High Performance Networking in Chrome

The Performance of Open Source Application

김지훈
devgrapher@gmail.com

Reference

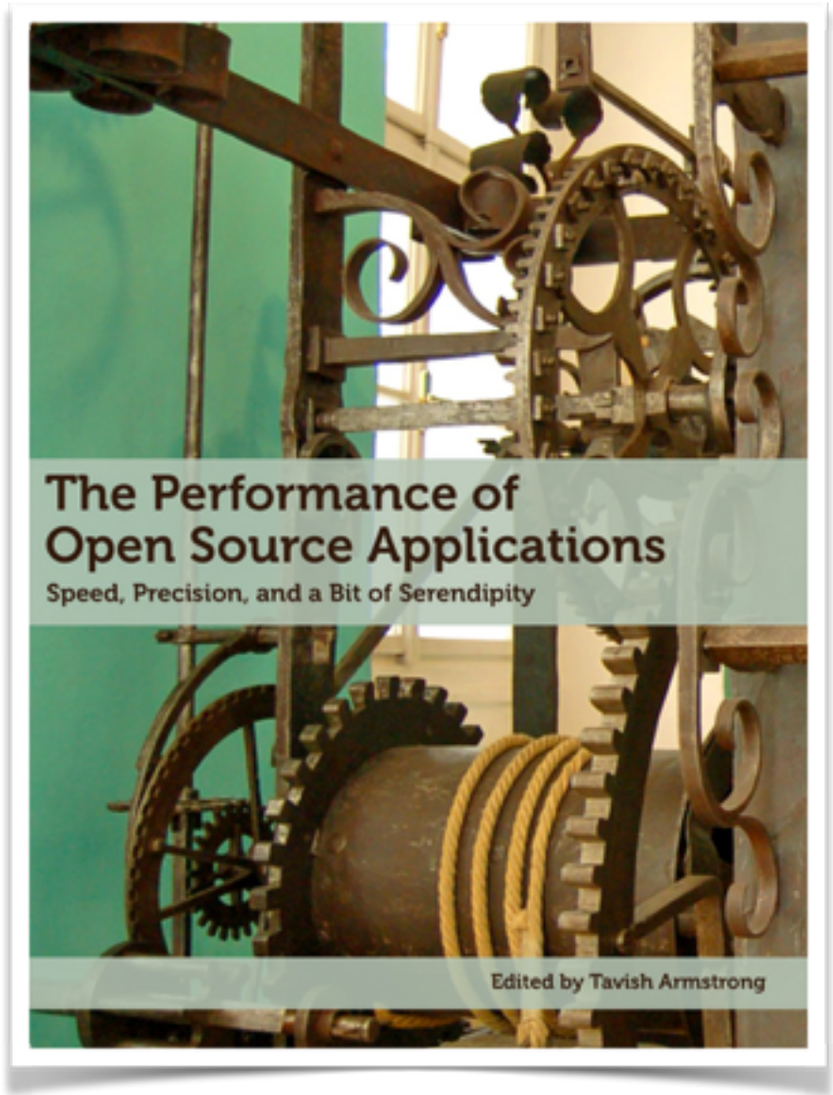
http://aosabook.org/en/posa/high-performance-networking-in-chrome.html

POSA

The Performance of Open Source Applications

자매품

The Architecture of Open Source Applications



Part 1.

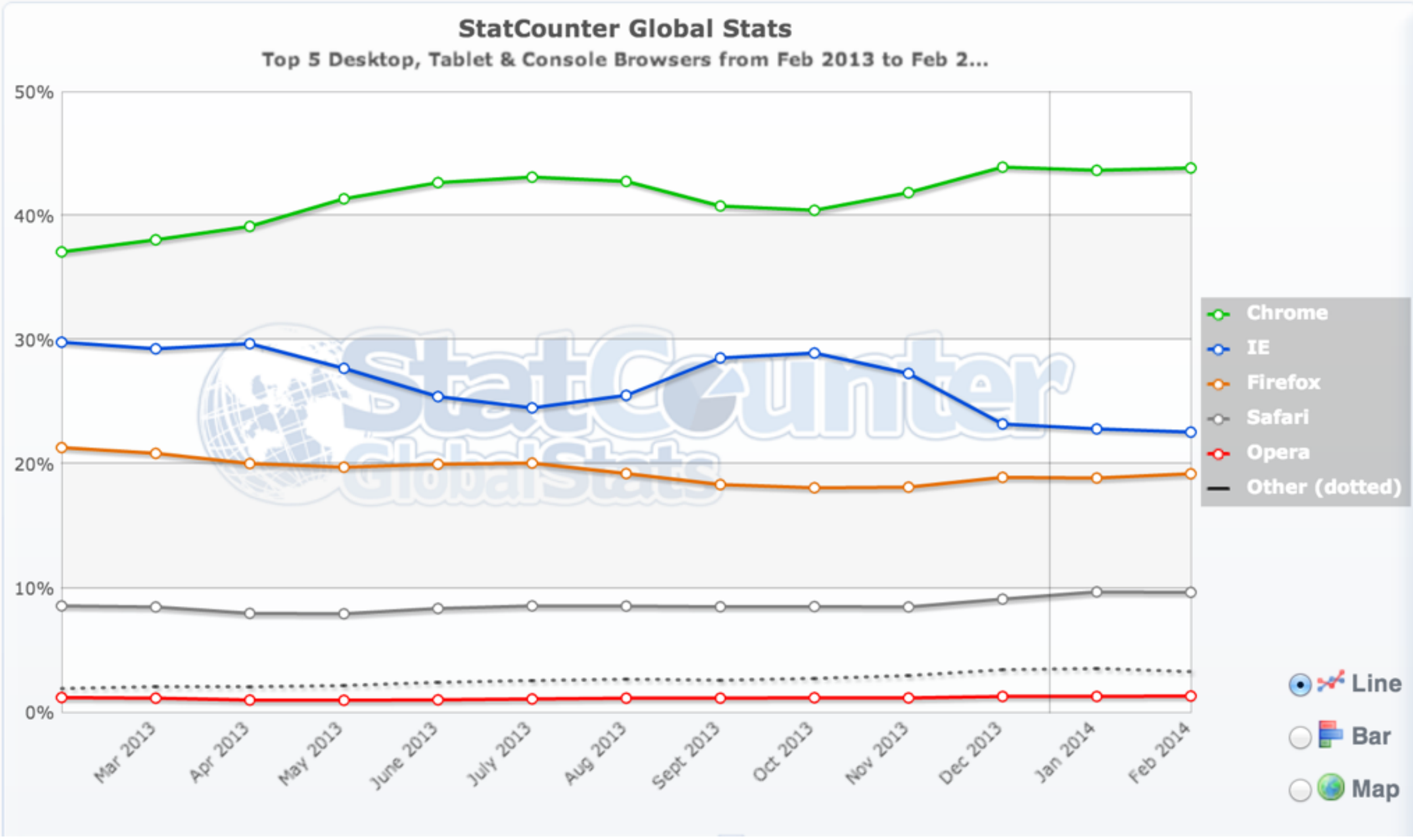
Background Knowledge

Google Chrome

세계에서 가장 많이 쓰는 브라우저

2008년 출시

Chromium 프로젝트, BSD License


StatCounter Global Stats
Top 5 Desktop, Tablet & Console Browsers from Feb 2013 to Feb 2...
50%
40%
30%
20%
10%
0%
Mar 2013
Apr 2013
May 2013
June 2013
July 2013
Aug 2013
Sept 2013
Oct 2013
Nov 2013
Dec 2013
Jan 2014
Feb 2014
Chrome
IE
Firefox
Safari
Opera
Other (dotted)
Line
Bar
Map

Guiding Principles

Speed

가장 빠른 브라우저

Security

안전한 웹 브라우징 환경 제공

Stability

탄력있는 안정적인 웹 플랫폼 제공

Simplicity

복잡한 기술적 요소를 숨기고 단순한 User Experience 제공

Prior to Chrome

단일형 구조(Monolithic), 싱글 프로세스

모든 페이지가 같은 주소 공간을 공유

한 페이지에서 오류가 생기면 프로세스 전체가 죽음

보안에 취약

Multi Process Model

각 페이지별 메모리 공간의 분리

한 페이지의 오류가 미치는 영향을 지역화

보안에 더 안전, Sandbox

멀티코어 시대에 더 나은 성능

크롬의 출현 이후 모든 브라우저가 이를 따름

Modern Web Application

상위 30만개 사이트의 평균 (2013년 1월)

1280 KB 페이지 사이즈

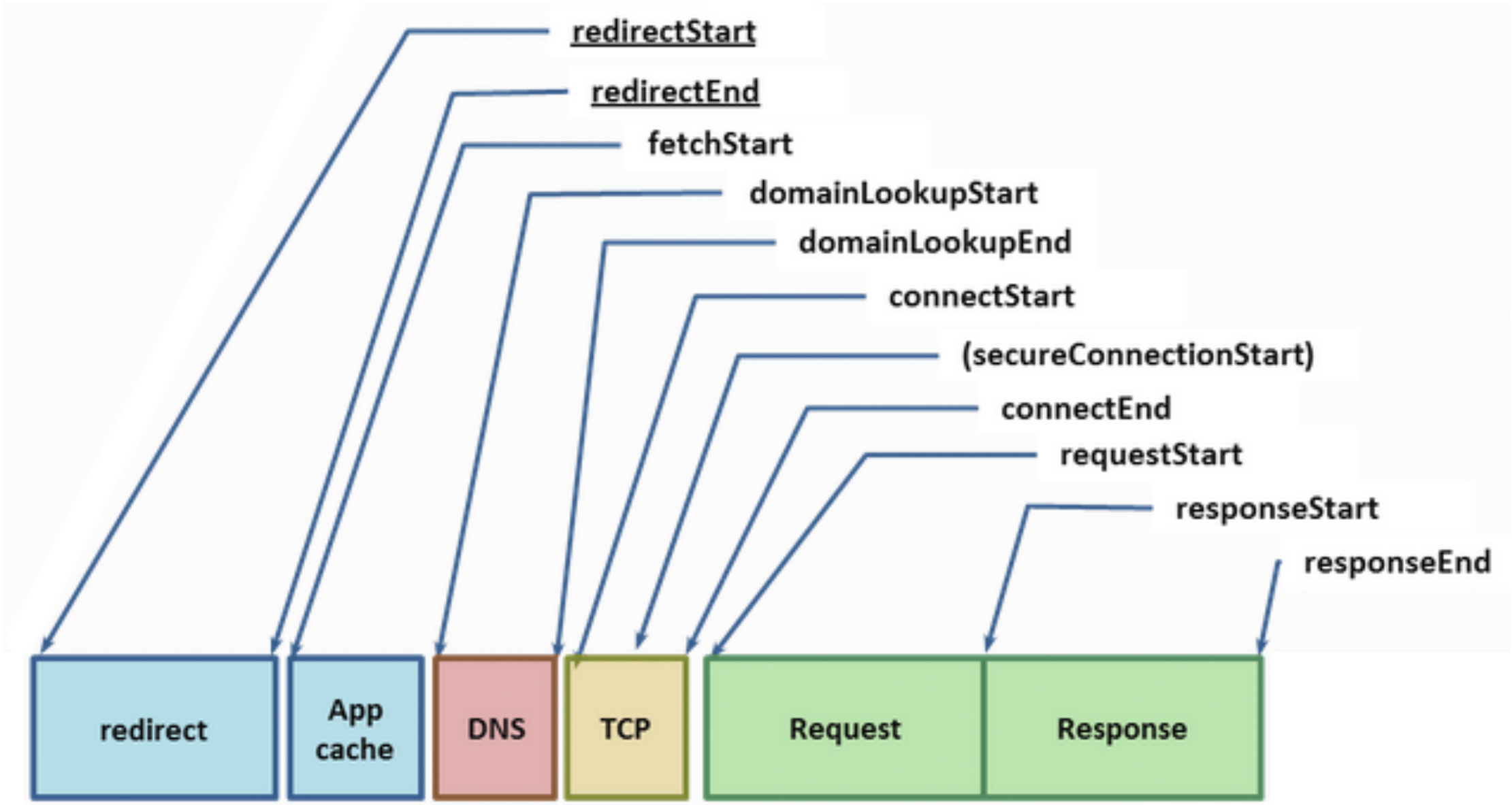
15개 이상의 서로 다른 호스트 링크

88개의 리소스 (이미지, 자바스크립트, CSS…)

1280 / 88 = 15 KB, TCP 의 성능에 부정적

점점 더 늘어 나는 중…

Life of a Resource Request



Delay of a Request

50ms, DNS 조회

80ms, TCP 연결 (한 번 왕복), 미국 기준 1 round-trip : 80ms

160ms, SSL 연결 (두 번 왕복)

40ms, 서버에 요청

100ms, 서버에서 요청 처리

40ms, 서버의 응답 시간

총 470ms

How Fast is Fast Enough

시간	사용자 반응
0 - 100 ms	즉시
100-300 ms	약간의 딜레이를 느낌
300 -1000 ms	컴퓨터가 뭔가 작업중..
1 s+	이미 딴 생각중..
10 s+	다음에 다시 오자

적어도 250ms 안에 시각적인 피드백을 주어야함 (비공식적인 관례)

Part 2.
Chrome's Network Stack

Multi Process Architecture

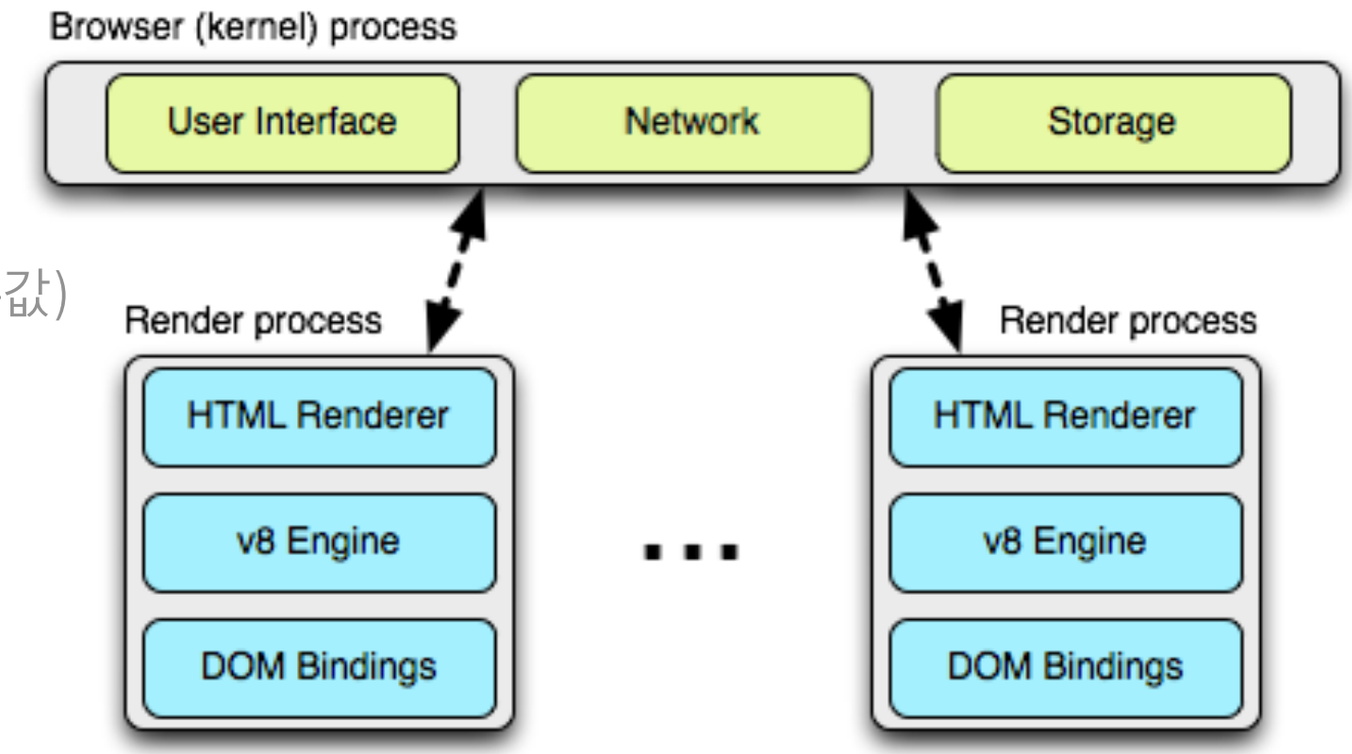
4가지 프로세스 모델

Process per site instance(기본값)

Process per site

Process per tab

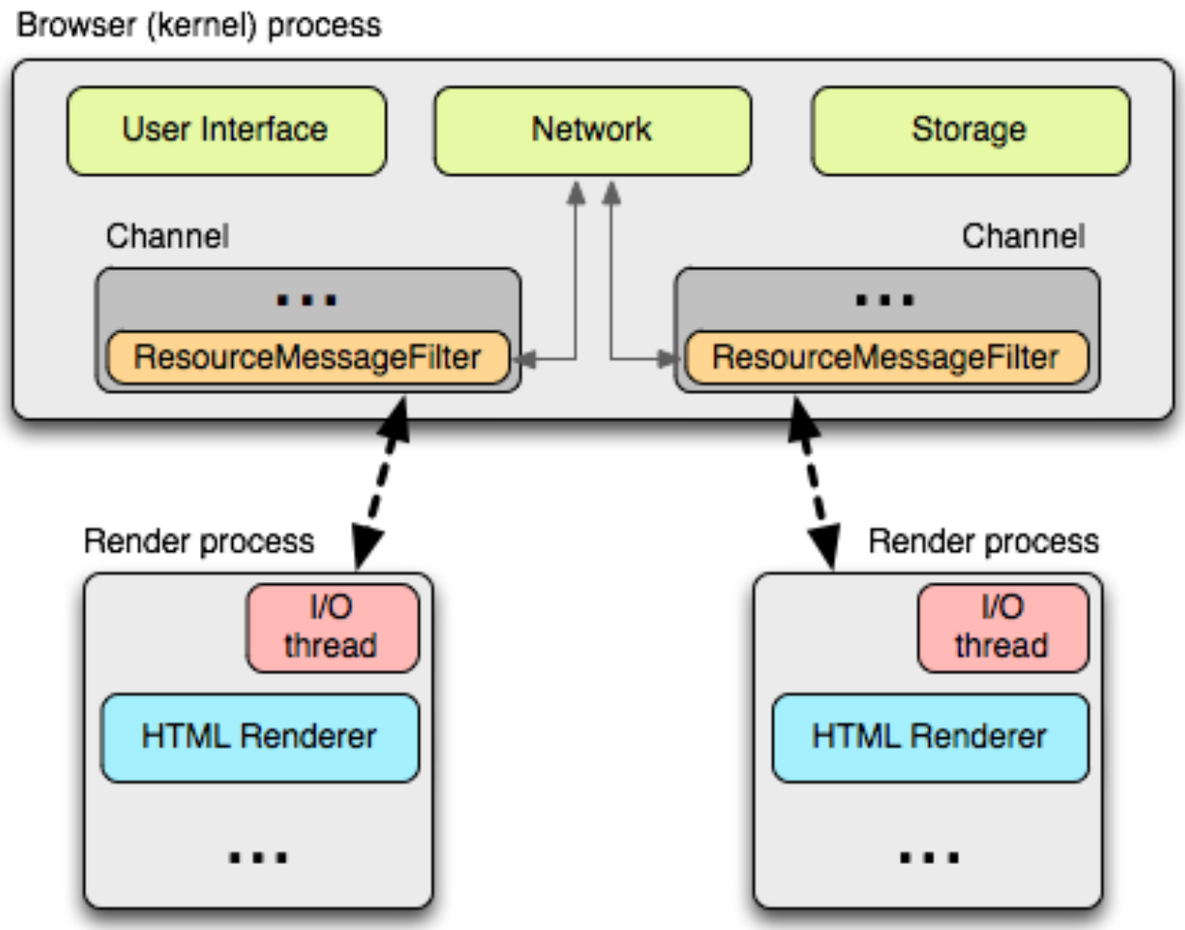
Single process



Renderer는 Sandbox에 의해 자원 접근이 제한됨

브라우저 프로세스와 IPC를 이용해 자원 요청

Resource Loading



IPC: named pipe, 비동기, socketpair()

I/O 스레드에서 IPC처리 담당

ResourceMessageFilter는 I/O스레드에서 도착한 리소스 메세지 필터링 후 ResoureDispatcherHost로 메세지 전달

UI메세지 처리는 I/O처리와 독립적인 스레드에서 작동(I/O때문에 UI에 행이 걸리는 일이 없음)

Resource Sharing

ResourceDispatcherHost는 다음과 같은 최적화 작업을 수행

소켓과 커넥션 수 제한

sockets per profile (256), proxy (32), and {scheme, host, port} (6) groups

소켓 재사용

지연된 소켓 바인딩

새로운 커넥션 작업보다 커넥션이 이뤄진 소켓의 작업 우선 순위를 높임.

세션 공유

쿠키, 캐시데이터등을 브라우저 프로세스가 가지고 모든 렌더러와 공유함.

전역 자원과 네트워크 최적화

브라우저 프로세스는 전역적인 자원 사용을 제어, ex) Foreground 프로세스에 높은 우선순위

예측 최적화(Predictive optimizations)

모든 네트워크 트래픽을 모니터링하여 예측 모델을 생성

Performance on Mobile

Android

Multi Process Model 사용

메모리 사용량에 따라 적절한 렌더러 개수 유지. 심지어 Single Process Model로 전환

iOS

플랫폼 제약으로 인해 항상 Single Process, Multi Thread 모델 사용

똑같은 Network Stack

모든 버전에서 똑같은 최적화 로직 적용

자원 상황에따라 몇몇 파라메터만 바뀜 ex) 캐시 사이즈, 소켓 타임아웃, 예측 최적화의 우선순위 값

모바일 플랫폼의 최적화 내용만 다음 POSA에서 챕터 할당 예정

Chrome’s Predictor

Predicor에 의해 사용할 수록 더 빨라진다.

자주 방문하는 사이트들에 대해

해당 사이트에서 사용하는 리소스 들을 미리 가져온다.

사용자가 링크에 마우스를 올리면

미리 DNS lookup 및 TCP 연결을 수행. 200ms가량의 시간 단축

주소창에 타이핑을 하면

Suggest 한 URL들에 대해 마찬가지로 DNS lookup 및 TCP연결

그래도 시간이 남으면 숨겨진 탭에서 Prerendering까지 수행함.

4 Optimization Techniques

DNS pre-resolve

호스트이름에 대한 IP를 미리 조회하여 dns 레이턴시 제거

TCP pre-connect

TCP 커넥션을 미리 수행하여 TCP handshake 레이턴시 제거

Resource prefetching

페이지 내의 부하가 큰 자원들을 미리 가져옴

Page prerendering

페이지 전체를 미리 렌더링 해두어 사용자가 요청시 즉시 페이지를 보여줌

Risk of Predictor

이러한 프리패칭 작업은 다소 도박성이 있음

예측이 틀린경우 필요없는 자원을 낭비

오히려 성능이 더 떨어질 위험

따라서 최대한 많은 정보를 수집

사용자 방문 히스토리, 렌더러와 네트워크에서 수집된 정보

ResolutionMotivation

```
// url_info.h

enum ResolutionMotivation {
    MOUSE_OVER_MOTIVATED,     // 사용자의 마우스 오버 이벤트
    OMNIBOX_MOTIVATED,        // 주소표시창에서의 이벤트
    STARTUP_LIST_MOTIVATED,   // 이 리소스는 스타트업 목록의 최상위 10개에 해당하는 리소스임
    EARLY_LOAD_MOTIVATED,     // 특정상황에서 prefetcher는 실제 요청이 일어나기 전에
                              // 미리 연결을 수행한다.

    // 다음은 네비게이션 상황에서 일어나는 프리패칭 이벤트임
    STATIC_REFERAL_MOTIVATED,  // External database suggested this resolution.
    LEARNED_REFERAL_MOTIVATED, // Prior navigation taught us this resolution.
    SELF_REFERAL_MOTIVATED,    // Guess about need for a second connection.
  // <snip> ...
};
```

Role of Predictor

Predictor의 목표는 이러한 신호들의 성공 가능성들을 평가하는 것

모든 신호들은 성공률과 우선순위, 유효기간 값을 가지고 있음

이러한 신호들에 대해 프리패칭을 수행하고 난 후에는 이것들의 성공률을 기록함

Part 3.

Lifetime of Your Browser Session

Cold-boot Experience

최초 설치 이후에는 사용자에 관한 정보가 거의 없음

일반적으로 사람들이 많이 방문하는 호스트이름에 대해 미리 dns lookup 수행

사용자가 브라우저 시작후 방문하는 패턴을 미리 분석해놓음

chrome://dns

가장 많이 접근하는 최상위 10개 호스트를 기록해둠

브라우저가 시작하면 이 목록의 호스트 이름들을 미리 패칭해둠

chrome://dns

현재 제 노트북의 chrome://dns

Future startups will prefetch DNS records for 10 hostnames

Host name	How long ago (HH:MM:SS)	Motivation
http://www.google-analytics.com/	05:54:50	n/a
https://accounts.google.com/	05:54:52	n/a
https://apis.google.com/	05:54:44	n/a
https://clients2.google.com/	05:54:51	n/a
https://lh4.googleusercontent.com/	05:54:44	n/a
https://mail.google.com/	05:54:50	n/a
https://ssl.google-analytics.com/	05:54:50	n/a
https://ssl.gstatic.com/	05:54:44	n/a
https://www.google.co.kr/	05:54:45	n/a
https://www.google.com/	05:54:47	n/a

새로운 Profile 생성 직후의 chrome://dns

Chrome | About DNS

chrome://dns

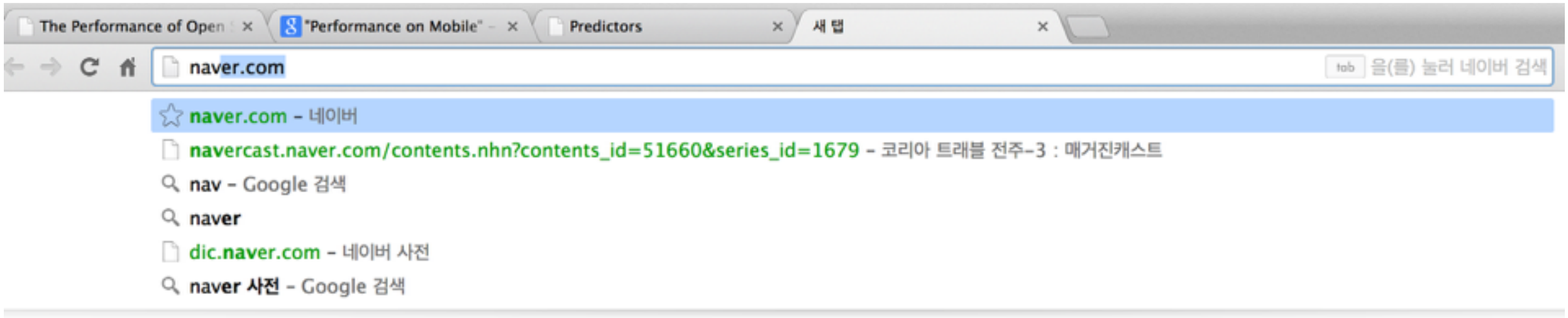
Future startups will prefetch DNS records for 9 hostnames

Host name	How long ago (HH:MM:SS)	Motivation
https://accounts.google.com/	24	n/a
https://accounts.youtube.com/	23	n/a
https://apis.google.com/	19	n/a
https://clients2.google.com/	24	n/a
https://clients2.googleusercontent.com/	22	n/a
https://ssl.gstatic.com/	23	n/a
https://www.google.co.kr/	20	n/a
https://www.google.com/	21	n/a
https://www.gstatic.com/	19	n/a

Host for Page	Page Load Count	Subresource Navigations	Subresource PreConnects	Subresource PreResolves	Expected Connects	Subresource Spec
https://accounts.google.com/	1	1	0	0	2.340	https://accounts.youtube.com/
		4	0	0	3.360	https://ssl.gstatic.com/
https://www.google.co.kr/	2	2	1	0	1.884	https://apis.google.com/
		2	1	0	1.884	https://ssl.gstatic.com/
		13	1	0	4.815	https://www.google.co.kr/
		2	1	0	1.884	https://www.gstatic.com/

Preresolution DNS records performed for 2 hostnames

Interactions with the Omnibox



주소창에 타이핑을 수행하면 방문한 url들과 검색어들을 자동 노출

이렇게 노출되는 목록들은 과거 사용자 방문기록에 의존

각 값들에 대해 성공률과 방문 빈도 수를 유지하고 있음

chrome://predictors

☑ Filter zero confidences

Entries: 125

User Text	URL	Hit Count	Miss Count	Confidence
g	http://gmail.com/	594	186	0.7615384615384615
gi	http://githubarchive.org/	25	55	0.3125
gi	https://gist.github.com/	16	49	0.24615384615384617
gis	https://gist.github.com/	19	1	0.95
gist	https://gist.github.com/	19	1	0.95
githuba	http://githubarchive.org/	3	0	1
gm	http://gmail.com/	411	1	0.9975728155339806

ResourceDisfetcher에 중요한 정보

노란색 : DNS prefetch 수행

녹색 : 높은 성공률, TCP pre-connect 까지 수행

Cache Performance

HTML Response Header

Expires, ETag, Last-Modified, Cache-Control

디스크 파일과 In memory 형태로 관리

LRU(Least Recently Used)

chrome://cache

DNS Prefetching

DNS prefetching이 일어나는 상황

Blink의 Document Parser가 페이지에 존재하는 호스트 이름들을 제공

링크에 마우스를 오버하거나, 클릭하면 렌더러가 알림

주소표시창에 타이핑시에 높은 확률을 가진 대상을 알림

과거 방문 기록과 Resource Request에 기반해 Predictor가 알림

페이지가 명시적으로 Prefetching할 호스트 이름을 제공

리다이렉트 해야 할때 유용.

```
<link rel="dns-prefetch" href="//host_name_to_prefetch.com">
```

getaddrinfo()

두 가지 형태의 DNS lookup 구현이 존재

getaddrinfo() API와 chrome 자체 구현

getaddrinfo()

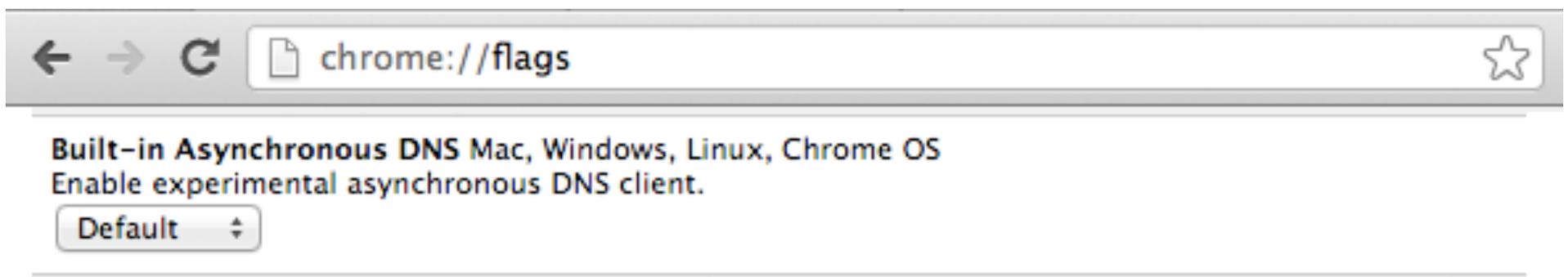
코드가 단순해짐

OS의 DNS캐시를 이용할 수 있음

Blocking 호출이라 모든 DNS lookup마다 개별 스레드를 할당해야 함

직접 관리하지 않으므로 정보가 부족함, ex) Time-to-live(TTL), DNS 캐시 상태

New DNS Resolver



크롬 내장 DNS Resolver

병렬처리 기능 강화

TTL정보에 접근 가능해짐으로서, 자주 방문하는 사이트는 값이 만료되기 전에 미리 리프레쉬

더 나은 Dual Stack 지원, (IPv4, IPv6)

RTT(Round Trip Time)나 기타 다른 정보를 이용해 Failover 수행

chrome://histograms/DNS

```
Histogram: DNS.PrefetchResolution recorded 2621 samples, average = 126.6 (flags = 0x1)
0   O                                                                          (0 = 0.0%)
15 ----------------------------------------------------------------------O (366 = 14.0%) {0.0%}
17 ----------------------------------------------------------O             (292 = 11.1%) {14.0%}
19 ----------------------------------------------------------O             (289 = 11.0%) {25.1%}
21 ---------------------------------------O                                (195 = 7.4%) {36.1%}
23 -------------------------O                                             (187 = 7.1%) {43.6%}
26 ---------------O                                                        (112 = 4.3%) {50.7%}
29 -----------O                                                           (85 = 3.2%) {55.0%}
32 --------O                                                              (79 = 3.0%) {58.2%}
36 ------O                                                                (59 = 2.3%) {61.2%}
40 ----O                                                                  (51 = 1.9%) {63.5%}
45 ----O                                                                  (47 = 1.8%) {65.4%}
50 ---O                                                                   (42 = 1.6%) {67.2%}
```

대략 50%의 프리패치가 20ms이내에 수행 되었음

TCP Pre-connect

chrome://dns

Host for Page	Page Load Count	Subresource Navigations	Subresource PreConnects	Subresource PreResolves	Expected Connects	Subresource Spec
https://plusone.google.com/	51	36	23	18	1.215	https://plusone.google.com/

먼저 소켓Pool에서 해당 Hostname으로 가용한 소켓이 있는지 확인

만약 있다면 이미 연결되어 있으므로 추가 커넥션 작업 필요 없음

없다면 Pre-connect 수행

chrome://net-internals#sockets

Sockets ▼ capturing events (445)

- Close idle sockets
- Flush socket pools May break pages with active connections
- View live sockets

Name	Handed Out	Idle	Connecting	Max	Max Per Group	Generation
transport_socket_pool	3	0	0	256	6	0
ssl_socket_pool	3	0	0	256	6	0

transport_socket_pool							
Name	**Pending**	**Top Priority**	**Active**	**Idle**	**Connect Jobs**	**Backup Timer**	**Stalled**
ssl/apis.google.com:443	0	-	1	0	0	stopped	false
ssl/mail.google.com:443	0	-	1	0	0	stopped	false
ssl/oauth.googleusercontent.com:443	0	-	1	0	0	stopped	false
ssl/ssl.gstatic.com:443	0	-	0	0	0	stopped	false

ssl_socket_pool							
Name	**Pending**	**Top Priority**	**Active**	**Idle**	**Connect Jobs**	**Backup Timer**	**Stalled**
ssl/apis.google.com:443	0	-	1	0	0	stopped	false
ssl/mail.google.com:443	0	-	1	0	0	stopped	false
ssl/oauth.googleusercontent.com:443	0	-	1	0	0	stopped	false

Prefetching Resource

Predictor에 의해 해당 URL에 관련된 리소스를 미리 로드

캐시에 이미 있다면 캐시를 이용

Pre-release channel을 통해 실제 사용자와 네트워크 환경에서 다양한 실험적인 로직을 테스트 중

Prerendering

숨겨진 탭에 페이지를 모두 렌더링 해둠

사용되는 케이스

주소표시줄에서 타이핑시 높은 확률의 Suggest를 만났을 때

때때로 구글은 검색 결과에 Prerender 지시자를 사용함(첫번째 검색 결과)

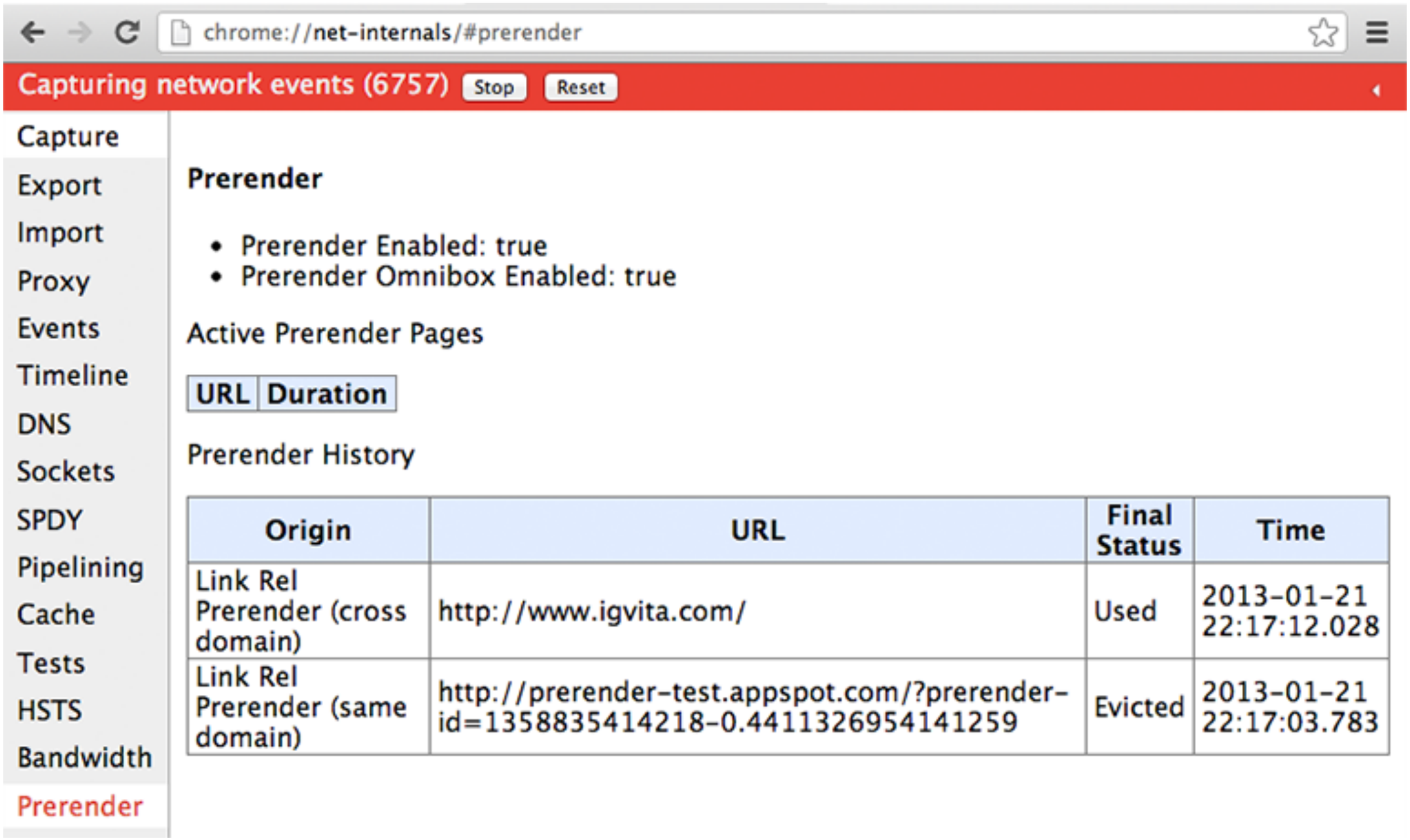
prerender 지시자

```
<link rel="prerender" href="http://example.org/index.html">
```

prerender-test.appspot.com

Prerendering 테스트 페이지

chrome://net-internals/#prerender



Restrictions of Prerendering

모든 프로세스 전역적으로 한 개의 Prerendering만을 허용

HTTPS나 인증과정을 포함하는 HTTP페이지는 불가능

GET 메소드로 호출하는 페이지만 가능

(POST는 나중에 로드할때 파라메터에 따라 페이지가 달라지므로)

모든 리소스 패칭 요청은 가장 낮은 네트워크 우선순위를 가짐

가장 낮은 CPU우선순위로 렌더링됨

메모리 요구량이 100MB 초과한다면 렌더링 포기

플러그인 초기화는 지연되고, 만약 HTML5 Media Element가 있다면 렌더링 포기

Chrome Gets Faster as You Use It

Appendix : Thread Type

```
class CONTENT_EXPORT BrowserThread {
 public:
  // An enumeration of the well-known threads.
  // NOTE: threads must be listed in the order of their life-time, with each
  // thread outliving every other thread below it.
  enum ID {
    // The main thread in the browser.
    UI,

    // This is the thread that interacts with the database.
    DB,

    // This is the thread that interacts with the file system.
    FILE,

    // Used for file system operations that block user interactions.
    // Responsiveness of this thread affect users.
    FILE_USER_BLOCKING,

    // Used to launch and terminate Chrome processes.
    PROCESS_LAUNCHER,

    // This is the thread to handle slow HTTP cache operations.
    CACHE,

    // This is the thread that processes IPC and network messages.
    IO,

    ID_COUNT
};
```

Appendix : Muti-platform Implementation

단일 소스

```
#if !defined(OS_ANDROID)
...
#endif
```

플랫폼별 별도 구현 파일

chrome_browser_main.h

chrome_browser_main.cc // 공통 구현

chrome_browser_main_linux.h

chrome_browser_main_linux.cc

chrome_browser_main_win.h

chrome_browser_main_win.cc

chrome_browser_main_mac.h

chrome_browser_main_mac.cc