HTTP/2 세대의 Java

2017-11-26

정상혁

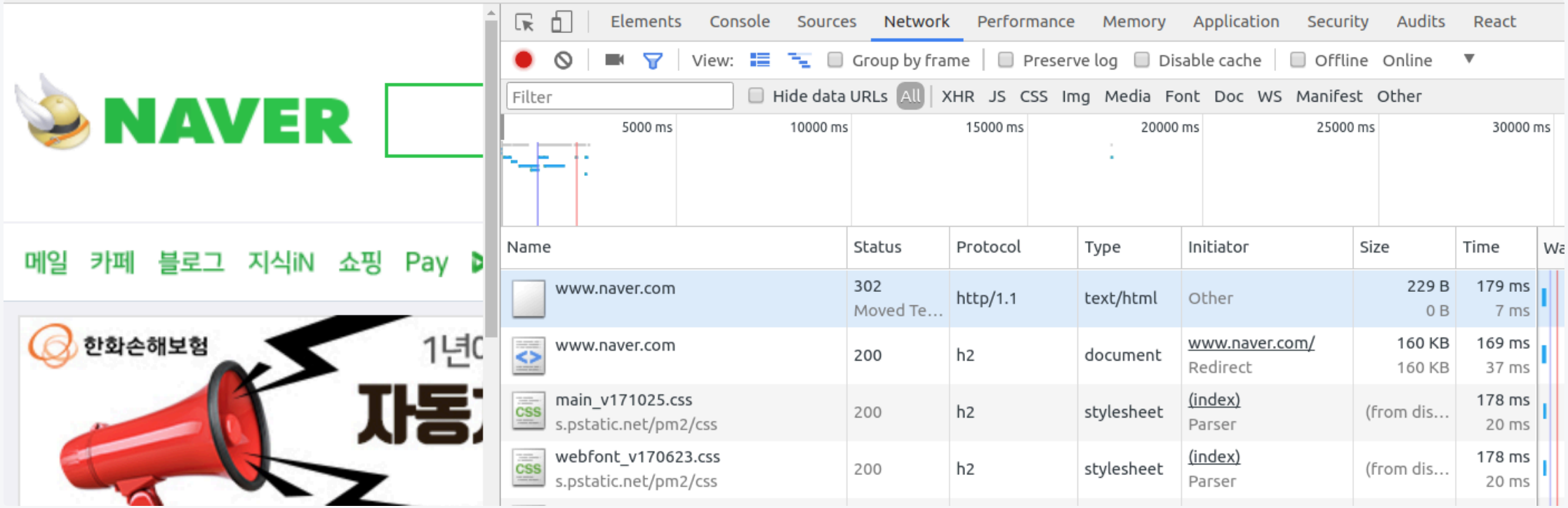
발표자 소개

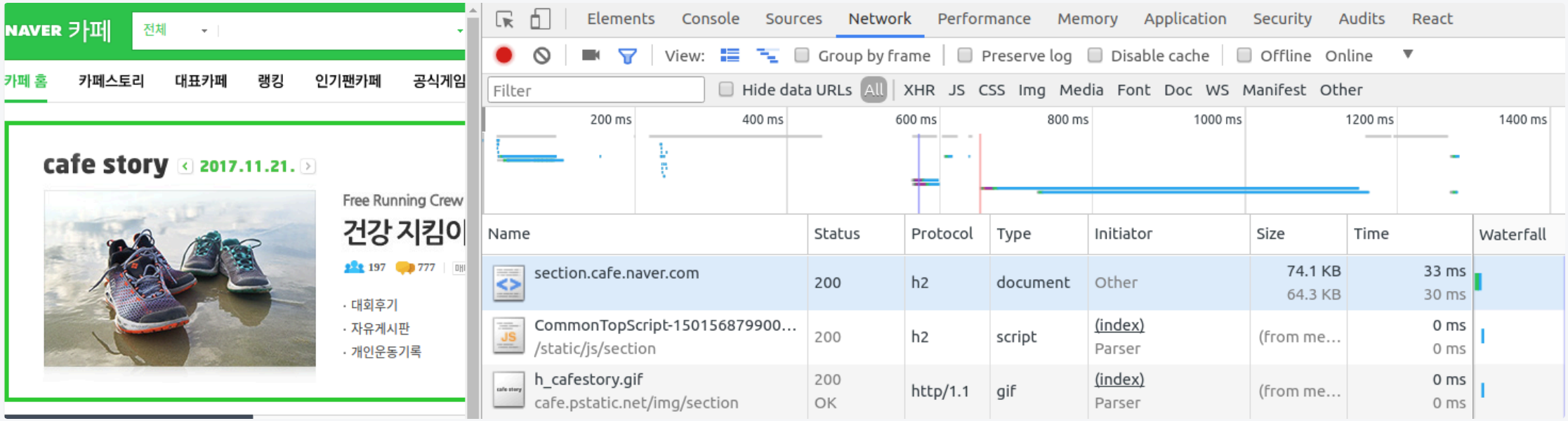
- KSUG 3기 일꾼단 활동
- 네이버 PaaS 소속
- 하는 일
  - 협업 플랫폼 개발
  - 사내 공통 프레임워크 기술 지원
  - 네이버 개발자센터(developers.naver.com) 개선
- 네이버 기술 블로그 (d2.naver.com) 에 가끔 기고

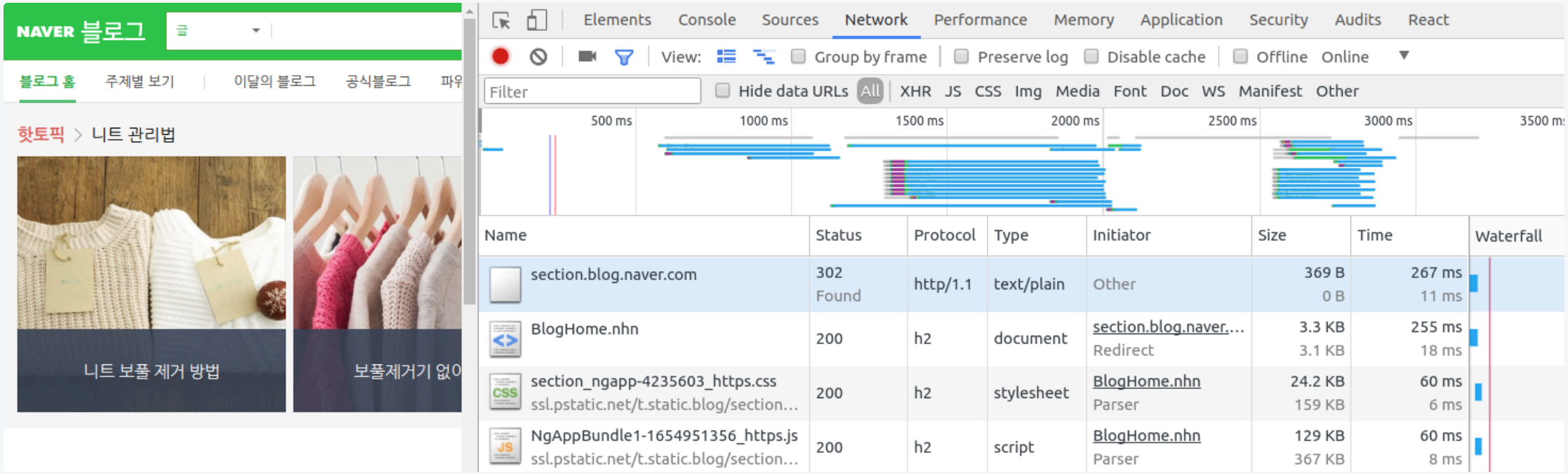
HTTP/2 세대의 이상

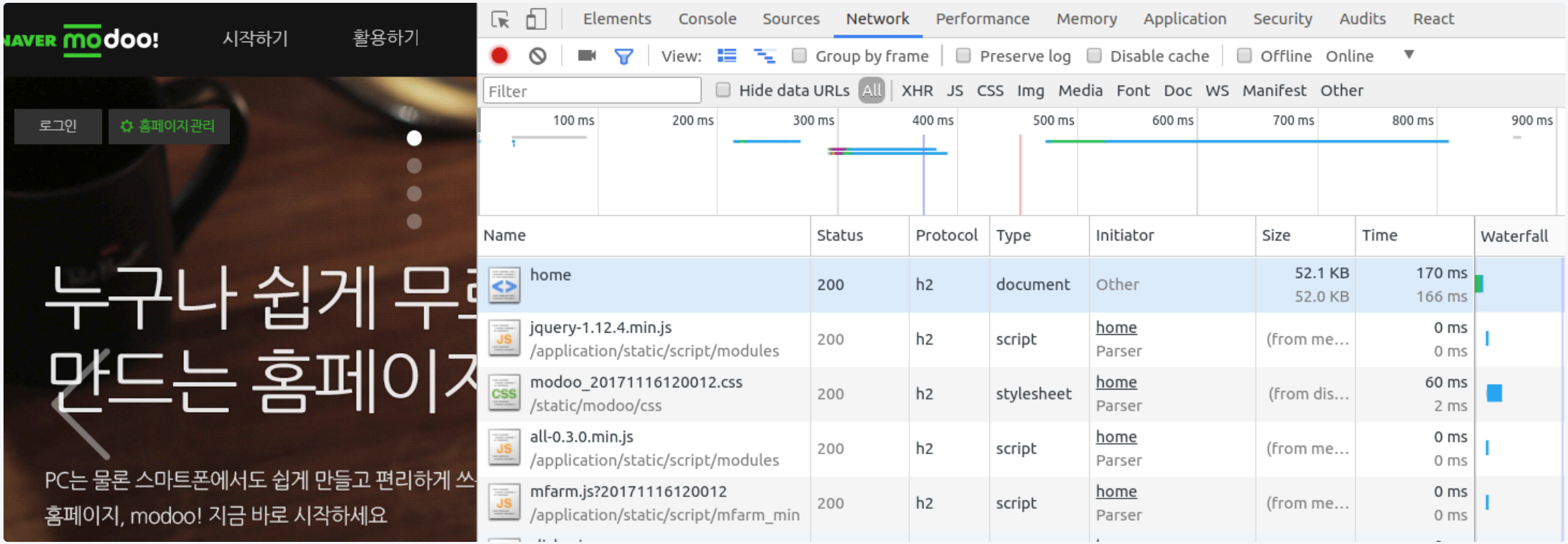
HTTP/2 ?

- HTTP/2 시대는 이미 와있다.




NAVER 카페
전체
카페 홈
카페스토리
대표카페
랭킹
인기팬카페
공식게임
cafe story
2017.11.21.
Free Running Crew
건강 지킴이
197
777
· 대회후기
· 자유게시판
· 개인운동기록
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
React
View:
Group by frame
Preserve log
Disable cache
Offline
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
200 ms
400 ms
600 ms
800 ms
1000 ms
1200 ms
1400 ms
Name
Status
Protocol
Type
Initiator
Size
Time
Waterfall
section.cafe.naver.com
200
h2
document
Other
74.1 KB
64.3 KB
33 ms
30 ms
CommonTopScript-150156879900…
/static/js/section
200
h2
script
(index)
Parser
(from me…
0 ms
0 ms
h_cafestory.gif
cafe.pstatic.net/img/section
200
OK
http/1.1
gif
(index)
Parser
(from me…
0 ms
0 ms


NAVER 블로그
블로그 홈
주제별 보기
이달의 블로그
공식블로그
핫토픽 > 니트 관리법
니트 보풀 제거 방법
보풀제거기 없이
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
React
View:
Group by frame
Preserve log
Disable cache
Offline
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
500 ms
1000 ms
1500 ms
2000 ms
2500 ms
3000 ms
Name
Status
Protocol
Type
Initiator
Size
Time
Waterfall
section.blog.naver.com
302 Found
http/1.1
text/plain
Other
369 B
0 B
267 ms
11 ms
BlogHome.nhn
200
h2
document
section.blog.naver....
Redirect
3.3 KB
3.1 KB
255 ms
18 ms
section_ngapp-4235603_https.css
ssl.pstatic.net/t.static.blog/section...
200
h2
stylesheet
BlogHome.nhn
Parser
24.2 KB
159 KB
60 ms
6 ms
NgAppBundle1-1654951356_https.js
ssl.pstatic.net/t.static.blog/section...
200
h2
script
BlogHome.nhn
Parser
129 KB
367 KB
60 ms
8 ms


NAVER modoo!
시작하기
활용하기
로그인
홈페이지관리
누구나 쉽게 무
만드는 홈페이지
PC는 물론 스마트폰에서도 쉽게 만들고 편리하게 쓰
홈페이지, modoo! 지금 바로 시작하세요
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
React
View:
Group by frame
Preserve log
Disable cache
Offline
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
800 ms
900 ms
Name
Status
Protocol
Type
Initiator
Size
Time
Waterfall
home
200
h2
document
Other
52.1 KB
52.0 KB
170 ms
166 ms
jquery-1.12.4.min.js
/application/static/script/modules
200
h2
script
home
Parser
(from me…
0 ms
0 ms
modoo_20171116120012.css
/static/modoo/css
200
h2
stylesheet
home
Parser
(from dis…
60 ms
2 ms
all-0.3.0.min.js
/application/static/script/modules
200
h2
script
home
Parser
(from me…
0 ms
0 ms
mfarm.js?20171116120012
/application/static/script/mfarm_min
200
h2
script
home
Parser
(from me…
0 ms
0 ms

- 2015년 5월 RFC 7540 스펙 공표

> *Idea was to maintain HTTP semantics but change how it is transported.*
>
> *Daniel Stenberg (curl 개발자)*

특징

- Binary protocol

데이터 전송 단위

- Stream
  - 하나의 H2 연결은 여러개의 Stream을 동시에 열어둘 수 있음.
- Frame
  - Stream 내부의 전송 단위

연결 지속

> *Servers are encouraged to maintain open connections for as long as possible but are permitted to terminate idle connections if necessary.*
>
> *https://http2.github.io/http2-spec/#rfc.section.9.1 참조*

- 기존 Keep-Alive 관련 헤더들은 무시됨.

```
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
```

헤더

- Pseudo-header 필드
  - HTTP/1.1 에서는 메시지 시작줄에 표시하던 정보
  - ':'로 시작
  - `:status`, `:authority`, `:scheme`, `:method`, `:path`
- https://http2.github.io/http2-spec/#HttpHeaders 참조

HTTP 1.1 을 계승했지만 다른 프로토콜

- 1.1로 요청이 오면 응답 코드 101을 내릴 수 있다.
  - https://http2.github.io/http2-spec/#discover-http 참조

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
```

익숙해질 용어

- h2 : HTTP/2 (with TLS)
  - 웹브라우저와 서버간의 통신
- h2c : Clear text HTTP/2 (without TLS)
  - Server to Server 연결
  - 지원하는 웹브라우저는 없음
- ALPN : Application-Layer Protocol Negotiation
  - 프로토콜 협상을 위한 TLS 확장
  - h2 연결에 필요

왜 HTTP/2 로 가는가?

더 나은 성능

- 헤더 압축
- Multiplexing
  - 하나의 TCP 커넥션으로 여러 요청을 동시에 전송
- Server push
  - 연관된 자원을 서버에서 밀어줌
  - 예) HTML에서 참조할 이미지를 HTML 요청 시점에 전송

- 성능 체감 Demo
  - http://www.http2demo.io/
  - https://http2.akamai.com/demo
  - https://www.httpvshttps.com/
  - https://www.tunetheweb.com/performance-test/

- HTTPS로 서비스할 사이트에는 HTTP/2로 안 하면 성능에서 불리
- HTTPS로 서비스 안 하기에는
  - HTTPS가 아닌 사이트에 대한 크롬의 경고 때문에 압박이 있음

기대!?

- .js, 이미지 파일 안 합쳐도 된다?
  - 더 효율적인 파일별 캐쉬
  - 빨라지는 FE 빌드
- Server push로 한 페이지에 필요한 자원은 미리 전송
  - 성능 그래프가 계단형에서 사다리형으로?

Server push 지원 사례

Apache httpd

설정 파일에서 지정 가능

예) 모든 html 요청이 오면 commons.css는 server push 해주기. 쿠키를 이용해서 이전 방문 여부를 검사

```
<filesMatch "\.([hH][tT][mM][lL]?)">
    Header add Link "</assets/css/common.css>;rel=preload;as=style" env=!cssloaded
    Header add Set-Cookie "cssloaded=1; Path=/; Secure; HttpOnly" env=!cssloaded
</filesMatch>
```

nghttpx

- A reverse proxy for HTTP/2, HTTP/1 and SPDY.
- Response 헤더에 Link헤더 필드가 있으면 server push를 해 줌.

```
Link: </assets/css/common.css>; rel=preload
```

- 어플리케이션 서버에 push 관련 로직을 넣을 수 있다면 보다 정교하고 유연하게 적용 가능
  - Java 생태계는 이를 잘 지원할까?

Java 생태계의 HTTP/2 지원

Servlet 4.0 : Server push

- `javax.servlet.http.PushBuilder`
  - Servlet 4.0을 지원하는 Container에서만 사용가능
  - Tomcat 9, Jetty 10, Undertow 2.0
- 이전 버전의 구현체
  - Jetty 9.3에서 비표준으로 먼저 지원: `org.eclipse.jetty.server.PushBuilder`
  - Tomcat 8.5에는 `org.apache.catalina.servlet4preview.http.PushBuilder` 포함

- Spring MVC Controller에서 바로 사용 가능
  - Spring framework 5, Spring Boot 2.x 이상

```
@RestController
public class HomeController{
    @GetMapping("/")
    String home(PushBuilder pushBuilder) {
        pushBuilder
                .path("h2-1.jpg")
                .addHeader("content-type", "image/png")
                .push();
        return "index";
    }
}
```

- Spring MVC 내부구현
  - ServletRequestMethodArgumentResolver 에서 `javax.servlet.http.PushBuilder` 관련 처리 로직 추가
  - Tomcat 8.5, Jetty 9.3의 PushBuilder 관련 클래스는 지원하지 않음.

Java9 : HttpClient

동기처리 방식

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest req = HttpRequest.newBuilder()
    .uri(new URI("https://h2demo.net"))
    .version(HttpClient.Version.HTTP_2)
    .GET().build();
HttpResponse<String> res = client.send(req, HttpResponse.BodyHandler.asString());
System.out.println(res.headers().map());
System.out.println(res.body());
```

비동기 방식

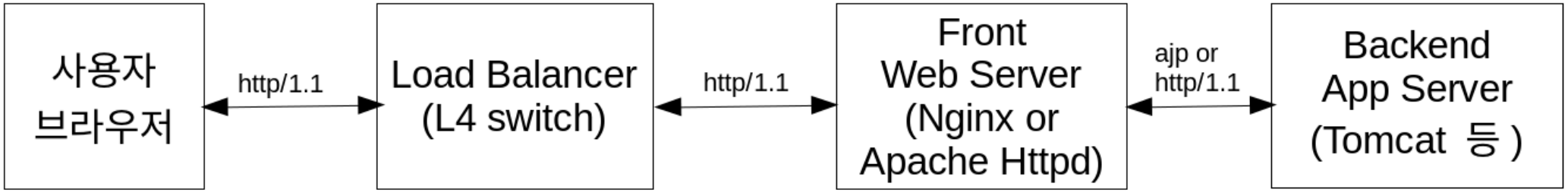
```
httpClient.sendAsync(req, HttpResponse.BodyHandler.asString())
    .thenApply(res -> {
        System.out.println(res.headers().map());
        return res.body();
    }).thenApply(body -> {
        System.out.println(body);
        return null;
    });
TimeUnit.SECONDS.sleep(2);
```

Java9 : ALPN 지원

- Java 서버에서 TLS offloading을 할 때 사용
- JEP-244
- Java7,8에서 ALPN을 쓰려면?
  - jetty-alpn-agent 활용 가능

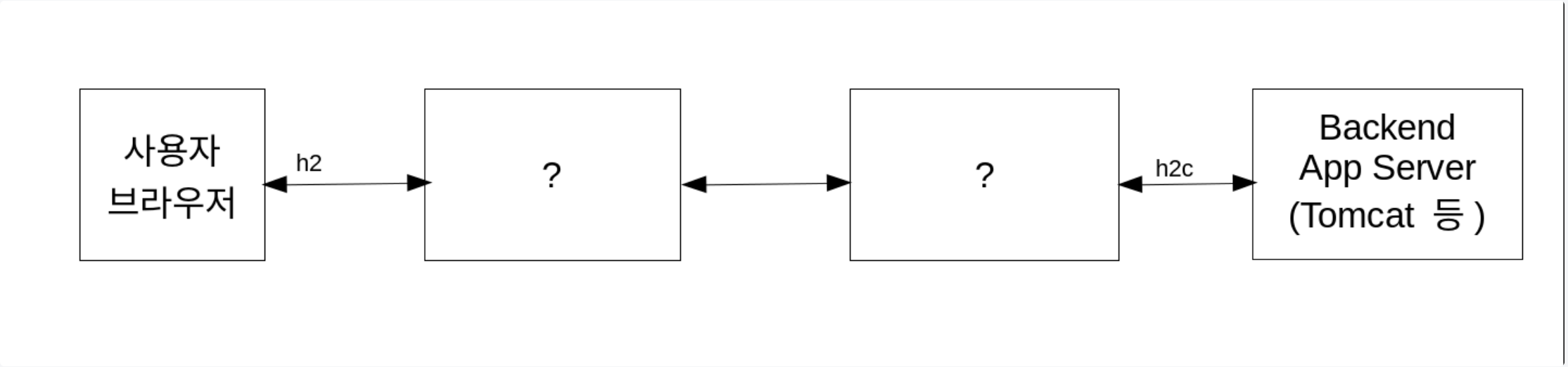
이상적인 서버 구성

레거시의 전형적인 구성



h2 시대의 이상적인 구성

- Java 서버 앞에서 TLS 처리
  - Java단에서는 TLS 처리하니 3.2ms->24ms로 느려졌다는 사례
- Front Web server - Backend App 서버 간에는 h2c
  - 프로토콜 전환 비용 최소화



이상적인 클라이언트

- 레거시 코드에서 1~2줄만 바꿔주면 바로 HTTP/2로 통신

현실의 난관

Demo

- Nginx + Tomcat으로 h2 서버 구성
- 크롬 개발자 탭 (Ctrl + Shift + i)
- HTTP/2 세션 페이지 (chrome://net-internals/#http2)
- 클라이언트 테스트
  - JDK9 HttpClient, OkHttp, Netty
- tshark로 Upstream 연결 확인
  - `tshark -i lo -f 'tcp port 8080' -T fields -e _ws.col.Protocol -e _ws.col.Info`
- Server push 시도

demo 결과

- Nginx-Tomcat 사이에는 HTTP 1.1 연결
  - HttpClient로 8080 포트에게 요청을 날리면 h2c 연결이 되는데도…
- Servlet 4.0의 Server push 안 먹힘

Front 웹서버의 제약

- Apache httpd : 2.4.17 이후 h2 지원
  - 2.4.26 버전까지는 crash 사례가 알려짐
  - 2.4.27 이후로 prefork 모드에서 http2_module 사용 불가

사용하는 Apache 모듈이 멀티스레드에서 안전한지 확신이 없어서 prefork mode를 실무에서는 많이 씀

- prefork mode : 연결 1개당 1개의 프로세스
- worker mode : 연결 1개당 1개의 스레드

- 안정화 되더라도 Apache httpd가 h2에 어울릴지는 의문
  - 연결당 자원소모가 큰 구조.
  - Event MPM이 있지만 Nginx보다 성능에서 뒤지는 것으로 나옴. (Appendix 참조)

- Nginx : 1.9.5 이상에서 HTTP/2 지원
  - 1.9.9 미만에서는 upstream으로의 HTTP/1.1 연결에 버그가 있었음
  - 서버 push 지원 안 함
  - Upstream (Backend server) 연결에 h2c를 지원 안 함
    - 아직 구체적인 계획 없음
    - Nginx 개발자는 Server to server 연결에서는 http/1.1로도 충분하다고 인식하고 있음 (http://mailman.nginx.org/pipermail/nginx/2015-December/049445.html 참조)

어플리케이션 서버의 제약

- Servlet 4.0의 Server push를 지원하는 서버들이 아직 초창기
  - Tomcat 9은 아직 Beta
  - Jetty 10은 아직 Snapshot
- AJP로는 서버 Push가 지원되지 않음.
  - Apache httpd + AJP 구조에서는 Java단에서 서버 push를 시작할 수 없음.

Java9 업그레이드의 어려움

- 3rd party 라이브러리에서 Java9 지원이 불안정한 경우가 있음
  - 예) Lombok
- JDK의 모듈화로 인한 추가 작업

예) JAXB 모듈을 쓰기 위해서는 모듈과 관련된 추가 선언이 필요함.

```
<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.7.0</version>
    <configuration>
        <compilerArgs>
            <arg>--add-modules</arg>
            <arg>java.xml.bind</arg>
        </compilerArgs>
    </configuration>
</plugin>
```

관련된 선언이 없으면 'package javax.xml.bind.annotation is not visible' 에러

Client 라이브러리의 미성숙

- Java9의 jdk.incubator.http.HttpClient
  - 아직 incubator
  - Java10(2018?) 이후에 java.httpclient로 정식 포함.
  - 하위호환성 깰 수도 있음.

정리

- Apache httpd는 지속적 연결에 Nginx보다 상대적으로 불리
- Nginx는 server push, Upstream과의 h2 연결을 하지 못함.
- Java단에서 TLS off loading하면 느림.
- Servlet 4.0의 Server push를 포기하면 익숙한 뒷단 구성을 살릴 수 있음.
  - 브라우저-(h2)-> Nginx -(http/1.1)-> Tomcat 구성
- nghttpx를 앞단에 두고 LINK 헤더를 이용한 push 자원 지정 방식도 가능

- 어차피 Server push 못 쓰니
  - h2를 위해 Tomcat 9, Spring 5 올릴 필요 없다.
- Java9 클라이언트
  - 인큐베이터이니 테스트 정도만?
- 내부망의 Server to Server 연결에서는 굳이 h2(with TLS)를 써야 하는지도 의문을 가져야 함.

- HTTP/2 시대에 대비하는 생계형 Java 개발자의 자세
  - HTTPS 전환, Nginx 사용
  - 이미 Nginx + HTTPS 쓰고 있다면 : Nginx 설정 한 줄만 바꿔도 h2 쓸 수 있다.
  - Servlet 4.0, Java9 Client 스펙은 아직 신경 쓰지 않아도 됨.

감사합니다

늘 그랬듯이 길을 찾기

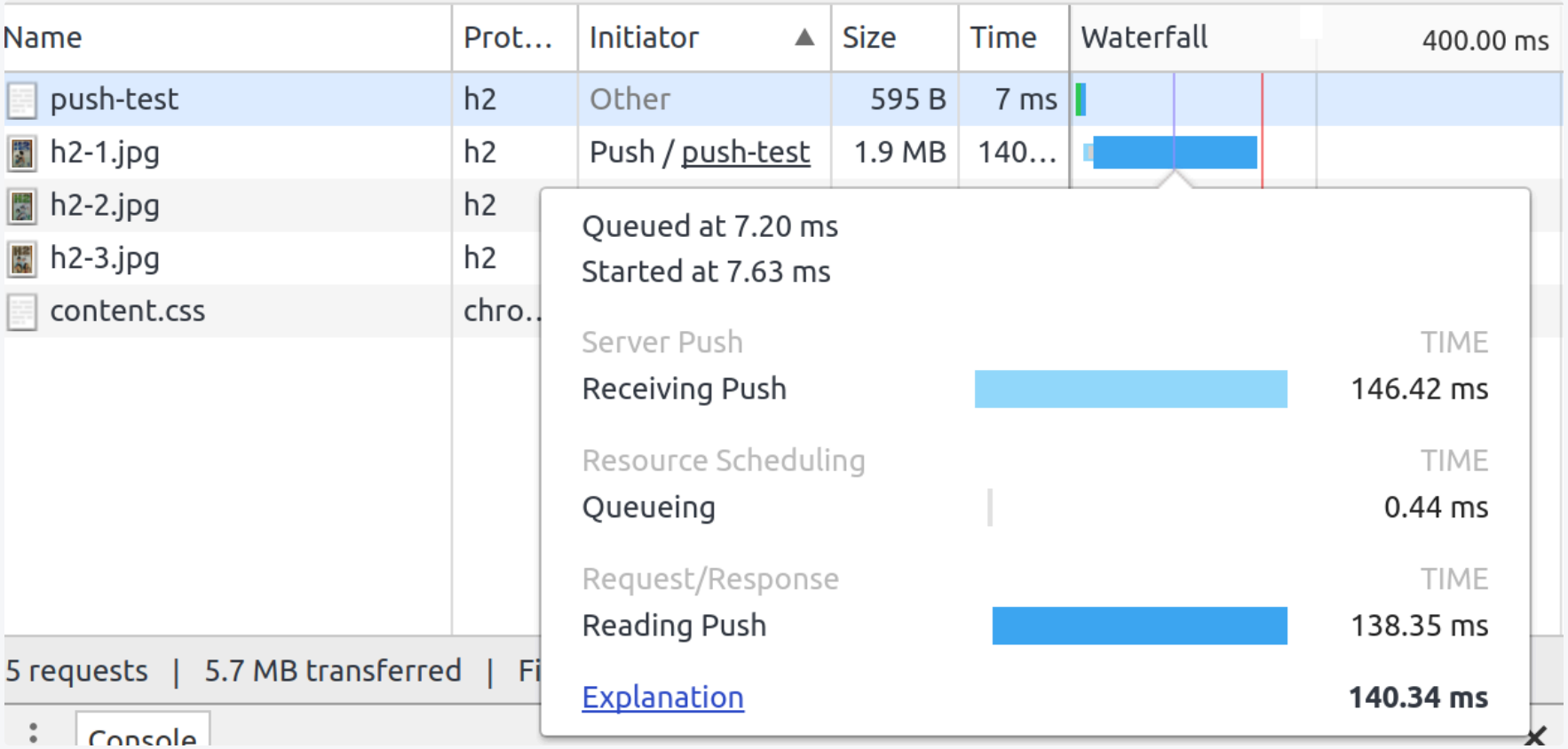
HAProxy

- 오픈소스 Load balancer & Proxy 서버
- http://d2.naver.com/helloworld/284659 참조
  - 2013의 네이버 개발자 블로그(Helloworld)의 글

Demo 2

- HAProxy + Tomcat으로 h2/h2c 서버 구성
- Server push
- 클라이언트 호출 테스트
  - JDK9 HttpClient, OkHttp, Netty

- Demo2 결과

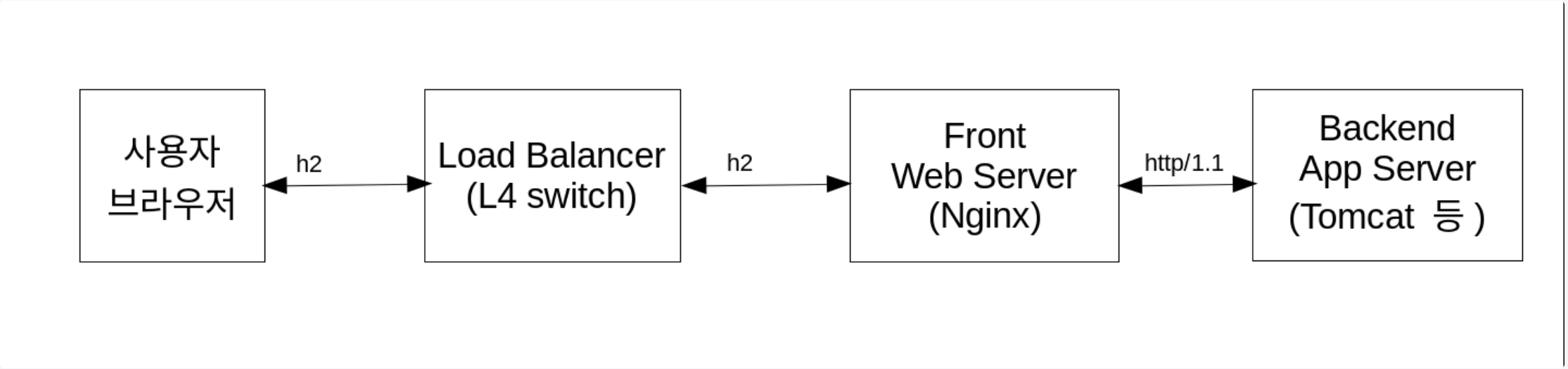

Name
Prot…
Initiator
Size
Time
Waterfall
400.00 ms
push-test
h2
Other
595 B
7 ms
h2-1.jpg
h2
Push / push-test
1.9 MB
140…
h2-2.jpg
h2
h2-3.jpg
h2
content.css
chro…
Queued at 7.20 ms
Started at 7.63 ms
Server Push
TIME
Receiving Push
146.42 ms
Resource Scheduling
TIME
Queueing
0.44 ms
Request/Response
TIME
Reading Push
138.35 ms
Explanation
140.34 ms
5 requests | 5.7 MB transferred | Fi
Console

현실 적용

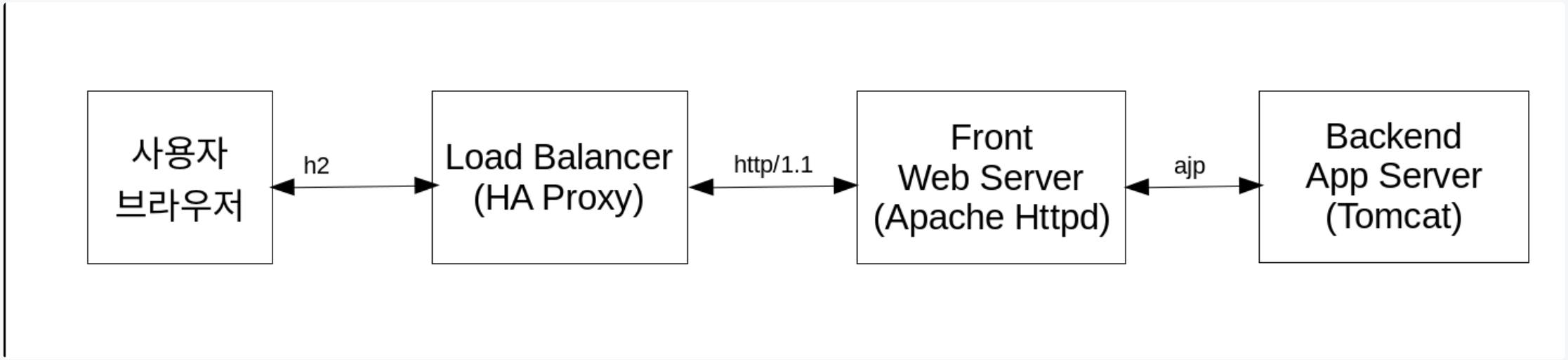
Server push를 포기하고 친숙한 구성

- 앞단에 Nginx
  - h2는 persistent connection이기 때문에 Nginx가 유리
- Upstream 서버는 HTTP 1.1 때와 동일하게 하면 됨
- 첫 번째 Demo와 유사한 구성

첫 번째 demo와 유사한 구성

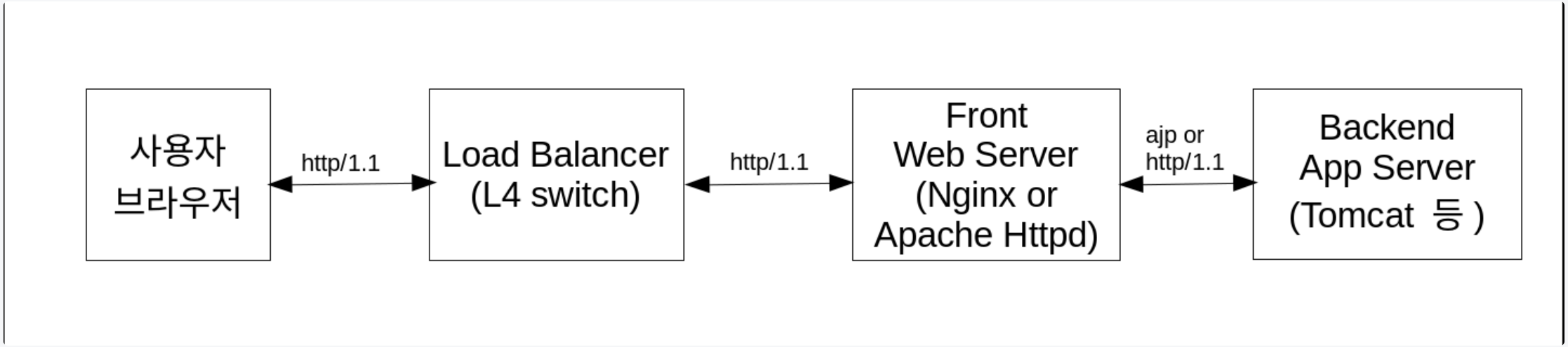


뒷단 서버의 구성을 건드리지 않고 HA Proxy만 추가해서도 가능

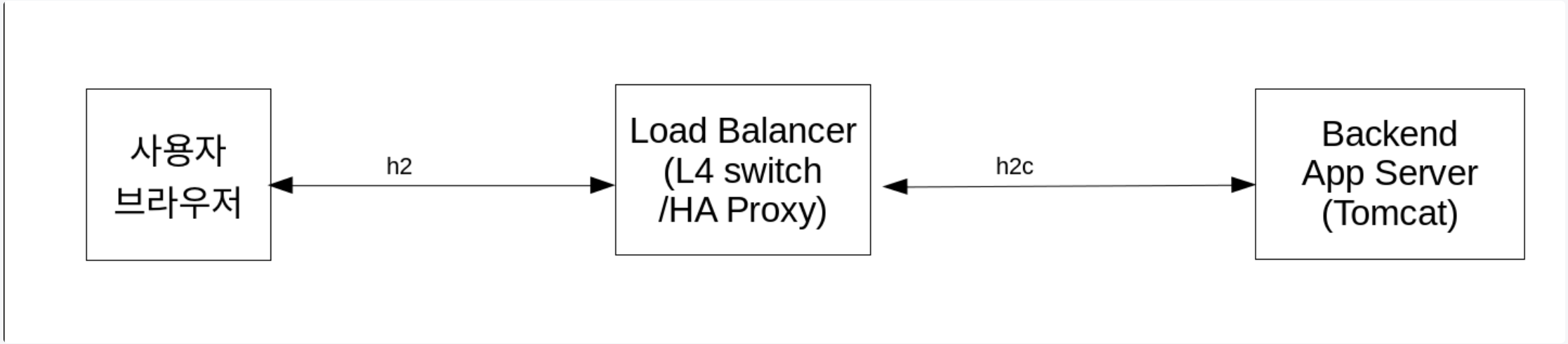


Server push를 지원하는 서버 구성

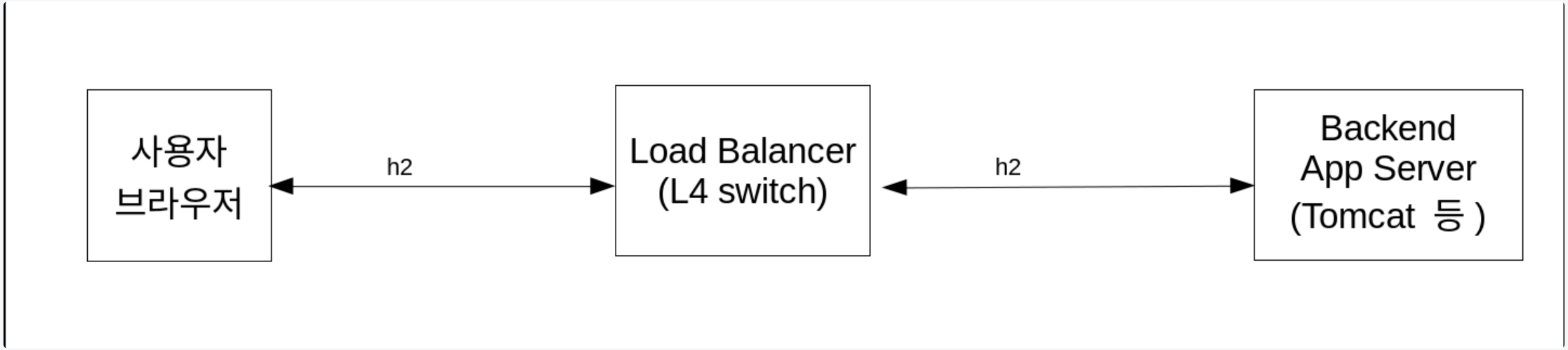
- nghttpx를 이용한 구성
  - 뒷단 연결이 HTTP/1.1 이면 Servlet 4.0의 Server push를 사용 불가
  - LINK 헤더를 이용한 Server push 가능
  - nghttpx와 Tomcat 중간에 Apache httpd 또는 Nginx를 넣을 수도 있음.



- HAProxy -> Tomcat 연결
  - Servlet 4.0 server push 사용 가능
  - Jetty에서도 권장하는 방식
  - Demo2와 같은 구성



- L4 Switch -> Tomcat
  - Servlet 4.0 server push 사용 가능
  - TLS offloading은 Forked Tomcat native 로



- 기존 서버 구성과 달라져서 고민해봐야 할 것
  - Health check 방식
  - 배포 시 서버를 제외하는 방식

클라이언트 모듈 선택

- 내부망에서의 호출은 HTTP 1.1로 충분할 수 있음.
  - OkHttp도 아직 h2c 미지원
- 외부 서버와 HTTPS + HTTP/2 호출을 해야 한다면
  - 현시점에서는 OkHttp 추천
- Java9 HttpClient는 h2c까지도 지원한다는 장점이 있음.
  - 테스트용 스크립트를 만들기에 좋음

RestTemplate + OkHttp3

```
OkHttp3ClientHttpRequestFactory requestFactory = new OkHttp3ClientHttpRequestFactory();

RestTemplate restTemplate = new RestTemplate(requestFactory);
ResponseEntity<String> response = restTemplate.getForEntity("https://www.naver.com", String.class);
System.out.print(response.getHeaders());
System.out.println(response.getBody());
```

향후 과제

Server push의 최적 활용 방식 찾기

- 불필요하게 push하지 않는 규칙이 필요
  - 예) 쿠키를 검사해서 신규 방문자일 때만 push
- Rules of Thumb for HTTP/2 Push 참조

- 더 효율적인 전송을 위해 제안된 스펙
  - Cache Digests for HTTP/2 : Cache 되지 않은 파일만 push할 수 있도록 클라이언트가 서버에게 알려 줌
  - An HTTP Status Code for Indicating Hints : Client에게 힌트를 줄 때 쓸 수 있는 새로운 상태 코드

Server to Server 연결에 h2c vs HTTP/1.1

- h2c의 multiplexing을 이용하면 Connection pool 구성이 어떻게 달라져야 할까?

프레임워크 지원

- Jetty의 PushCacheFilter
  - 연관된 자원의 push를 자동으로 알아서

정리

- HTTP/2 서버 시대를 맞이하는 2개의 길
  - 길1: 기존에 익숙한 서버 구성을 쓰고 Server push를 포기하기
    - 충분히 좋은 선택일 수도 있다.
    - 이 길에서도 Apache httpd + AJP의 전망은 밝지 않다
  - 길2: Server push의 쓰임새를 찾아보기
    - 기존과 서버 구성이 많이 바뀔 수 있다

- 클라이언트 : 더 다양한 프로그래밍 모델, 라이브러리가 난립하는 시대.
  - Spring 내에서도 RestTemplate, AsyncRestTemplate, WebClient
  - HTTP/2 지원 여부까지 포함하면 경우의 수가 더 많아짐.
  - 다양한 시도와 공유가 필요

- 아직까지 기회의 땅
  - 수치화될 수 있는 성과
  - Framework 지원이나 사용 패턴에 있어서 새로운 아이디어가 나올 여지가 큼

바이너리 웹 세대

- 바이너리 실행 코드(WebAssembly) + 바이너리 프로토콜(h2)
- 어플리케이션 실행 플랫폼으로서의 최적화된 웹
- 브라우저 위에서 바이너리 RPC 프로토콜을 실행할 수 있는 여지도 커짐
  - 예) browser-thrift, grpc-web-client
  - 게임, 지도 등 고성능이 필요한 클라이언트 환경에서 고려해볼 만함
  - Web API 형식의 다양화를 이끌지 않을까?

Appendix

Apache Event MPM vs Nginx 비교 사례

Nginx 쪽의 응답시간/처리량이 모두 좋게 나옴.

- (2014년) http://www.eschrade.com/page/performance-of-apache-2-4-with-the-event-mpm-compared-to-nginx/
- (2013년) http://www.mimul.com/pebble/default/2013/04/11/1365675641729.html
- (2013년) https://www.nidc.kr/customer/content.asp?board=tec&uid=994&board_group=0&params=page=9

event mpm의 한계

- worker MPM 을 바탕으로 하고 있기 때문에 context switching이 발생하는 구조
  - event driven을 근본적인 설계에 담은 Nginx와는 다름.
- mod_gzip , mod_ssl 등의 모듈 때문에 blocking 되는 구간이 생김

(https://stackoverflow.com/questions/27856231/why-is-the-apache-event-mpm-performing-poorly)

AJP의 server push 미지원

- AjpProcessor의 상위 클래스인 AbstractProcessor 에서 doPush() 메서드는 UnsupportedOperationException을 던지고 있음.
- AjpProcessor 에서는 이를 특별히 override하지 않고 있음.
- h2c 연결에 쓰이는 StreamProcessor 에서는 doPush() 메서드가 구현되어 있음.