HTTP(HyperText Transfer Protocol)는 인터넷에서 정보를 주고받는 가장 기본적인 통신 규약이다 웹 브라우저가 웹 서버에 특정 웹 페이지를 요청하고 서버가 그 요청에 맞는 데이터를 보내주는 과정에서 HTTP 프로토콜이 사용된다.

쉽게 말해 웹사이트를 방문할 때마다 브라우저와 서버 간에 HTTP 요청과 응답이 수없이 오가는 것이다.

HTTP의 특징

1. 텍스트 기반 : HTTP 메시지는 사람이 읽을 수 있는 텍스트 형식으로 구성된다.
2. 비연결성 : 각 요청과 응답은 독립적이며 지속적인 연결을 유지하지 않는다.
3. 무상태 : 서버는 이전 요청에 대한 정보를 기억하지 않는다.
4. 클라이언트-서버 구조 : 클라이언트(브라우저)가 서버에 요청하고 서버가 응답하는 구조이다.

HTTP 주요 메서드

1. GET : 특정 자원을 요청 (웹 페이지 조회)
2. POST : 서버에 새로운 데이터를 전송 (회원 가입, 게시글 작성)
3. PUT : 특정 자원을 업데이트
4. DELETE : 특정 자원을 삭제

HTTP 요청과 응답

1. 요청 : 클라이언트가 서버에 보내는 메시지로 요청하는 자원, 사용할 메서드 등의 정보를 포함한다.
2. 응답 : 서버가 클라이언트에게 보내는 메시지로, 요청에 대한 결과(성공/실패), 데이터 등을 포함한다.

HTTP의 역할

1. 웹 페이지 전송 : 우리가 웹사이트를 방문할 때마다 HTTP를 통해 HTML, CSS, JavaScript 등의 파일을 받아와 웹 페이지를 구성한다.
2. 데이터 전송 : 웹 페이지뿐만 아니라 이미지, 동영상, 파일 등 다양한 형태의 데이터를 전송하는 데 사용된다.
3. 웹 서비스 : RESTful API를 통해 서버와 클라이언트 간의 데이터 교환을 가능하게 한다.

HTTP 버전 별 정리

HTTP/1.0

웹 통신의 가장 초기 버전으로 웹의 발전에 중요한 역할을 했지만 현재는 거의 사용되지 않는다.

HTTP/1.0의 특징

1. 비지속 연결 : 한 번의 요청과 응답이 끝나면 TCP 연결이 끊어졌습니다. 매번 새로운 연결을 맺어야 했기 때문에 성능이 좋지 않다.
2. 단순한 요청/응답 : 하나의 요청에 대해 하나의 응답만 처리할 수 있었다.
3. 제한적인 메서드 : GET, HEAD, POST 등 기본적인 메서드만 지원했다.
4. 헤더 필드: HTTP/1.0에서는 요청과 응답에 다양한 헤더를 추가할 수 있게 되어 요청에 대한 추가 정보(예: 콘텐츠 유형, 서버 정보 등)를 전달할 수 있다.

HTTP/1.0의 한계

HTTP/1.0의 주요 한계는 비효율적인 연결 관리로 인해 대규모 트래픽을 처리하는데 적합하지 않다는 점이다. 이후 버전인 HTTP/1.1에서는 이를 개선하여 연결 재사용 및 더 많은 기능을 추가되었다.

HTTP/1.0과 HTTP/1.1의 비교

기능	HTTP/1.0	HTTP/1.1
연결	비지속 연결	지속 연결
요청	단일 요청	여러 요청 (파이프라이닝)
메서드	제한적	다양한 메서드 (PUT, DELETE 등)
캐싱	기본적	강화된 캐싱

HTTP/1.0은 웹 통신의 초기 단계에서 중요한 역할을 했지만 현재는 거의 사용되지 않다. HTTP/1.1과 HTTP/2 등 더욱 발전된 프로토콜이 등장하면서 HTTP/1.0의 단점은 대부분 해결되었고 웹 성능과 기능이 크게 향상되었다.

HTTP/1.1

웹 통신의 기본 프로토콜인 HTTP의 주요 버전 중 하나이다. HTTP/1.0의 한계를 극복하고 웹 성능을 향상시키기 위해 많은 기능들이 추가되었다.

HTTP/1.1의 특징

1. 지속 연결 :
    1. 하나의 TCP 연결을 유지하여 여러 요청과 응답을 주고받을 수 있다.

    2. 매번 새로운 연결을 맺는 오버헤드를 줄여 응답 시간을 단축시킨다.
    3. `Connection: keep-alive` 헤더를 통해 지속 연결을 요청한다.
2. 파이프라이닝 :
    1. 지속 연결을 활용하여 여러 요청을 순서대로 보낼 수 있다.
    2. 서버는 요청을 받는 순서대로 응답하지 않을 수 있으므로 순서 보장 문제가 발생할 수 있다.
3. 새로운 메서드 :
    1. PUT, DELETE 등 새로운 메서드가 추가되었다.
4. 캐싱 :
    1. 캐싱 메커니즘이 강화되어 동일한 자원에 대한 반복적인 요청을 줄이고 응답 시간을 단축시킨다.
5. 오류 처리 :
    1. 정확하고 다양한 오류 코드를 제공하게 되었다.

HTTP/1.1의 한계

1. 헤더 오버헤드 :
    1. HTTP/1.1의 요청과 응답은 텍스트 기반의 헤더를 포함한다. 이 헤더들은 각 요청마다 전송되며, 종종 동일한 정보가 반복됩니다. 예를 들어, `User-Agent`, `Host`, `Accept-Language` 등의 헤더는 여러 요청에 반복적으로 포함되며, 이러한 반복적인 전송은 네트워크 자원을 낭비하게 만든다. 모바일 네트워크나 대역폭이 제한된 환경에서는 이 오버헤드가 성능 저하의 주요 원인이 될 수 있다.
2. HEAD 블로킹 :
    1. HTTP/1.1에서 클라이언트는 한 번에 하나의 요청만을 처리할 수 있다. 이로 인해 앞선 요청이 완료되지 않으면 뒤따르는 모든 요청이 대기 상태에 놓이게 된다. 이를 "헤드 오브 라인 블로킹(Head-of-Line Blocking)"이라고 부른다. 이미지, 스크립트 등 여러 리소스를 동시에 요청해야 하는 상황에서 첫 번째 요청이 지연되면 이후의 요청들도 연쇄적으로 지연되며 전체 페이지 로딩 시간이 길어진다.
3. TCP 오버헤드 :
    1. HTTP/1.1에서는 기본적으로 각 도메인에 대해 하나의 TCP 연결을 유지한다. 그러나 각 연결에는 설정과 종료 과정에서 TCP의 3-way handshake와 같은 추가적인 오버헤드가 발생한다. 또한 HTTP/1.1에서는 Keep-Alive 기능을 통해 TCP 연결을 재사용할 수 있지만 이 방식에서도 연결 유지와 관련된 비용이 존재한다. 다수의 요청이 있을 때 개별적인 TCP 연결을 관리하는 데 따른 자원 소모가 커지게 된다.
4. 파이프라이닝의 한계 :
   HTTP/1.1에서는 성능을 개선하기 위해 파이프라이닝이라는 기술을 도입했다. 이는 하나의 연결에서 여러 개의 요청을 순차적으로 보내고, 서버로부터 순서대로 응답을 받는 방식이다. 파이프라이닝은 몇 가지 심각한 한계를 가지고 있습니다
   1. 순서 보장 문제 : 파이프라이닝에서는 요청이 순서대로 처리되어야 한다. 만약 앞선 요청

의 응답이 지연되면 뒤따르는 모든 요청의 응답도 지연된다. 이로 인해 성능 저하가 발생할 수 있다.
2. 헤더 차단 문제 : 파이프라이닝 사용 시 앞선 요청의 헤더가 뒤따르는 요청의 처리를 막을 수 있는 문제가 발생한다. 특히 서버가 요청을 처리하는 데 시간이 오래 걸릴 경우 전체 파이프라인의 효율성이 저하된다.
3. 지원 부족 : 파이프라이닝은 이론적으로 성능을 향상시킬 수 있지만 실제로는 많은 서버와 클라이언트가 이를 완전히 지원하지 않거나 비활성화하고 있다. 이는 파이프라이닝의 사용을 제한하고 결국 기대했던 성능 향상이 이루어지지 않는 결과가 된다.

HTTP/1.1과 HTTP/2 비교

기능	HTTP/1.1	HTTP/2
프레임	텍스트 기반	바이너리 기반
헤더	압축 없음	HPACK 압축
멀티플렉싱	파이프라이닝 (제한적)	스트림 기반 멀티플렉싱
서버 푸시	지원하지 않음	지원

HTTP/1.1은 웹 통신의 기본을 다지고 웹 성능 향상에 크게 기여했다. 하지만 HTTP/2의 등장으로 인해 점차 그 중요성이 줄어들고 있다.

HTTP/2

HTTP/1.1의 한계를 극복하고 웹 페이지 로딩 속도를 비약적으로 향상시킨 차세대 HTTP 프로토콜이다. 기존의 텍스트 기반 프로토콜에서 벗어나 이진 형식으로 데이터를 전송하며 다양한 최적화 기술을 통해 웹 성능을 획기적으로 개선됐다.

HTTP/2의 특징

1. 바이너리 프레임 :
    1. HTTP/2는 텍스트 기반의 HTTP/1.1과 달리, 바이너리 프레이밍 계층을 사용한다. 이는 요청과 응답을 바이너리 포맷으로 변환하여 전송함으로써 데이터 처리와 파싱이 더 빠르고 효율적으로 이루어지도록 한다.
2. 헤더 압축 : \
    1. HTTP/2는 헤더 데이터를 효율적으로 전송하기 위해 HPACK이라는 헤더 압축 기법을 사용한다. HPACK은 요청과 응답에서 반복되는 헤더 필드를 효율적으로 압축하여 네트워크 대역폭 사용을 줄인다. 이로 인해 헤더 오버헤드가 크게 감소하며 특히 모바일 환경에서 성능 개선에 기여한다.

3. 멀티플렉싱 :
    1. HTTP/2의 가장 큰 특징 중 하나는 멀티플렉싱 기능이다. HTTP/1.1에서는 하나의 TCP 연결에서 한 번에 하나의 요청만 처리할 수 있었던 반면 HTTP/2는 하나의 연결에서 여러 요청과 응답을 동시에 주고받을 수 있다. 각각의 요청과 응답은 개별적인 스트림으로 처리되며 스트림 간의 차단 없이 병렬로 전송된다. 이는 "헤드 오브 라인 블로킹" 문제를 효과적으로 해결한다.
4. 서버 푸시 :
    1. HTTP/2는 서버 푸시 기능을 지원한다. 서버 푸시는 클라이언트가 요청하지 않은 리소스를 서버가 미리 전송할 수 있게 한다. 클라이언트가 HTML 문서를 요청하면 서버는 클라이언트가 곧 필요로 할 CSS 파일이나 JavaScript 파일을 함께 전송할 수 있다. 이는 페이지 로딩 속도를 크게 개선할 수 있다.
5. 스트림 우선순위 :
    1. HTTP/2는 각 스트림에 우선순위를 설정할 수 있는 기능을 제공한다. 클라이언트는 중요한 리소스(초기 렌더링에 필요한 CSS나 JavaScript 파일)에 높은 우선순위를 부여할 수 있다. 서버는 이 우선순위를 고려하여 리소스를 전송함으로써 사용자 경험을 최적화할 수 있다.
6. 연결 관리 개선 :
    1. HTTP/2는 단일 연결을 통해 모든 데이터를 주고받기 때문에 TCP 연결 수를 최소화할 수 있다. 이는 연결 설정과 종료에 따른 오버헤드를 줄이고 네트워크 자원의 효율적인 사용을 가능하게 한다. 특히 TLS(HTTPS)와 결합했을 때, 성능상의 이점이 더욱 두드러집니다.
7. 보안 :
    1. HTTP/2는 HTTPS와 함께 사용되는 경우가 많으며 TLS를 통해 데이터를 암호화한다. HTTP/2 자체는 보안을 필수로 요구하지 않지만 대부분의 브라우저는 HTTP/2를 HTTPS로만 사용하도록 강제하고 있다. 이는 웹 트래픽의 보안성을 높이는 데 기여한다.

HTTP/2의 한계

1. TCP의 한계 : HTTP/2는 기본적으로 TCP 위에 구축되어 있다. TCP는 연결 지향형 프로토콜로 연결 설정에 시간이 소요되고 네트워크 환경 변화에 민감하게 반응한다. 모바일 환경에서 발생하는 네트워크 끊김이나 혼잡은 HTTP/2의 성능 저하가 될 수 있다.
2. 헤드 오브 라인 블로킹 : 하나의 TCP 연결에서 여러 요청이 순차적으로 처리되기 때문에 앞쪽 요청이 지연되면 뒤쪽 요청도 함께 지연되는 현상이 발생한다 이는 특히 많은 자원을 사용하는 웹 페이지에서 성능 저하를 초래할 수 있다.
3. 연결 설정 오버헤드 : TCP 연결을 설정하는 데 필요한 시간은 무시할 수 없으며 특히 많은 수의 작은 요청을 처리하는 경우 성능에 영향을 미칠 수 있다.

HTTP/2와 HTTP/3의 비교

특징	HTTP/2	HTTP/3
기반 프로토콜	TCP	QUIC (UDP 기반)
헤더 압축	HPACK	HPACK 유사
멀티플렉싱	하나의 TCP 연결 내에서 여러 스트림	하나의 QUIC 연결 내에서 여러 스트림
흐름 제어	흐름 제어 프레임	흐름 제어 프레임
연결 설정	TCP 핸드셰이크	TLS 핸드셰이크 (0-RTT 가능)
오류 복구	TCP 재전송	QUIC의 자체적인 오류 복구 메커니즘
보안	TLS	TLS (기본적으로 암호화)

HTTP/2는 웹 성능을 획기적으로 향상시킨 프로토콜로 웹 개발에서 필수적인 기술이다. HTTP/2를 도입하면 웹 사이트의 로딩 속도를 빠르게 개선하고 사용자 경험을 향상시킬 수 있다.

HTTP/3

HTTP 프로토콜의 최신 버전으로 HTTP/2의 한계를 극복하기 위해 개발되었다. 2018년에 처음 도입되었으며 HTTP/3는 기존의 TCP 대신 QUIC(Quick UDP Internet Connections)라는 새로운 전송 프로토콜을 기반으로 하여 성능과 안정성을 크게 향상시켰다. 특히 모바일 환경에서 발생하는 네트워크 변동에 강하고 더 낮은 지연 시간을 제공하여 사용자 경험을 향상 시켰다.

HTTP/3의 특징

1. QUIC 프로토콜 기반 :
    1. HTTP/3의 가장 큰 특징은 전송 계층에서 TCP 대신 QUIC을 사용하는 것이다. QUIC은 UDP(사용자 데이터그램 프로토콜) 위에서 작동하며 TCP의 기능을 제공하면서도 여러 가지 장점을 추가 되었다.
        1. 빠른 연결 설정 : QUIC은 TCP와 달리 핸드셰이크 과정을 최소화하여 더 빠르게 연결을 설정할 수 있다. 특히 TLS 암호화와 연결 설정을 동시에 처리하여 지연 시간을 줄인다.
        2. 헤드 오브 라인 블로킹 문제 해결 : TCP는 패킷이 손실되면 그 패킷이 복구될 때까지 모든 후속 패킷을 기다려야 한다. 이로 인해 성능 저하가 발생하는데 이를 "헤드 오브 라인 블로킹"이라고 한다. QUIC은 각 스트림이 독립적으로 관리되기 때문에 특정 스트림에서 패킷 손실이 발생하더라도 다른 스트림의 데이터 전송에 영향을 미치지 않는다.
2. 멀티플렉싱과 스트림 관리 :

    1. HTTP/3는 HTTP/2와 마찬가지로 멀티플렉싱을 지원하지만 스트림 관리가 더 효율적이다. QUIC은 각 스트림을 독립적으로 처리하여 데이터가 비동기적으로 전송될 수 있다. 이는 HTTP/2의 멀티플렉싱에서 발생할 수 있는 헤드 오브 라인 블로킹 문제를 해결하는 데 크게 기여한다.
3. 보안과 암호화 :
    1. HTTP/3는 QUIC 위에서 작동하며 QUIC은 기본적으로 TLS 1.3을 사용하여 모든 데이터를 암호화한다. 이 암호화는 기본적으로 활성화되며 QUIC 연결을 통해 주고받는 모든 데이터는 안전하게 보호된다. QUIC의 설계 자체에 보안이 포함되어 있기 때문에 HTTP/3는 추가적인 보안 계층 없이도 높은 수준의 보안을 제공한다.
4. 패킷 손실에 대한 더 나은 처리 :
    1. QUIC은 패킷 손실에 민감하지 않다. 전송된 데이터 중 일부 패킷이 손실되더라도 나머지 패킷은 정상적으로 처리될 수 있다. QUIC은 손실된 패킷만을 재전송하고 나머지 데이터 스트림은 중단 없이 계속 처리되기 때문에 연결 성능이 향상된다.

HTTP/3의 한계

1. 네트워크 환경과의 호환성 : QUIC은 UDP 기반이기 때문에 일부 방화벽이나 네트워크 장비가 UDP 트래픽을 제한하거나 차단할 수 있다.
2. 복잡한 구현 : QUIC과 HTTP/3의 복잡한 구조는 서버와 클라이언트 모두에서 구현과 관리에 많은 노력이 필요한다.
3. 구형 장치와의 호환성 : HTTP/3는 최신 기술이기 때문에 오래된 장치나 소프트웨어에서는 지원되지 않을 수 있다.

HTTP/3는 웹의 요구에 부응하는 프로토콜로 특히 대규모 웹 애플리케이션과 실시간 통신이 필요한 서비스에서 좋은 성능을 발휘한다. QUIC의 도입으로 인해 성능, 보안, 효율성 면에서 많은 개선이 이루어졌으며 HTTP/2의 한계를 효과적으로 극복하고 있다.

HTTPS (HyperText Transfer Protocol Secure)

웹에서 데이터를 안전하게 전송하기 위해 사용되는 프로토콜이다. HTTP의 보안 버전으로 주로 웹사이트와 서버 간의 데이터를 암호화하여 제3자가 이를 읽거나 변조하는 것을 방지한디. HTTPS는 특히 인터넷에서 개인정보를 보호하고 안전한 통신을 보장하는 데 중요한 역할을 한다.

HTTPS의 구성 및 특징

1. TLS (Transport Layer Security) / SSL (Secure Sockets Layer) :

    1. TLS : HTTPS는 TLS프로토콜을 사용하여 데이터를 암호화한다. TLS는 SSL의 후속 버전으로 보안성과 성능이 개선된 프로토콜이다. HTTPS는 서버와 클라이언트 간의 통신을 암호화하고 데이터의 무결성을 보장하기 위해 TLS를 사용된다.
    2. SSL : TLS의 초기 버전이며 현재는 TLS가 대부분의 보안 통신에서 사용되고 있다. SSL의 최신 버전은 SSL 3.0이지만 웹에서는 TLS 1.2와 TLS 1.3이 주로 사용된다.
2. 암호화 :
    1. 대칭 암호화 : 데이터 전송 중 실시간으로 데이터를 암호화하는 방식이다. 암호화와 복호화에 같은 키를 사용된다.
    2. 비대칭 암호화 : 공개 키와 개인 키의 쌍을 사용하여 데이터를 암호화한다. 공개 키로 암호화된 데이터는 개인 키로만 복호화할 수 있다.
3. 인증서 :
    1. HTTPS는 디지털 인증서를 사용하여 서버의 신원을 확인한다. 인증서는 인증 기관에서 발급되며 웹사이트의 도메인과 기업 정보를 포함한다. 클라이언트는 인증서를 통해 서버가 신뢰할 수 있는지 확인할 수 있다.
    2. SSL/TLS 인증서 : 인증서는 공개 키 암호화를 지원하며 웹사이트의 신원을 검증한다. 인증서에는 발급자의 정보, 인증서 소유자의 정보, 유효 기간 등이 포함된다.
4. 핸드셰이크 과정 :
    1. TLS 핸드셰이크 : 클라이언트와 서버 간의 보안 연결을 설정하기 위한 과정이다. 이 과정에서 암호화 방식과 세션 키가 협상되며 서버는 인증서를 클라이언트에게 전송된다. 클라이언트는 인증서를 검증하고 서버와의 안전한 통신을 위한 세션 키를 생성한다.

HTTPS의 작동 방식

1. 클라이언트와 서버 연결 : 클라이언트(웹 브라우저)는 서버에 HTTPS 요청을 보낸다. 서버는 SSL/TLS 인증서를 클라이언트에 제공하여 자신의 신원을 증명한다.
2. TLS 핸드셰이크 : 클라이언트와 서버 간의 핸드셰이크 과정이 시작된다. 이 과정에서 암호화 방식과 세션 키를 협상하며 클라이언트는 서버의 인증서를 검토하여 신뢰성을 확인한다.
3. 암호화된 통신 : 핸드셰이크가 완료되면 클라이언트와 서버 간의 데이터 전송은 암호화된다. 이를 통해 데이터가 중간에서 가로채이거나 변조되는 것을 방지할 수 있다.
4. 세션 종료 : 통신이 종료되면 세션 키와 같은 암호화 정보는 폐기된다. 이후의 통신은 새로운 핸드셰이크 과정으로 시작된다.

HTTPS의 사용 이유

1. 데이터 암호화 : HTTPS는 데이터 전송 시 암호화를 통해 개인 정보와 중요한 데이터를 보호한다. 이로 인해 데이터가 중간에서 도청되거나 변경되는 것을 방지할 수 있다.
2. 서버 인증 : 디지털 인증서를 사용하여 서버의 신뢰성을 검증함으로써 사용자에게 안전한 웹사이트 접속을 보장된다.
3. 데이터 무결성 : 데이터 전송 중에 내용이 변경되지 않았음을 확인할 수 있다. HTTPS는 데이터가 전송되는 동안 손상되거나 변조되는 것을 방지한다.

4. 사용자 신뢰 : 구글과 같은 검색 엔진은 HTTPS를 사용하는 웹사이트에 대해 더 높은 순위를 부여한다. 웹사이트 방문자는 HTTPS를 통해 보안이 강화된 사이트에 더 많은 신뢰를 가진다.

HTTPS와 HTTP의 차이

1. 보안 : HTTP는 데이터를 암호화하지 않지만 HTTPS는 TLS를 사용하여 데이터를 암호화하고 보호한다.
2. 포트 : HTTP는 기본적으로 포트 80을 사용하며 HTTPS는 포트 443을 사용한다.
3. URL 표시 : HTTPS 웹사이트의 URL은 `https://` 로 시작하며 HTTP는 `http://` 로 시작한다. HTTPS 웹사이트는 종종 브라우저 주소창에 자물쇠 아이콘을 표시하여 보안 상태를 나타낸다.

웹 기술의 발전에 따라 HTTPS는 계속해서 발전하고 있으며 HTTP/3와 같은 최신 프로토콜은 더욱 향상된 성능과 보안을 제공한다. 앞으로도 HTTPS는 웹의 신뢰성과 보안을 유지하는 데 필수적인 역할을 할 것이다.

추가 정리

HTTP/3는 기본적으로 QUIC 프로토콜 위에서 작동하며 QUIC은 모든 데이터를 전송할 때 TLS 1.3을 사용하여 데이터를 암호화한다. 이로 인해 HTTP/3 통신은 자동으로 암호화되어 안전한 전송이 보장된다.
그러나 HTTP/3가 TLS 1.3을 사용한다는 사실이 HTTPS가 필요 없다는 것을 의미하지는 않는다. 오히려 HTTP/3는 일반적으로 HTTPS를 통해 사용된다고 한다.

이유

1. 보안 표준 : HTTPS는 웹에서 보안 통신을 위한 표준이다. HTTP/3가 TLS 1.3을 사용하더라도 그 통신이 "HTTPS"로 명시되어야 웹 브라우저와 서버가 이를 안전한 연결로 인식하고 처리한다. HTTP/3도 여전히 "https://" 프로토콜을 사용하며 이 표기 방식이 웹에서의 보안 통신을 의미한다.
2. 브라우저 및 서버 호환성 : 대부분의 웹 브라우저와 서버는 HTTP/3를 지원할 때 이를 HTTPS와 함께 사용하도록 설계되어 있다. 웹 서버와 브라우저는 "https://"를 통해 접속 요청이 들어오면 HTTP/3 프로토콜을 사용해 연결을 설정한다.
3. 사용자 경험 : 사용자는 HTTPS를 통해 웹사이트에 접속할 때 URL에 "https://"를 기대한다. 이는 사용자가 연결이 안전하다는 확신을 가지는 데 중요한 역할을 한다. HTTP/3가 암호화를 기본으로 한다고 해도 HTTPS는 여전히 보안의 상징이며 웹 사용자의 신뢰를 유지하는 데 필요한다.

요약

HTTP/3는 TLS 1.3을 통해 모든 데이터를 암호화하지만 여전히 HTTPS가 필요합니다. HTTP/3를 사용할 때 HTTPS는 안전한 웹 통신의 상징으로 계속 사용되며 브라우저와 서버 모두에서 이를 통해 보안 통신이 이루어진다 따라서 HTTP/3 환경에서도 HTTPS는 여전히 필수적이다.