고 HTTP가 아닌 전송은 나중 작업으로 남겨두기로 했다.

2018년 3월 이 책의 작업을 시작할 때 QUIC 버전 1의 최종 명세를 2018년 11월에 발표 할 계획이었다. 이는 나중에 2019년 7월로 연기되었다.

IETF-QUIC 작업이 진행되는 동안 Google 팀은 IETF 버전의 세부 내용을 받아들였고 IETF 버전이 만들어질 방향으로 Google 버전의 프로토콜을 천천히 발전시켰다. Google은 자사의 브라우저와 서비스에서 QUIC의 Google 버전을 계속해서 사용하고 있다.

개발 중인 대부분의 새로운 구현은 IETF 버전에 집중하고 Google 버전과는 호환되지 않는다.

HTTP/2에서의 경험

HTTP/2 명세 RFC 7540는 2015년 5월 발행되었는데 이는 QUIC이 처음으로 IETF에 들어오기 바로 한 달 전이다.

HTTP/2에서 유선으로 HTTP를 통한 HTTP를 변경할 기반이 마련되었고 HTTP/2를 만든 워킹그룹이 버전 1에서 버전 2로 가는 것보다 새로운 HTTP 버전으로 가는 걸 돕는 것이 훨씬 빠르다고 생각하게 되었다.(약 16년)

HTTP/2를 겪으면서 사용자와 소프트웨어 스택은 HTTP는 더는 텍스트 기반 프로토콜이라고만 가정할 수 없다고 생각하게 되었다.

HTTP-over-QUIC은 2018년 11월에 HTTP/3로 개명되었다.

상태

QUIC 워킹 그룹은 2016년 후반부터 프로토콜을 제정하는 일에 열심히 노력했고 현재 계획은 2019년 7월에 완료하는 것이다.

2018년 11월까지도 HTTP/3의 아직 대규모 연동 테스트를 진행하지 않았다. 기존에 두 개의 구현체가 있는데 둘 다 브라우저나 인기있는 공개 서버 소프트웨어와의 테스트를 하지 않았다.

QUIC 워킹 그룹의 위키 페이지에 나열된 15개 정도의 서로 다른 QUIC 구현체가 있지만 이들 모두 최신 명세 드래프트 개정판과 연동되지 않는다.

QUIC을 구현하는 것은 쉽지 않고 프로토콜은 오늘날 까지도 계속 진화하며 변하고 있다.

서버

지금까지 Apache나 nginx에서 공개적으로 QUIC을 지원한다는 발표는 없다.

클라이언트

아직 대형 브라우저 벤더중 아무도 QUIC이나 HTTP/3의 IETF 버전을 실행할 수 있는 버전을 제공하거나 발표하지 않았다.

수년간 Google Chrome은 Google 자체의 QUIC 버전의 구현체를 포함해서 배포했지만 이는 IETF QUIC 프로토콜과 연동되지 않고 그 HTTP 구현체는 HTTP/3와도 다르다.

구현 방해물

QUIC은 뭔가 새로운 것을 발명하지 않고 신뢰할 수 있는 기존 프로토콜을 사용하고자 TLS 1.3을 암호화와 보안 계층의 기반으로 사용하기로 했다. 하지만 이 작업을 하면서 워킹 그룹은 QUIC에서 TLS의 사용을 실제로 간소화하기로 발표했다. 즉, 프로토콜은 "TLS 레코드"는 사용하지 않고 "TLS 메시지"만 사용해야 한다.

이것이 무해한 변화처럼 들릴 수 있지만 수많은 QUIC 스택 구현자들에게는 커다란 걸림돌이 되었다. TLS 1.3을 지원하는 기존 TLS 라이브러리는 이 기능을 공개하거나 QUIC이 접근할 수 있는 API가 충분치 않다. 더 큰 조직에서 온 몇몇 QUIC 구현자들은 자신들의 TLS 스택에도 동시에 작업을 하고 있지만 모두가 그런 것은 아니다.

예시로 중량급 지배적 오픈 소스인 OpenSSL은 이에 대한 API가 전혀 없고 근래에 제공할 의사를 밝힌 적이 없다.(2018년 11월 기준)

이는 QUIC 스택이 다른 TLS 라이브러리(별도로 수정한 OpenSSL 빌드)를 사용하거나 향후 OpenSSL 버전을 수정할 필요가 있기 때문에 결국 배포 상의 걸림돌이다.

커널과 CPU 부하

Google과 Facebook은 QUIC의 대규모 배포가 TLS에서 HTTP/2로 서비스 할때보다 같은 트래픽 기준으로 거의 2배의 CPU가 필요하다고 얘기했다.

이는 다음과 같은 이유 때문이다.

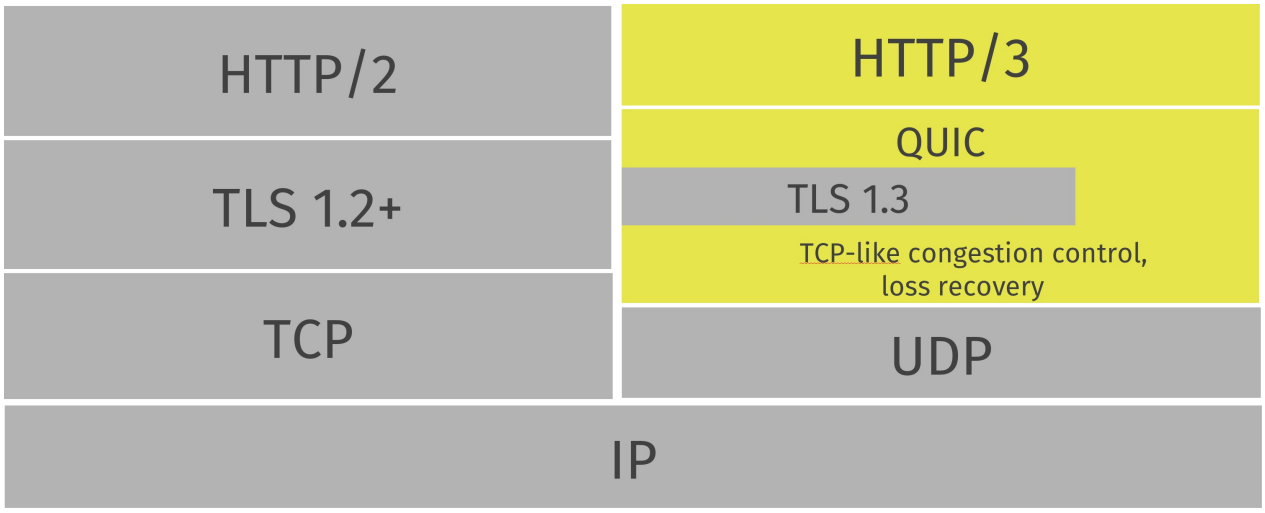
- 본질적으로 Linux의 UDP 부분은 TCP 스택만큼 최적화되어 있지 않다. 이는 전통적으로 지금처럼 고속 전송에 사용해오지 않았기 때문이다.
- 하드웨어가 TCP와 TLS의 부담을 덜어주고 있지만 UDP에 대해서 그렇게 하는 경우는 드물고 기본적으로 QUIC에 대해서는 아예 존재하지 않는다.

시간이 지나면 성능 및 필요한 CPU가 개선될 것이라고 생각한다.

프로토콜 기능

고수준에서 QUIC 프로토콜을 보자.

아래 그림은 HTTP를 전송할 때 HTTP/2 네트워크 스택을 왼쪽에 QUIC 네트워크 스택을 오른쪽에 보여준다.



UDP 상의 전송 프로토콜

QUIC은 UDP 위에 구현한 전송 프로토콜이다. 우리가 임의로 네트워크 트래픽을 보면 QUIC이 UDP 패킷으로 나타나는 것을 볼 것이다.

UDP에 기반을 둔 QUIC은 UDP 포트 번호를 사용해서 주어진 IP 주소의 특정 네트워크 서비스를 식별한다.

현재 알려진 모든 QUIC 구현체는 사용자 영역에 있으므로 커널 영역의 구현체보다 훨씬 빠른 발전이 가능하다.

동작할 것인가?

(DNS에 사용되는) 53 포트가 아닌 포트의 UDP 트래픽을 차단하는 기업용 및 기타 네트워크 설정이 있다. 또 다른 설정은 여러 방법으로 이러한 데이터를 제한하기 때문에 QUIC은 TCP에 기반을 둔 프로토콜보다 성능이 더 좋지 않다. 이 때문에, 몇몇 운영자가 해야 할 일에는 끝이 없다.

가까운 미래에 QUIC 기반의 모든 전송은 아마도 그레이스풀하게 (TCP 기반의) 다른 대안 프로토콜로 전환될 수 있어야 한다. Google 엔지니어는 측정한 실패 비율이 낮은 한 자릿수라고 얘기했다.

나아질 것인가?

QUIC이 인터넷 세상에 가치를 더할 수 있다고 증명한다면 사람들은 사용하기를 원할 것이고 그들의 네트워크에서 동작하길 원할 것이다. 그러면 회사들은 자신들의 장애물을 재고하기 시작할 것이다. 수년간 QUIC 개발은 진척이 있었으며, 인터넷에서 QUIC 연결을 설정하고 사용하는 성공률이 증가하고 있다.

신뢰할 수 있는 데이터 전송

UDP가 데이터 전송의 신뢰성을 보장하지 않지만, QUIC은 UDP 위에 새로운 계층을 추가함으로써 신뢰성을 제공한다. 추가된 계층은 TCP에 존재하는 패킷 재전송, 혼잡 제어, 속도 조정 및 다른 기능들을 제공한다.

한 엔드포인트로부터 QUIC을 통해 전송된 데이터는 연결이 유지되는 한 다른 엔드포인트에서 수신할 수 있다.

연결 내의 다중 스트림

QUIC은 SCTP, SSH, HTTP/2와 마찬가지로 물리 연결 내에서 논리적 스트림을 나눌 수 있다. 하나의 연결로 다수의 병렬 스트림으로 다른 스트림에 영향을 주지 않고 데이터를 동시에 전송할 수 있다.

두 엔드포인트 사이에 연결 협상이 이뤄지는 방식은 TCP 연결이 동작하는 방식과 비슷하다. QUIC 연결은 UDP 포트와 IP 주소로 이루어져 있지만 일단 연결을 만들고 나면 "connection ID"로 연결된다.

만들어진 연결을 통해 양쪽에서 스트림을 만들어 다른 쪽으로 데이터를 보낼 수 있다. 스트림은 순서대로 전달되고 신뢰할 수 있지만 서로 다른 스트림은 순서 없이 전달될 수 있다.

QUIC은 연결과 스트림 모두에서 흐름 제어를 제공한다.

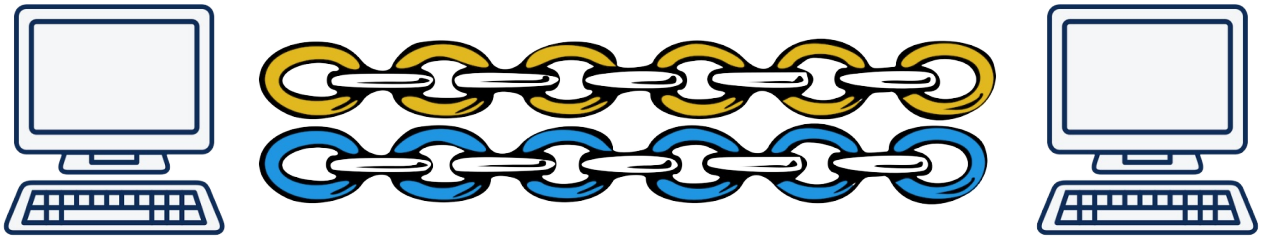
더 자세한 내용은 연결과 스트림 부분을 참고하라.

순서에 맞는 전송

QUIC은 스트림의 순서 있는 전송을 보장하지만 스트림 사이에서는 (순서를) 보장하지 않는다. 스트림은 순서대로 데이터를 전송하고 유지하지만, 각 스트림은 애플리케이션이 보낸 것과는 다른 순서대로 목적지에 도착할 수 있다!

스트림 A와 B가 서버에서 클라이언트로 이동하는 예를 생각해 보자. 스트림 A를 먼저 시작하고 이어서 스트림 B를 시작했다. QUIC에서 잃어버린 패킷은 해당 패킷이 속한 스트림에만 영향을 준다. 스트림 A는 패킷을 잃어버렸지만 스트림 B는 잃어버리지 않았다면 스트림 A의 잃어버린 패킷을 다시 전송하는 동안 스트림 B는 계속해서 전송하면서 완료될 수 있다. 이는 HTTP/2에서는 불가능했다.

다음은 하나의 연결을 통해 두 QUIC 앤드포인트 사이에 보내진 노란색 스트림과 파란색 스트림의 그림이다. 이 두 스트림은 독립적이고 다른 순서로 도착할 것이지만 각 스트림은 신뢰할 수 있게 애플리케이션에 순서대로 전달된다.



빠른 핸드쉐이크

QUIC은 0-RTT와 1-RTT 연결 설정을 둘 다 지원한다. 즉, 최선의 상황에서는 새로운 연결을 설정할 때 추가적인 라운드 트립이 전혀 필요치 않는다. 둘 중 가장 빠른 0-RTT 핸드쉐이크는 호스트에 이미 이전 연결이 구성되어 있고 해당 연결의 시크릿이 캐시 되어 있을 때만 동작한다.

초기 데이터

QUIC은 클라이언트가 이미 0-RTT 핸드쉐이크에 데이터를 넣을 수 있다. 이 기능으로 데이터를 피어에 최대한 빨리 전달할 수 있으므로 서버가 더 빨리 응답하고 데이터를 돌려줄 수 있다.

TLS 1.3

QUIC에서 사용하는 전송 보안은 TLS 1.3(RFC 8446)이며 암호화하지 않은 QUIC 연결은 절대로 존재하지 않는다.

이전 버전의 TLS와 비교해서 TLS 1.3에 몇몇 장점이 있지만 QUIC이 TLS 1.3을 사용한 주된 이유는 핸드쉐이크에 더 적은 라운드 트립이 필요하도록 바뀌었기 때문이다. 이는 프로토콜 지연을 줄여준다.

QUIC의 Google 레거시 버전은 커스텀 암호화를 사용했다.

전송 계층과 애플리케이션 계층

IETF QUIC 프로토콜은 전송 프로토콜로써 다른 애플리케이션 프로토콜이 그 위에서 사용할 수 있다. 첫 애플리케이션 계층 프로토콜은 HTTP/3(h3)다.

전송 계층은 연결과 스트림을 지원한다.

레거시 Google 버전 QUIC은 전송과 HTTP가 하나로 합쳐져 있어서 (IETF QUIC) 보다 더 특수한 목적의 send-http/2-frames-over-udp 프로토콜이었다.

QUIC을 통한 HTTP/3

HTTP/3라고 부르는 HTTP 계층은 HTTP 형태의 전송을 한다. 그 중에는 HPACK이라 부르는 HTTP/2 헤더 압축과 비슷한 QPACK을 사용한 HTTP 헤더 압축도 있다.

HPACK 알고리즘은 순차적인 스트림 전달에 의존하는데 QUIC은 스트림을 순서 없이 전달할 수 있으므로 HPACK을 수정하지 않고는 HTTP/3에서 재사용할 수 없다. HPACK을 QUIC에 맞게 수정한 것을 QPACK이라 볼 수 있다.

QUIC을 통한 HTTP가 아닌 프로토콜

QUIC을 통해 HTTP가 아닌 프로토콜을 보내는 작업은 QUIC 버전 1 출시 이후로 연기되었다.

QUIC의 동작 방식

이번 장에서는 QUIC 전송 프로토콜의 기본적인 구성 요소가 어떻게 동작하는지를 설명한다. QUIC 스택을 직접 구현하고자 한다면 이 책의 설명을 통해 전반적인 내용을 이해할 수는 있지만, 세부 내용은 IETF의 인터넷 드래프트와 RFC를 참고하기 바란다.

1. 연결을 설정한다.
2. 여기에는 TLS 보안이 포함되어 있다.
3. 스트림을 사용한다.

연결

QUIC 연결은 두 QUIC 엔드포인트 사이의 대화이다. QUIC의 연결 설정은 버전 협상, 암호화, 전송 핸드쉐이크로 구성되어 있으므로 연결 설정의 지연시간을 줄여준다.

이러한 연결을 통해 실제 데이터를 보내려면 하나 이상의 스트림을 만들어서 사용해야 한다.

연결 ID

각 연결은 연결 식별자나 연결 ID를 가지므로 이를 통해 연결을 식별한다. 엔드포인트가 자유롭게 연결 ID를 선택한다. 각 엔드포인트는 엔드포인트의 피어가 사용할 연결 ID를 선택한다.

이 연결 ID의 주요 기능은 하위 프로토콜 계층(UDP, IP 혹은 그 아래 계층)에서 주소가 변경되더라도 QUIC 연결의 패킷이 잘못된 엔드포인트로 전달되지 않도록 보장하는 것이다.

연결 ID를 이용하면 TCP에서는 불가능했던 방법으로 IP 주소와 네트워크 인터페이스 사이에서 연결이 마이그레이션 할 수 있다. 예를 들면 사용자가 자신의 기기를 들고 wifi가 지원되는 곳으로 이동했을 때 다운로드를 진행하면서 셀룰러 네트워크 연결에서 더 빠른 wifi 연결로 변경되는 것이 이 마이그레이션을 통해 가능하다. 마찬가지로 wifi를 이용할 수 없게 되었을 때 셀룰러 연결을 통해 다운로드를 진행할 수 있다.

포트 번호

QUIC이 UDP 위에 만들어졌으므로 들어오는 연결을 구분하기 위해 16비트 포트 번호 필드를 사용한다.

버전 협상

클라이언트는 QUIC 연결 요청에서 어떤 QUIC 프로토콜 버전으로 통신하고 싶은지 서버에게 알려주고 서버는 클라이언트가 선택할 수 있도록 지원하는 버전 목록을 응답한다.

TLS을 사용하는 연결

초기 패킷이 연결을 설정하자마자 초기화하는 쪽에서 보안 계층 핸드쉐이크의 설정을 시작하는 암호화 프레임을 보낸다. 보안 계층은 TLS 1.3 보안을 사용한다.

QUIC 연결에서에는 반드시 TLS를 사용해야 한다. QUIC 프로토콜은 프로토콜 고착화를 방지하기 위해 미들박스가 조작하기 어렵게 설계되었다.

스트림

QUIC 스트림은 경량이면서 순서가 있는 바이트 스트림 추상화를 제공한다.

QUIC에는 두가지 기본 스트림 타입이 있다.

- 단방향 스트림은 한쪽으로 데이터를 전달한다. 즉, 스트림을 시작한 쪽에서 피어로 전달된다.
- 양방향 스트림은 양쪽으로 데이터를 보낼 수 있다.

두 엔드포인트가 두 타입의 스트림을 모두 만들 수 있고 스트림은 다른 스트림과 상호배치된 데이터를 동시에 보낼 수 있고 취소할 수도 있다.

QUIC 연결을 통해 데이터를 보낼 때 하나 이상의 스트림을 사용한다.

흐름 제어

스트림은 개별적으로 흐름 제어가 되므로 엔드포인트가 메모리 사용을 제한할 수 있고 백프레셔(back pressure)를 적용할 수도 있다. 스트림의 생성도 흐름 제어가 되고 각 피어는 일정 시간 동안 받고자 하는 최대 스트림 ID를 정의할 수 있다.

스트림 식별자

스트림 ID라고 부르는 부호없는 62비트 정수로 스트림을 식별한다. 스트림 ID의 최하위 2비트를 스트림(단방향이나 양방향이나)의 타입과 스트림을 시작한 쪽을 식별하는 데 사용한다.

스트림 ID의 최하위 비트(0x1)는 누가 스트림을 시작했는지를 식별한다. 클라이언트는 짝수의 스트림(최하위 비트가 0으로 설정된 스트림)을 초기화 하고 서버는 홀수의 스트림(최하위 비트가 1으로 설정된 스트림)을 초기화한다.

스트림 ID의 두 번째 하위 비트(0x2)는 단방향 스트림과 양방향 스트림을 구분한다. 단방향 스트림을 항상 이 비트를 1로 설정하고 양방향 스트림은 이 비트를 0으로 설정한다.

스트림 동시성

QUIC에서는 임의의 스트림이 다수 동시에 동작할 수 있다. 엔드포인드는 최대 스트림 ID를 제한해서 동시에 받아들이는 활성 스트림의 수를 제한한다.

엔드포인트마다 최대 스트림 ID를 설정하고 설정을 받는 피어에만 적용된다.

데이터 보내기와 받기

엔드포인트는 스트림을 사용해서 데이터를 보내고 받고 이는 스트림의 원래 목적이다. 스트림은 순서가 맞는 바이트 스트림 추상화다. 하지만 별도의 스트림은 원래 순서대로 전달되지 않는다.

스트림의 우선순위

스트림에 할당된 리소스의 우선순위가 제대로 설정되면 스트림 멀티플렉싱은 애플리케이션 성능에 상당한 영향을 준다. HTTP/2 같은 멀티플렉싱을 지원하는 다른 프로토콜의 경험에서 상당히 긍적적인 성능 효과를 보여주는 효율적인 우선순위 전략을 볼 수 있다.

QUIC 자체에서 우선순위를 결정하는 정보를 교환하는 프레임을 제공하지는 않는다. 대신 QUIC을 사용하는 애플리케이션에서 우선순위 정보를 받는데 의존한다. QUIC을 사용하는 프로토콜은 애플리케이션의 의미에 적합한 우선순위 스키마를 정의할 수 있다.

QUIC을 통해 HTTP/3을 사용할 때 우선순위는 HTTP 계층에서 동작한다.

0-RTT

새로운 연결을 설정하는데 필요한 시간을 줄이려고 이전에 서버에 연결했던 클라이언트는 해당 연결의 특정 파라미터를 캐시한 뒤 이어서 서버와 **0-RTT** 연결을 설정할 수 있다. 이로 인해서 클라이언트는 핸드쉐이크가 완료되기를 기다리지 않고 바로 데이터를 보낼 수 있다.

스핀 비트(Spin Bit)

QUIC 워킹그룹에서 수백 개의 이메일을 주고받고 수십 시간의 토론을 걸친 가장 긴 설계 토론 중 하나가 단일 비트 즉, 스핀 비트(Spin Bit)에 관한 것이다.

이 스핀 비트를 지지하는 사람들은 두 QUIC 엔드포인트 사이에 있는 운영자나 사람들이 지연 시간을 측정할 필요가 있다고 주장한다.

이 기능을 반대하는 사람들은 잠재적인 정보 유출을 좋아하지 않았다.

비트 회전시키기

클라이언트와 서버 두 엔드포인트는 각 QUIC 연결에 대해 0 또는 1의 스핀 값을 유지하면서 해당 연결에 적절한 값으로 스핀 비트를 설정해서 패킷을 보낸다.

양 측은 한 라운드 트립동안 같은 값으로 설정된 스핀 비트를 가진 패킷을 보내고 이후 이 값을 토글한다. 이로 인해 비트 필드에 0과 1의 파동이 나타나고 관찰자가 이를 모니터링할 수 있다.

발송자가 애플리케이션도 아니고 제약된 흐름 제어도 아닌 경우에만 이 측정을 할 수 있고 네트워크에서 패킷의 순서가 재정리될 때도 데이터에 잡음이 낄 수 있다.

사용자 영역

사용자 영역에서 전송 프로토콜을 구현하면 클라이언트와 서버가 새로운 버전을 배포하기 위해 운영체제 커널을 업데이트할 필요가 없어서 비교적 쉽게 프로토콜을 발전시킬 수 있으므로 프로토콜을 빠르게 반복할 수 있다.

미래에 운영체제 커널에서 구현되거나 제공하지 않도록 막는 것은 QUIC에 아무것도 없으므로 누군가 좋은 방법을 찾아야 한다.

많은 구현체

사용자 영역에서 새로운 전송 프로토콜을 구현하는 한 가지 명백한 효과는 다수의 독립적인 구현체를 볼 수 있다는 것이다.

예견할 수 있는 미래에 다른 애플리케이션은 다른 HTTP/3와 QUIC 구현체를 포함할(또는 계층 위에서) 것이다.

API

TCP와 TCP를 사용하는 프로그램의 성공 요소 중 하나는 표준화된 소켓 API이다. 소켓 API는 기능이 잘 정의되어 있고 이 API를 사용하면 TCP가 똑같이 동작하므로 다수의 다른 운영체제 사이에서 프로그램을 이동시킬 수 있다.

QUIC은 그렇지 않다. QUIC에는 표준 API가 없다.

QUIC에서는 기존 라이브러리 구현체 중 하나를 선택하고 그 API를 따라야 한다. 이는 애플리케이션을 어떤 부분에서 하나의 라이브러리에 "락인(locked in)"시킨다. 다른 라이브러리로 바꾼다는 것은 다른 API를 뜻하므로 많은 작업이 필요할 것이다.

또한 QUIC이 보통 사용자 영역에서 구현되므로 소켓 API를 쉽게 확장할 수 없고 기존의 TCP나 UDP 기능과 비슷하게 보일 수 없다. QUIC을 사용한다는 것은 소켓 API와는 다른 API를 사용한다는 것을 의미한다.

HTTP/3

이전에 얘기했듯이 QUIC을 통해 전송하는 첫 주요 프로토콜은 HTTP다.

완전히 새로운 방법으로 유선을 통해 HTTP를 전송하려고 HTTP/2를 도입한 것처럼 HTTP/3는 다시 한번 네트워크를 통해 HTTP를 전송하는 새로운 방법을 도입한다.

HTTP는 여전히 이전과 같은 개념과 패러다임을 유지한다. 헤더와 보디가 있고 요청과 응답이 있고 동사, 쿠키, 캐시가 있다. 통신 상대측으로 비트를 보내는 방법이 HTTP/3에서 주된 변경점이다.

QUIC을 통해 HTTP를 보내기 위해 변경이 필요했고 그 결과물을 HTTP/3라고 부른다. QUIC이 제공하는 특성이 TCP의 특성과는 다르므로 변경이 필요했다. 변경사항은 다음과 같다.

- QUIC에서 전송 자체에서 스트림을 제공하지만 HTTP/2에서는 HTTP 계층에서 스트림이 제공된다.
- 스트림이 서로 독립적이므로 HTTP/2에서 사용된 헤더 압축 프로토콜을 head of line 블로킹 문제를 발생시키지 않으면서 사용할 수 없다.
- QUIC 스트림은 HTTP/2 스트림과는 다소 다르다. HTTP/3 부분에서 자세히 설명할 것이다.

HTTPS:// URL

HTTP/3는 `HTTPS://` URL을 사용해서 실행될 것이다. 세상은 이러한 URL로 가득 차 있고 새로운 프로토콜에 또 다른 URL 스킴을 도입하는 것은 실용적이지도 않고 완전히 불합리하다고 여겨졌다. HTTP/2에 새로운 스킴이 필요 없었듯이 HTTP/3에도 필요 없다.

하지만 HTTP/2가 유선으로 HTTP를 전송하는 완전히 새로운 방법이지만 HTTP/1이 그랬듯이 여전히 TLS와 TCP에 기반을 두고 있었기에 HTTP/3 상황에서 복잡성을 추가하게 되었다. HTTP/3가 QUIC을 통해 수행된다는 점은 몇 가지 중요한 관점에서 변경이 발생했다.

레거시, 평문, `HTTP://` URL은 지금 그대로 유지될 것이지만 더 안전한 전송을 하는 미래로 나아갈수록 점점 덜 사용될 것이다. 이러한 URL에 대한 요청은 HTTP/3를 사용하도록 업그레이드되지 않을 것이다. 현실에서 이러한 것들이 HTTP/2로 업그레이드하는 경우는 거의 없지만 다른 이유 때문이다.

초기 연결

이전에 방문한 적이 없는 HTTPS:// URL에 대한 새로운 첫 연결은 아마도 TCP를 통해 수행될 것이다. (추가로 QUIC을 통한 병렬 연결 시도가 있을 수 있다.) 호스트가 QUIC을 지원하지 않는 레거시 서버일 수도 있고 중간에 있는 미들박스가 QUIC 연결이 성공적으로 이뤄지지 않도록 하는 장애물일 수도 있다.

최신 클라이언트와 서버는 아마도 첫 핸드쉐이크에서 HTTP/2를 협상할 것이다. 연결이 설정되고 클라이언트의 HTTP 요청에 서버가 응답할 때 서버는 HTTP/3의 지원과 선호도에 관해 클라이언트에게 알려줄 수 있다.

Alt-svc

대체 서비스 헤더(Alt-svc:)와 이에 대응되는 `ALT-SVC` HTTP/2 프레임이 QUIC이나 HTTP/3를 위해 특별히 만들어진 것은 아니다. 서버가 클라이언트에게 "봐라. 이 포트에서 이 프로토콜로 이 호스트에서 같은 서비스를 운영하고 있다."라고 말할 수 있도록 이미 설계되고 만들어진 메커니즘의 일부이다. 자세한 내용은 RFC 7838를 참고해라.

이러한 Alt-svc 응답을 받은 클라이언트는 (이를 지원하고 원하는 경우) 주어진 다른 호스트에 지정된 프로토콜로 백그라운드에서 병렬 연결을 하도록 권고받는다. 성공적으로 전환된다면 초기 연결 대신 새로운 연결을 통해서 해당 작업을 수행한다.

초기 연결이 HTTP/2나 HTTP/1을 사용하더라도 서버는 다시 연결해서 HTTP/3를 시도할 수 있다고 클라이언트에게 알려줄 수 있다. 이는 같은 호스트일 수도 있고 요청한 내용을 제공하는 방법을 아는 다른 호스트일 수도 있다. 이러한 Alt-svc 응답에서 제공된 정보에는 만료 타이머가 있어서, 클라이언트가 일정 시간 동안 제안받은 대체 프로토콜로 직접 대체 호스트에 이어진 연결과 요청을 할 수 있다.

예시

HTTP 서버는 응답에 `Alt-Svc:` 헤더를 다음과 같이 포함한다.

```
Alt-Svc: h3=":50781"
```

이는 해당 응답을 얻는데 사용한 것과 같은 호스트 이름의 50781 UDP 포트에서 HTTP/3를 사용할 수 있음을 나타낸다.

그 다음 클라이언트는 해당 목적지에 QUIC 연결을 설정하려고 시도하고 연결이 성공하면 초기 HTTP 버전 대신 이러한 출처와 계속해서 통신한다.

QUIC 스트림과 HTTP/3

HTTP/2가 TCP를 기반으로 전체적인 스트림과 멀티플렉싱 개념을 설계해야했던 반면 HTTP/3는 QUIC을 위해 만들어졌으므로 QUIC 스트림이 가진 이점을 최대한 활용한다.

HTTP/3를 통해 수행되는 HTTP 요청은 특정 스트림 세트를 사용한다.

HTTP/3 프레임

HTTP/3는 QUIC 스트림을 설정하고 반대쪽 끝으로 프레임 세트를 보내는 것을 의미한다. HTTP/3에는 몇 가지 알려진 프레임이 고정되어 있다. 이 중 가장 중요한 것은 다음과 같다.

- HEADERS, 압축된 HTTP 헤더를 보내라.
- DATA, 바이너리 데이터 콘텐츠를 보내라.
- GOAWAY, 이 연결을 종료해라.

HTTP 요청

클라이언트는 클라이언트가 초기화한 양방향 QUIC 스트림으로 HTTP 요청을 보낸다.

단일 HEADERS 프레임으로 구성된 요청은 하나 또는 두 개의 다른 프레임, 즉 일련의 DATA 프레임과 뒤이어 오는 것들을 위한 최종 HEADERS 프레임이 따라올 수 있다.

요청을 보낸 후 클라이언트는 전송용 스트림을 닫는다.

HTTP 응답

서버는 양방향 스트림으로 해당 HTTP 응답을 돌려보낸다. 여기에는 HEADERS 프레임과 일련의 DATA 프레임이 있고 뒤따라오는 HEADERS 프레임이 있을 수 있다.

QPACK 헤더

HEADERS 프레임은 QPACK 알고리즘으로 압축된 HTTP 헤더를 담고 있다. QPACK은 HPACK이라고 부르는 HTTP/2의 압축과 형식 면에서 비슷하지만, 순서가 맞지 않게 전송된 스트림에서 동작하도록 수정되었다.

QPACK 자체는 두 엔드포인트 사이에서 추가적인 두 개의 단방향 QUIC 스트림을 사용한다. 이 스트림은 양방향으로 동적 테이블 정보를 전달하는 데 사용된다.

HTTP/3 우선순위 정하기

HTTP/3 스트림 프레임 중에는 `PRIORITY` 라는 프레임이 있다. 이 프레임은 HTTP/2에서 동작한 방식과 비슷하게 스트림의 우선순위와 의존성을 설정하는 데 사용한다.

이 프레임은 특정 스트림이 다른 스트림에 의존하도록 설정할 수 있고 해당 스트림에 "가중치"를 설정할 수 있다.

종속된 스트림은 의존하는 스트림이 모두 닫혔을 때만 리소스가 할당받아야 하고 아니면 진행될 수 없다.

스트림 가중치는 1부터 256 사이의 값을 가지며 같은 부모를 가진 스트림은 **반드시** 가중치에 따라 리소스를 할당받아야 한다.

HTTP/3 서버 푸시

HTTP/3 서버 푸시는 HTTP/2에서 설명된 내용(RFC 7540)과 비슷하지만 다른 메커니즘을 사용한다.

서버 푸시는 클라이언트가 보낸 적이 없는 요청에 대한 효율적인 응답이다!

서버 푸시는 클라이언트 쪽에서 서버 푸시에 동의했을 때만 발생할 수 있다. HTTP/3에서는 클라이언트가 최대 푸시 스트림의 ID가 무엇인지 서버에게 알려주어 클라이언트가 얼마나 많은 푸시를 받을지를 제한한다. 이 제한을 초과하면 연결 오류가 발생할 것이다.

클라이언트가 요청하지는 않았지만 어쨌든 서버가 받아야 하는 추가 리소스가 있다고 판단하면, (요청 스트림을 통해) 서버가 요청에 푸시로 응답한 것처럼 보이는 `PUSH_PROMISE` 프레임을 보낼 수 있다. 그다음 실제 응답은 새로운 스트림을 통해 보낸다.

클라이언트가 미리 푸시를 받을 수 있다고 말했더라도, 클라이언트가 적합하다고 판단하면 푸시된 개별 스트림을 언제든 취소할 수 있다. 그리고 서버에 `CANCEL_PUSH` 프레임을 보낸다.

문제점

이 기능은 HTTP/2 개발에서 처음 논의된 이후 HTTP/2 프로토콜이 나오고 인터넷에 배포된 뒤, 이 기능을 유용하게 만들기 위해 셀 수 없이 다양한 방법으로 논의되고, 싫어하게 했으며, 두들겨 맞았다.

라운드 트립의 절반을 줄일 수 있지만, 여전히 대역폭을 사용하기 때문에 푸시는 결코 "공짜"가 아니다. 실제로 리소스가 푸시되어야 하는지 아닌지를 서버 측에서 확실하게 알기가 어렵거나 불가능하다.

HTTP/3과 HTTP/2의 비교

HTTP/3는 자체적으로 스트림을 다루는 전송 프로토콜인 QUIC을 위해 설계되었다.

HTTP/2는 TCP를 위해 설계되었으므로 HTTP 계층에서 스트림을 다룬다.

유사점

이 두 프로토콜을 사실상 같은 기능을 제공한다.

- 두 프로토콜은 스트림을 제공한다.
- 두 프로토콜은 서버 푸시를 지원한다
- 두 프로토콜은 헤더 압축을 제공한다. QPACK과 HPACK은 설계상 비슷하다.
- 두 프로토콜은 스트림을 이용해서 하나의 연결을 통해 멀티플랙싱을 제공한다.
- 두 프로토콜은 스트림에 우선순위를 정한다.

차이점

세부 내용에 차이점이 있는데 주로 HTTP/3의 QUIC 사용 때문에 생긴다.

- QUIC의 0-RTT 핸드쉐이크 덕에 HTTP/3에서는 이른 데이터 지원이 더 낫게 잘 동작한다. TCP Fast Open과 TLS는 더 적은 데이터를 보내지만, 종종 문제점에 직면한다.
- HTTP/3는 QUIC 덕에 TCP + TLS보다 훨씬 더 빠른 핸드쉐이크를 제공한다.
- HTTP/3에는 안전하지 않거나 암호화되지 않은 버전이 없다. 인터넷에서 드물기는 하지만 HTTP/2는 HTTPS 없이 구현하고 사용할 수 있다.
- HTTP/2가 ALPN 확장을 이용하여 즉시 TLS 핸드쉐이크 협상을 완료할 수 있는 반면 HTTP/3는 QUIC을 사용하므로 클라이언트에 이 사실을 알리기 위해 `Alt-Svc:` 헤더 응답이 먼저 있어야 한다.

일반적인 비판

UDP는 절대 동작하지 않을 것이다

(DNS에서 사용하는) 53 포트가 아닌 UDP 트래픽이 최근에는 주로 공격에 사용되기 때문에 많은 기업, 운영자, 조직에서 차단하거나 속도를 제한하고 있다. 특히 기존의 UDP 프로토콜과 잘 알려진 서버 구현체 중 일부는 증폭 공격에 대한 취약점이 있으므로 공격자가 무고한 대상 피해자에게 대량의 트래픽을 보낼 수 있다.

QUIC에서는 초기 패킷이 최소 1200바이트여야 한다는 조건과 서버가 클라이언트로부터 응답 패킷을 받지 않으면 요청 크기의 3배 이상은 절대 보내면 안 된다는 프로토콜의 제약사항으로 증폭 공격을 완화하는 기능이 들어있다.

커널에서 UDP는 느리다

이는 적어도 2018년 현재까지 사실로 보인다. 물론 앞으로 어떻게 발전할 것인지, 수년간 UDP의 전송 성능이 개발자의 관심사가 아니었다는 결과가 간단히 어느 정도인지 말할 수 없다.

대부분 클라이언트는 이 "느림"을 결코 알아채지 못했다.

QUIC은 CPU를 너무 많이 사용한다

위에서 얘기한 "UDP는 느리다"와 비슷하게 이 또한 부분적으로는 TCP와 TLS가 세계적으로 더 오랫동안 성숙하고 개선되고 하드웨어의 지원을 받았기 때문이다.

시간이 지나면 개선되리라 기대할 근거는 있다. 문제는 추가적인 CPU 사용이 배포자에게 얼마나 영향을 끼치는가이다.

그냥 구글이다

전혀 그렇지 않다. Google이 대규모 인터넷 환경에서 증명을 마친 후 IETF에 초기 명세를 가져왔다. UDP를 통한 이 방식의 프로토콜 배포는 실제로 잘 동작한다.

그 이후 많은 회사와 조직의 사람들이 벤더 중립적 조직인 IETF에서 독립적으로 표준 전송 프로토콜을 작업했다. 물론 이 작업에 Google 직원도 참여했지만, 인터넷 전송 프로토콜의 상태를 향상하려는 Mozilla, Fastly, Cloudflare, Akamai, Microsoft, Facebook, Apple 등 많은 회사의 직원들도 참여했다.

개선된 부분이 너무 적다

이것은 정말 비판이 아니라 의견일 뿐이다. 어쩌면 HTTP/2가 나온 지 얼마 안 되었기 때문에 개선된 부분이 너무 적을 수도 있다.

HTTP/3는 패킷 손실이 많은 네트워크에서 훨씬 더 잘 동작할 것이고, 더 빠른 핸드쉐이크를 제공하므로 체감 지연시간과 실제 지연시간을 모두 개선할 것이다. 하지만, 사람들이 자신의 서버와 서비스에 HTTP/3 지원을 추가하고 싶을 정도로 장점이 충분할까? 시간과 미래의 성능 측정으로 분명히 알 수 있을 것이다!

명세

다음은 QUIC과 HTTP/3의 여러 부분 및 구성요소의 공식 드래프트 최신 버전이다.

불변

QUIC의 버전 독립적인 프로퍼티

전송

QUIC: UDP에 기반을 둔 멀티플랙싱 보안 전송

복구

QUIC 손실 탐지와 혼잡 제어

TLS

Transport Layer Security(TLS)를 사용해서 QUIC 안전하게 하기

HTTP

QUIC을 통한 Hypertext Transfer Protocol(HTTP)

QPACK

QPACK: QUIC을 통한 HTTP의 헤더 압축

QUIC v2

핵심 QUIC 프로토콜에 가장 집중해서 제때 출시할 수 있도록 핵심 프로토콜의 일부로 원래 계획되어 있던 몇 가지 기능을 연기했고 QUIC 2나 그 뒤 후속 QUIC 버전에 포함할 예정이다.

하지만 이 문서의 작성자가 잘못된 예측을 할 수도 있으므로 버전 2에 어떤 기능이 들어가고 어떤 기능이 안 들어갈지 정확하게 얘기할 수는 없다. 그래도 버전 1에서 명시적으로 제거되어 "나중에 작업"하기로 해서 버전 2에 포함될 가능성이 있는 기능은 얘기할 수 있다.

순방향 오류 정정(Forward Error Correction)

순방향 오류 정정(FEC)은 데이터 전송에서 오류 제어를 하는 방법으로 송신기는 중복 데이터를 전송하고 받는 쪽에서는 식별할 수 있는 오류가 없는 데이터의 부분만을 인식한다.

Google이 원래의 QUIC 작업에서 이를 실험했지만, 실험 결과가 좋지 않아서 후에 다시 제거되었다. 이 기능은 QUIC v2의 토론 주제이지만 너무 큰 불이익이 없고 유용한 기능이라는 것을 실제로 누군가 증명해야 할 것이다.

다중 경로

다중 경로는 리소스 사용을 최대화하고 중복을 늘리기 위해 전송 자체가 다중 네트워크 경로를 사용하는 것을 의미한다.

SCTP 지지자는 SCTP에는 이미 다중 경로 기능이 있고 오늘날의 TCP도 그렇다고 말할 것이다.

신뢰할 수 없는 데이터

"신뢰할 수 없는" 스트림을 선택사항으로 제공해서 UDP 방식의 애플리케이션도 QUIC이 대체할 수 있게 하는 것이 논의되었다.

HTTP가 아닌 프로토콜 도입

QUIC을 통한 DNS는 QUIC v1과 HTTP/3가 출시되면 관심을 끌 수 있는 HTTP가 아닌 프로토콜로 초기에 언급된 것 중 하나이다. 하지만 이 새로운 전송을 세상에 내놓으면 거기서 끝이라고 상상할 수 없다.