(3) HTTP/2의 특징은 무엇이며, HTTP/1.1과의 차이점은 무엇인가?

HTTP/1.1 는 연결 하나당 하나의 요청과 응답을 처리하기에 동시에 전송하는 것과 많은 리소스를 순차적으로 처리하기 때문에 동시에 처리하는 것이 어렵다. 이러한 특징 때문에 발생하는 문제점은 다음과 같다.

* HOL(Head-Of-Line) Blocking - TCP 프로토콜은 체인처럼 연결된 형태로 순차적으로 데이터를 전송하는데, 첫 번째 패킷이 손실되거나 분실된 경우 그 패킷을 다시 찾거나 서버로부터 재전송을 받기 전까지 다음 패킷이 기다려야 하는 현상을 말한다. 특정 응답의 지연 현상.
* RTT(Round Trip Time)의 증가 - TCP 프로토콜 위에서 동작하는 HTTP의 특성 상, 3-way Handshake가 반복적으로 발생하는데 이는 곧 불필요한 RTT 증가와 네트워크 지연을 초래하여 성능을 떨어뜨린다.
* 무거운 헤더 구조 - HTTP/1.1의 헤더에는 많은 메타 정보들이 저장되어 있는데, 요청시마다 중복되어 헤더값을 전송한다.

이러한 문제점들을 개선하기 위해 나온 것이 HTTP/2이다. HTTP/2에서는 HTTP/1.1과 달리, 한 커넥션에 여러 개의 메시지를 동시에 주고받을 수 있고(multiplexing streams), 서버 푸시를 하며, 헤더 압축을 하여 전송 지연 시간을 획기적으로 감소시켰다. 정보를 전달하는 과정을 음식에 비유하자면 HTTP/1.1이 코스 요리라면 HTTP/2는 한 상 차림인 셈이다. 그만큼 속도 측면에서 확실히 개선되었음을 알 수 있다. 단, 이러한 점은 표준 프로토콜인 HTTP/1.1를 대체하는 것이 아니라 확장한다는 것이다.

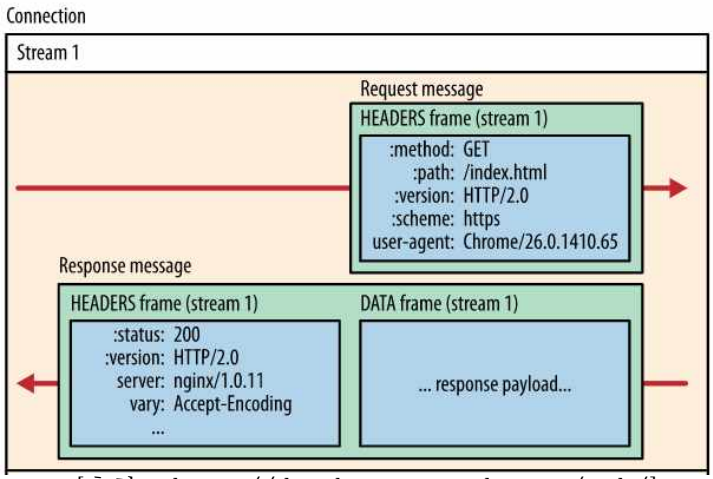
#TCP 3-way Handshake : TCP/IP 프로토콜을 이용해서 통신을 하는 응용프로그램이 데이터를 전송하기 전에 먼저 정확한 정보 전송을 보장하기 위해 상대방 컴퓨터와 사전에 세션을 수립하는 과정을 의미한다. 이러한 절차는 TCP 접속(연결)을 성공적으로 하기 위해 반드시 필요하며 그 과정은 다음과 같다.
  클라이언트 -> 서버 : TCP SYN
  서버 -> 클라이언트 : TCP SYN ACK
  클라이언트 -> 서버 : TCP ACK
handshake : 정보기술과 전기통신 및 관련 분야에서 채널에 대한 정상적인 통신이 시작되기 전에 두 개의 실체 간에 확립된 통신 채널의 변수를 동적으로 설정하는 자동화된 협상 과정이다. 정상적인 정보 전송 이전에 이뤄지며 채널의 물리적인 확립이 따른다.

HTTP/2의 각 특징에 대해 살펴보면 다음과 같다.

- 바이너리 프레이밍 계층(Binary Framing Layer)

HTTP/2에서 가장 중요한 특징으로, HTTP/1.1에서 줄바꿈으로 구분되던 일반 텍스트와 달리 프레임을 단위로 하여 분할되고 바이너리 형식으로 인코딩되어 전달된다. 즉, 정보를 캡슐화하여 전송하는 것이다. HTTP/2에서는 프레임(frame)이 가장 작은 단위가 되고 최소 하나의 프레임 헤더가 포함된다. 요청이나 응답 메시지에 대응되는 프레임의 전체 시퀀스를 메시지라고 하며 이 메시지는 스트림 위에 올라가서 전달된다. 이때 스트림은 양방향성 연결을 갖는다. 다음 그림을 참고하면 프레임, 메시지, 스트림, 커넥션 간의 관계를 파악할 수 있다.

이러한 기능은 HTTP/1.1에서 발생하는 HOL Blocking 문제를 해결할 수 있으며 하나의 커넥션으로도 여러 응답과 요청을 처리할 수 있다. 즉, 하나의 출처(origin)는 하나의 커넥션으로 충분하다. 그 결과 TCP 커넥션을 더욱 효율적으로 사용할 수 있으며 전체적으로 보면 사용되는 커넥션이 감소했으므로 전체 연결 경로에서 메모리와 처리량이 줄어드는 셈이다. 이는 곧 운영 비용의 절감으로 연결된다.



[출처 : https://developers.google.com/web/]

- 다중화(Multiplexing-멀티플렉싱)

바이너리로 인코딩된 프레임들은 특정한 스트림에 속하게 되고 이러한 스트림이 하나의 커넥션 안에서 멀티플렉싱하게 동작하므로 동시에 여러 개의 정보 전달을 수행할 수 있다. HTTP/1.1에서는 여러 병렬 요청을 처리하려며 그만큼 여러 개의 커넥션을 사용해야 했으나, HTTP/2의 멀티플렉싱은 연결 하나당 여러 병렬 요청을 처리하는 것이 가능하다. 이때 HTTP 메시지를 독립된 프레임으로 세분화하고 이 프레임을 인터리빙(교차되어 정리된 것)한 다음, 다른 쪽에 다시 재조립하는 기능은 여러 요청이나 응답을 병렬 처리 가능하며 하나의 커넥션으로 전달할 수 있다. 이러한 기능은 지연 시간을 축소하여 네트워크 용량의 활용성을 개선시킨다.

- **스트림의 우선순위 지정(Stream Prioritization)**

HTTP/1.1에서는 순차적으로 정보를 처리했다면 HTTP/2에서는 프레임들로 구성된 메시지를 담은 스트림의 가중치에 따라 처리된다. 각 스트림은 가중치와 다른 스트림에 대한 명시적 종속성을 가지며 클라이언트가 '우선순위 지정 트리'를 구성하고 통신할 수 있다. 서버는 클라이언트가 보낸 우선순위를 바탕으로 CPU, 메모리 및 기타 리소스의 할당을 제어함으로써 스트림 처리의 우선순위를 지정한다. 응답 메시지를 처리할 때도 해당 요청의 가중치에 따라 우선순위를 지정한다. 단, 클라이언트는 특정 순서로 스트림을 처리하도록 서버에게 강요할 수는 없다.

- **흐름 제어(Flow Control)**

흐름 제어 메커니즘을 사용하면 필요 이상으로 데이터가 크거나 처리 불가능한 데이터를 수신기가 받는 것을 막을 수 있다. 예를 들어서 프록시 서버의 다운스트림 커넥션은 빠르고 업스트림 커넥션이 느린 경우, 프록시 서버가 업스트림 속도에 맞게 다운스트림의 데이터 전달 속도를 조절하여 리소스 사용량을 제어한다. 흐름 제어는 양방향이고 비활성화될 수 없다. 흐름 제어 안의 기본값은 65,535바이트로 설정되지만 데이터가 수신될 때마다 수신기가 window-update frame을 전송하여 창의 최대 크기(2^31-1 바이트)를 설정하고 유지할 수 있다. 흐름 제어는 홉(Hop-by-Hop)방식을 따르는데 각 패킷이 매 노드(또는 라우터)를 건너뛰면서 전달되는 것을 뜻한다. 자체적으로 최적의 경로를 찾아내어 수신 패킷을 다음 라우터에 전달한다.

프록시 서버(proxy) : 클라이언트와 서버 간의 중간 매개체 역할을 한다. 프록시 서버에 요청된 내용들을 캐시를 이용하여 저장해두고 이러한 캐시 안에 담긴 내용을 요청받을 경우, 굳이 서버까지 가지 않더라도 캐시를 전달함으로써 전송 시간을 줄일 수 있고 불필요한 서버와의 연결을 할 필요가 없으므로 효율적이다. 이러한 특징은 원치 않은 사이트를 차단하거나 역으로 IP추적을 당하지 않게 하거나 전달받거나 전달할 데이터를 검사하는 등에 이용될 수 있다.

- **서버 푸시(Server Push)**

HTTP/2가 HTTP/1.1과 다른 점 중에 하나로 서버가 클라이언트 요청에 해당하는 응답뿐만 아니라 여러 응답을 보낼 수 있는 기능을 말한다. 서버 측에서는 클라이언트에게 어떤 리소스가 필요한지 알고 있기에, 클라이언트가 모든 리소스를 확인하여 서버에게 요청하는 과정을 덜어주는 격이다. 즉, 클라이언트가 할 일을 서버가 대신 해주는 셈이다. 서버 푸시 리소스에는 클라이언트에 의해 캐시된 것, 다른 페이지에서 재사용된 것, 다른 리소스와 함께 멀티플렉싱(다중화)된 것, 서버에서 우선 순위가 지정된 것, 클라이언트에 의해 거부된 것 등이 있다. HTTP/2에서 클라이언트는 서버 푸시의 사용 방식을 제어하는데 푸시되는 스트림의 수를 제한하거나 완전히 비활성화시킬 수 있다.

- **헤더 압축(Head Compression)**

HTTP/1.1에서 HTTP 전송 시, 헤더 세트를 함께 전달하는데 이 메타데이터는 항상 일반 텍스트로 전송되며 기본 500-800 바이트의 오버헤드가 추가되는데 쿠키를 사용할 경우 더 커진다. 이 오버헤드를 줄이기 위해 HTTP/2에서는 HPACK 압축 형식을 사용하여 HTTP 메시지의 헤더 부분을 압축한다. 이 방식은 이전에 전송했던 메시지의 헤더 부분과 중복되는 부분은 생략하고 차이가 있는 부분만 전송한다. 이러한 코딩은 Huffman 코딩이라 하며, 이전에 표시된 헤더 필드의 인덱스 리스트를 레퍼런스로 사용하여 이전에 전송된 값을 효율적으로 인코딩할 수 있다.