Universidad Nacional de Luján

Proxy Adaptativo Para Protocolos Web Avanzados

Trabajo Final presentado por Pablo Maximiliano Lulic
para obtener el grado de Licenciatura en Sistemas de Información

Director: Mg. Gabriel Tolosa
Co-Director: Lic. Marcelo Fernández

2014

Agradecimientos

...

Índice general

Agradecimientos **2**

1. Introducción **6**

2. Desarrollo de la Web Actual **8**

2.1. Introducción 8
2.2. El crecimiento de Internet 8
2.3. El crecimiento de los Sitios 10

3. Conceptos sobre Protocolos **15**

3.1. Definición 15
3.2. Modelo OSI 16
3.2.1. Capa de Aplicación 17
3.2.2. Capa de Presentación 17
3.2.3. Capa de Sesión 18
3.2.4. Capa de Transporte 18
3.2.5. Capa de Red 18
3.2.6. Capa de Enlace 18
3.2.7. Capa Física 18
3.2.8. Funciones de los Protocolos 19
3.3. TCP/IP 20
3.3.1. Las Capas 20
3.3.2. TCP/IP y el Modelo OSI 23
3.4. HTTP 23
3.4.1. HEADERS 26

3.5. HTTPS . . . 26
3.5.1. Criptografía . . . 27
3.5.2. Conexión . . . 29
3.6. SPDY . . . 29
3.6.1. Introducción . . . 29
3.6.2. El Protocolo . . . 31
3.6.3. Alternativas Propuestas . . . 33
3.6.4. Estudios Relacionados . . . 34

4. Problemáticas asociadas a la Web **36**
4.1. Introducción . . . 36
4.2. Performance . . . 36
4.3. Técnicas de optimización . . . 39

5. Proxy **45**
5.1. Definición . . . 45
5.2. Usos de un Proxy . . . 47
5.3. Ventajas y Desventajas . . . 48
5.3.1. Ventajas . . . 48
5.3.2. Desventajas . . . 49
5.4. Configuración . . . 50

6. Caché Web **52**
6.1. Definición . . . 52
6.2. Tipos . . . 53
6.3. Politicas de Reemplazo / Algoritmos de Caché . . . 54
6.4. Control de Caché . . . 55

7. Proxy Adaptativo para Protocolos Web Avanzados **57**
7.1. Introducción . . . 57
7.2. Funcionamiento . . . 58
7.2.1. Análisis de los Recursos . . . 59
7.2.2. Árbol de Decisión . . . 60
7.3. Módulos Adicionales . . . 60

7.3.1. Cálculo del RTT 60
7.3.2. Protocolos Habilitados 60
7.3.3. Cliente SPDY 62

8. Experimentos y Resultados **63**
8.1. Introducción 63
8.2. Extracción Top Alexa 63
8.3. Experimento 63
8.4. Resultados 63

9. Conclusiones **64**

Bibliografía **68**

Capítulo 1

Introducción

El protocolo HTTP tuvo su primera versión en mayo de 1996, culminando en 1999 con el estándar actual que es HTTP 1.1. Es un protocolo sin estado, lo que conlleva a que sea necesario realizar una conexión nueva por cada recurso que se necesite en un sitio.

Los sitios web de la actualidad difieren de las páginas de hace más de 10 años, tanto en tamaño como en cantidad de recursos. Este fenómeno, hace que el protocolo ya no tenga el mismo rendimiento que en épocas anteriores. A pesar del avance de la tecnología en cuanto a mejoras en las velocidades de los enlaces de red, poseer gran ancho de banda no es el único factor que interviene en la performance de la carga de las páginas. Google propuso un protocolo llamado SPDY, como solución a los problemas actuales, busca mejorar la performance de HTTP siendo transparente su implementación. Veremos en los capítulos siguientes, los problemas que presentan los protocolos, y bajo qué circunstancias se ven beneficiados o perjudicados en cuanto a su performance.

Se busca como objetivo de este trabajo, desarrollar un proxy con un algoritmo inteligente que pueda determinar según el perfil del sitio que método (HTTP, HTTPS ó SPDY) elegir para mejorar la performance, es decir, que decisión sería la óptima.

El trabajo se encuentra organizado de la siguiente manera, veremos el desarrollo que tuvo la Web a lo largo de estos años, avanzaremos sobre diversos conceptos de los Protocolos que se utilizan actualmente, así también como sus debilidades. Estudiaremos las problemáticas asociadas a la Web y ciertas técnicas de optimización.

Conoceremos el concepto de Proxy, necesario para comprender el desarrollo en sí, también se introducirán ciertas cuestiones de Caché Web. Ya con toda la información necesaria, nos adentraremos en el desarrollo del Proxy propuesto, con sus características y funcionalidades. Hacia el final, con el fin de probar el Proxy en un ambiente real, veremos un experimento, sus resultados y las conclusiones finales. El glosario, la bibliografía y los anexos se encuentran al final de este trabajo. No existe un capítulo dedicado a Trabajos Relacionados como tal, pero estos pueden encontrarse a lo largo de los capítulos, especialmente en el capítulo de Protocolos (Capítulo 3).

Capítulo 2

Desarrollo de la Web Actual

2.1. Introducción

Internet es la interconexión global de redes individuales alrededor del mundo. Originalmente fué utilizada para interconectar laboratorios dedicados a la investigación gubernamental. Desde 1994 se expandió para millones de usuarios de todo el mundo que la utilizan con múltiples propósitos. A medida que fué creciendo, cambió la forma de hacer negocios y de comunicarse hasta llegar a ser una fuente de información Universal para millones de personas [9].

2.2. El crecimiento de Internet

Internet continúa creciendo día a día. Hacia 1995 la cantidad de usuarios promedio era de 16 millones, 1 año después, la cifra ascendió a más del doble. Para el año 2000 se incrementó 10 veces la cantidad de usuarios. Esta información puede verse en el Cuadro 2.1.

Fecha	Usuarios(mill)	% Población Mundial
Diciembre, 1995	16	0.4
Diciembre, 1996	36	0.9
Diciembre, 1997	70	1.7
Diciembre, 1998	147	3.6

Sigue en la página siguiente.

Fecha	Usuarios(mill)	% Población Mundial
Diciembre, 1999	248	4.1
Diciembre, 2000	361	5.8
Agosto, 2001	513	8.6
Septiembre, 2002	587	9.4
Diciembre, 2003	719	11.1
Diciembre, 2004	817	12.7
Diciembre, 2005	1018	15.7
Diciembre, 2006	1093	16.7
Diciembre, 2007	1319	20.0
Diciembre, 2008	1574	23.5
Marzo, 2009	1596	23.8
Junio, 2009	1669	24.7
Septiembre, 2009	1734	25.6
Diciembre, 2009	1802	26.6
Junio, 2010	1966	28.7
Septiembre, 2010	1971	28.8
Marzo, 2011	2095	30.2
Junio, 2011	2110	30.4
Septiembre, 2011	2180	31.5
Diciembre, 2011	2267	32.7
Marzo, 2012	2336	33.3
Junio, 2012	2405	34.3
Septiembre, 2012	2439	34.8
Diciembre, 2012	2497	35.7
Marzo, 2013	2749	38.8

Cuadro 2.1: Crecimiento de la cantidad de usuarios de Internet, información extraída de [9]

También puede observarse en el Gráfico 2.1 el increíble crecimiento que tuvo la cantidad de usuarios de Internet, y es clara la tendencia de seguir creciendo.

Además de la cantidad de usuarios, en paralelo iba creciendo la cantidad de

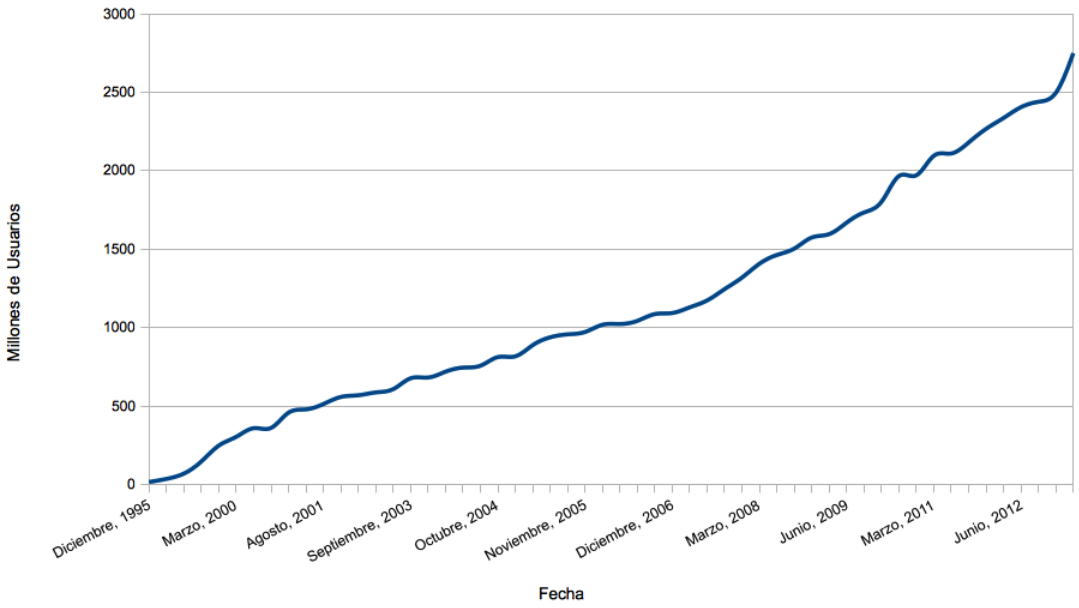

Figura 2.1: Crecimiento de la cantidad de usuarios de Internet, información extraída de [9]

Dominios. En sus inicios, la cantidad de dominios era de 19.732 (medición realizada en Agosto de 1995), la medición más actual (Febrero 2014) realizada por Netcraft[1] da un total aproximado de sitios de 920.102.079 [5] (58 millones más que el mes anterior). Esto se puede ver en el Gráfico 2.2.

2.3. El crecimiento de los Sitios

Otra variable que entra en juego es el aumento del tamaño promedio de los sitios así también como la cantidad de recursos que poseen los mismos. En el año 1997, el tamaño promedio de un sitio era de 60Kb [35], prácticamente era todo texto, pocas imágenes y poca interactividad. Esto fué cambiando con el tiempo, con el avance de la tecnología y de los recursos que se podían compartir en la red. El crecimiento a lo largo del tiempo puede verse en el Gráfico 2.3

[1]NetCraft - http://news.netcraft.com/

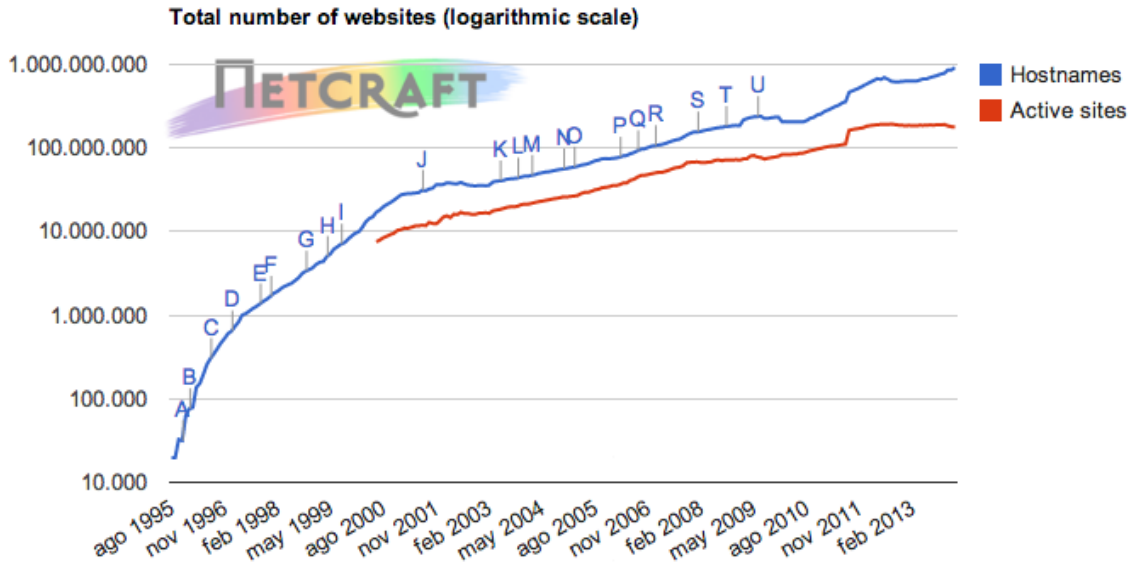

Figura 2.2: Crecimiento de la cantidad de sitios en Internet, extraído de [5]

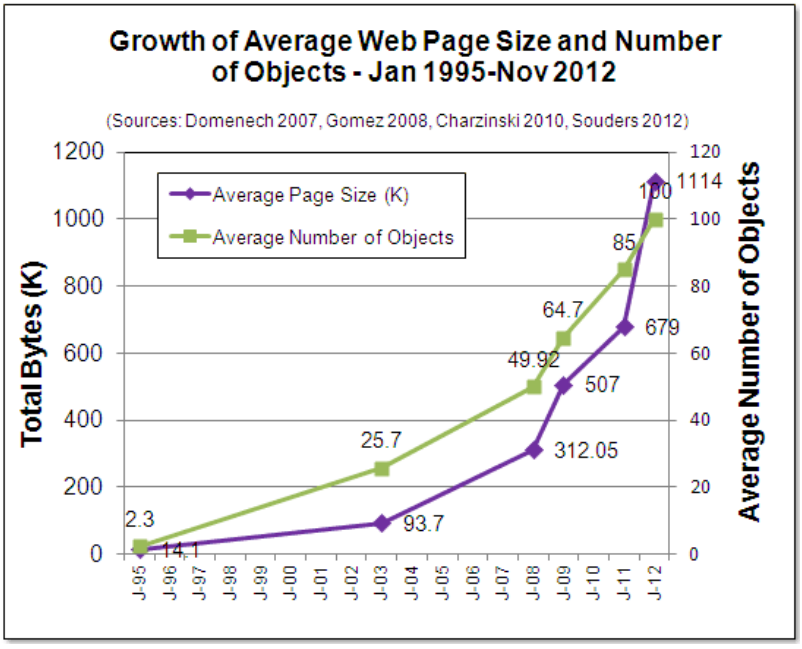

Figura 2.3: Crecimiento del tamaño de los sitios en Internet (1995 a 2012), extraído de [29]

Hoy en día, el contenido de los sitios es mucho más variado e interactivo. Se componen de diversos tipos de recursos, scripts, hojas de estilo, variados tipos de imágenes, video, contenido Flash[2], etc. Esto produce un aumento en el tiempo de carga de los sitios y la necesidad de poseer un dispositivo con la capacidad necesaria para procesar y renderizar[3] la página.

Comparando 2 medidas realizadas por HTTP Archive[8] con casi 4 años de diferencia, se observa que, en la primera medición el promedio es de 702Kb[4], el promedio detallado por contenido de los sitios se puede ver en el Gráfico 2.4. En la segunda medición realizada en Junio de 2014, se observa que el promedio es de 1808Kb, es decir, que hubo un aumento de un 157 %.

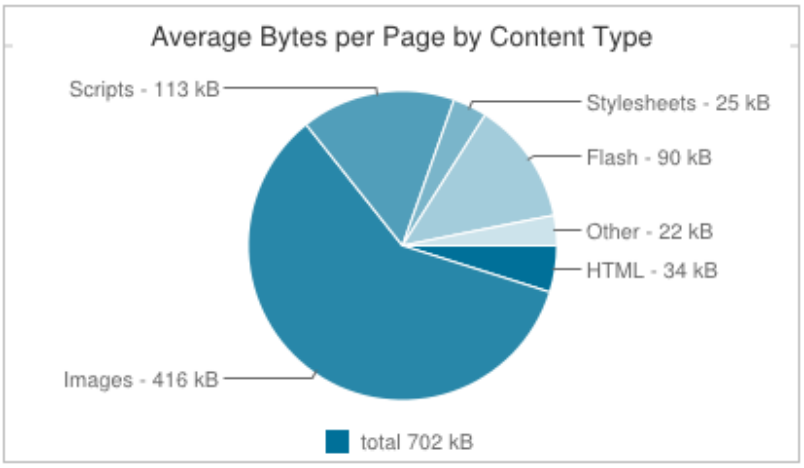

Figura 2.4: Tamaño promedio de los sitios en Internet (Noviembre 2010), detallado por contenido, extraído de [8]

Otra cuestión es el aumento de la utilización de HTTPS, la versión segura de HTTP (ver Capítulo 3), como se verá más adelante, este protocolo añade tiempo en la negociación previa a recuperar la página que debe hacer.

[2]http://www.adobe.com/
[3]El navegador "dibuja"la página según el código HTML y el estilo
[4]Medición extraída de [8] el 15 de Noviembre de 2010

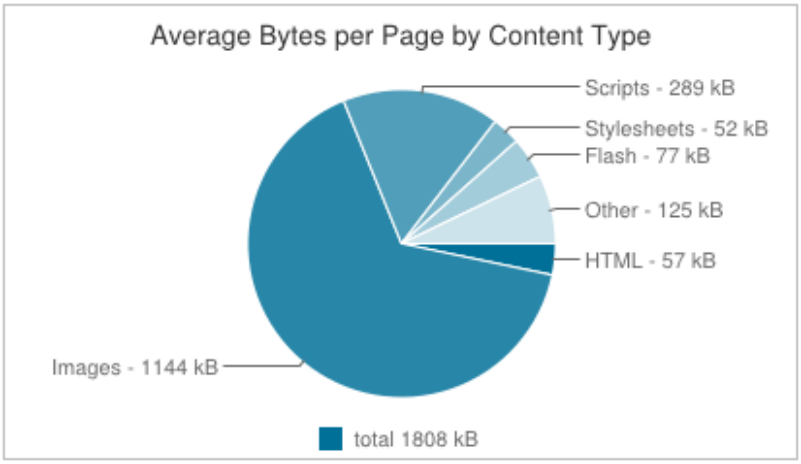

Figura 2.5: Tamaño promedio de los sitios en Internet (Junio 2014), detallado por contenido, extraído de [8]

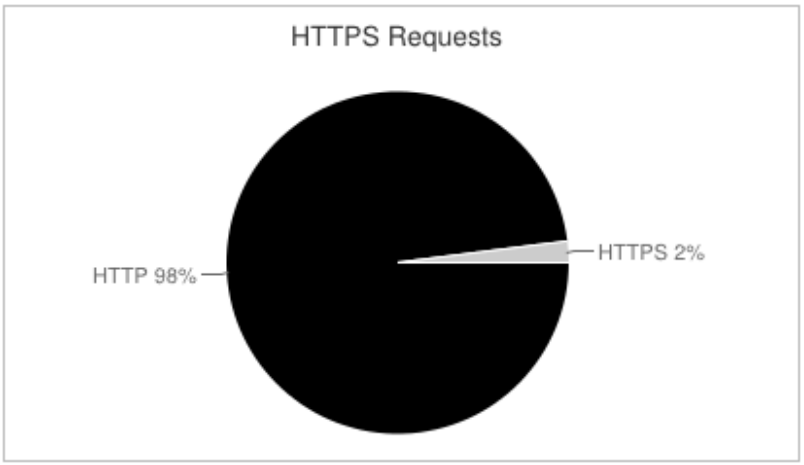

Figura 2.6: Porcentaje de utilización de HTTP/HTTPS (Noviembre 2010), extraído de [8]

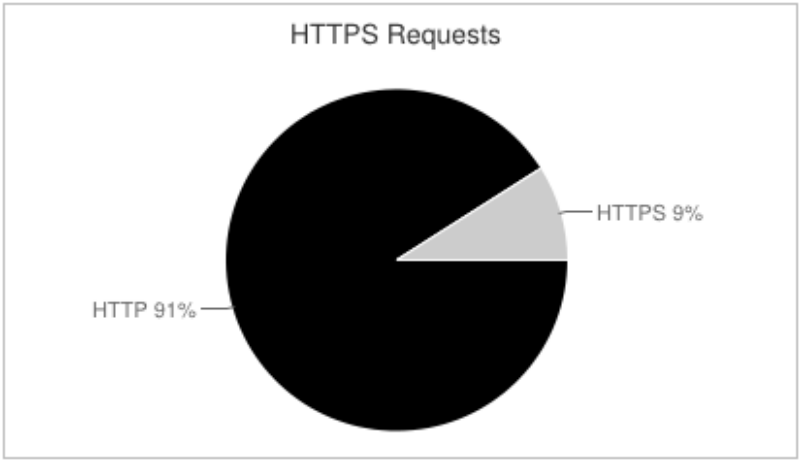

Figura 2.7: Porcentaje de utilización de HTTP/HTTPS (Junio 2014), extraído de [8]

Capítulo 3

Conceptos sobre Protocolos

3.1. Definición

Un protocolo de comunicación es un conjunto de reglas y normas de transmisión, que permite a dos o más entidades, comunicarse entre sí en un canal determinado. Es necesario que, antes de establecer una comunicación entre partes, se definan ciertos protocolos para poder asegurar la interoperabilidad y hacer posible la comunicación entre el emisor y receptor.

Las reglas definen la forma en la que debe efectuarse la comunicacin, incluyendo cuestiones como la temporización, secuencia, revisión y corrección de errores. Define una *sintaxis* (formato de los mensajes), una *semántica* (significado de los mensajes) y *sincronización* (secuenciamiento y temporización en la comunicación).

Para implementar los protocolos, se dividen las tareas a realizar y se realizan en niveles separados (capas). Definir los modelos en capas brinda las siguientes ventajas:

1. Dividir la comunicación en partes más pequeñas y sencillas

2. Normalizar los componentes de red para permitir el desarrollo y el soporte de los productos de diferentes fabricantes.

3. Permitir la interoperabilidad de diferentes tipos de hardware y software de red para comunicarse entre sí.

4. Impedir que los cambios en una capa puedan afectar a las otras capas. Facilita

la actualización del protocolo pudiendo modificar un módulo a reemplazarlo todo por completo.

3.2. Modelo OSI

En los inicios de Internet, hubo un gran crecimiento tanto en cantidad como en tamao de las redes. Debido a esto, muchas de las redes eran incompatibles entre sí, y era extremadamente complicada la comunicación entre ellas. Para solucionar este problema, la ISO[1], realizó ciertas investigaciones acerca de los esquemas de red. En 1980 [40], creó un modelo de red para ayudar a los diseadores de redes a poder implementar redes que puedan comunicarse entre sí y trabajar en conjunto. Este es el modelo de referencia OSI, definida en la ISO/IEC 7498-1 [24]. Este modelo se agrupa en capas, cada capa agrupa una función determinada. Están organizadas jerárquicamente, cada capa ofrece un servicio a la capa superior.

El modelo tiene 7 capas:

7. Aplicación
6. Presentación
5. Sesión
4. Transporte
3. Red
2. Enlace
1. Físico

Se puede ver su distribución en la Figura 3.1

[1]International Standard Organization

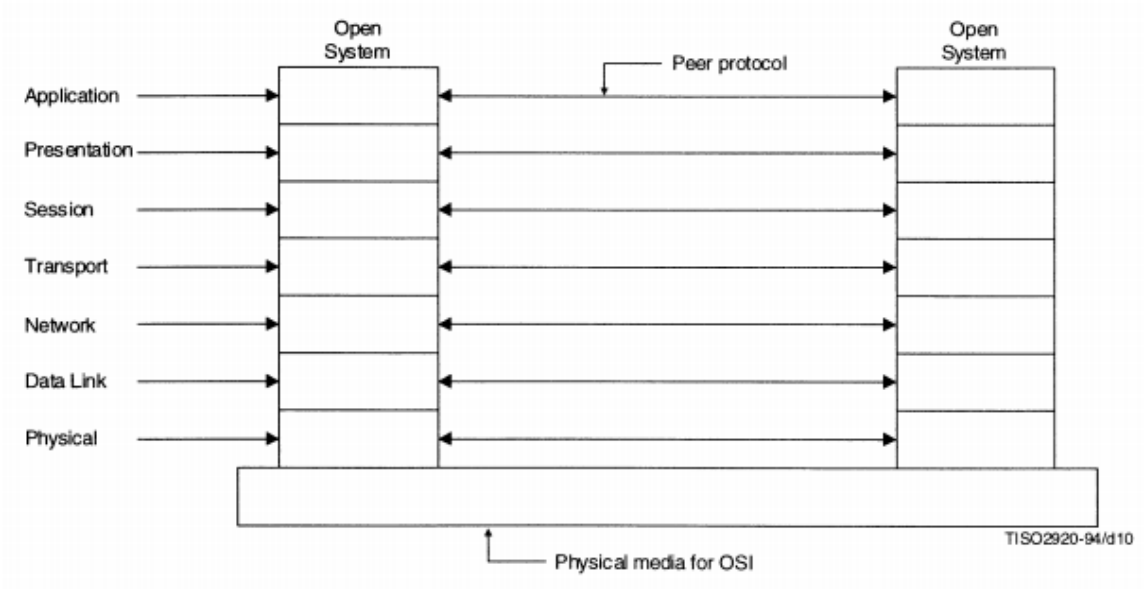

Figura 3.1: Gráfico de las capas OSI extraída de [24]

3.2.1. Capa de Aplicación

Proporciona servicios de red a las aplicaciones del Usuario, es la única capa que no provee servicios a otra capa. Si bien es la interfaz hacia el usuario, este no interactúa directamente con esta capa, sino que lo hace a travéz de una aplicación que si tiene acceso directo a esta capa. Entre los protocolos de esta capa se encuentran FTP, POP, SMTP, HTTP, HTTPS, SSH entre otros.

3.2.2. Capa de Presentación

Define el formato de los datos que se van a intercambiar entre las aplicaciones y ofrece a los programas de aplicacin un conjunto de servicios de transformacin de datos como podemos resumir definiendo a esta capa como la encargada de manejar las estructuras de datos abstractas y realizar las conversiones de representacin de datos necesarias para la correcta interpretacin de los mismos

3.2.3. Capa de Sesión

Esta capa establece, administra y finaliza las sesiones entre las partes intervinientes en la comunicación, a esto se le llama Servicio de Administración de la Sesión. También realiza el control del intercambio de datos y sincroniza el diálogo entre las partes, a esto se le llama Servicio de Administración del Diálogo.

3.2.4. Capa de Transporte

Esta capa provee un Servicio de Transporte Universal. Ofrece transparencia en el intercambio de datos entre los hosts involucrados (extremo a extremo) y aísla a las capas superiores de los detalles de implementación del transporte. Asegura que los datos enviados lleguen en el mismo orden en el que han sido enviados y sin errores, brindando calidad de servicio y confiabilidad en la comunicación.

3.2.5. Capa de Red

Se encarga de la conexión de hosts que pueden encontrarse en redes diferentes, define un esquema de direccionamiento, enrutamiento y selección de rutas. Permite que los datos viajen de un extremo a otro a travéz de redes interconectadas. Se utilizan los paquetes como unidad de información, estos paquetes son los que se rutean a travéz de la red para llegar del origen al destino.

3.2.6. Capa de Enlace

Proporciona una comunicación confiable entre equipos adyacentes. Se ocupa del direccionamiento físico, la topología de la red y el acceso a la misma. Se utilizan tramas como unidad de información, aquí se realizan los controles de flujo, controles de secuencia, y notificación de errores.

3.2.7. Capa Física

Esta capa provee las características procedurales, funcionales, eléctricas y mecánicas para establecer, mantener y cerrar conexiones físicas. Define como se transmiten los datos al medio, recibe mensajes y los transforma en bits para su posterior envío

a travéz de señales. Ciertas características tales como los conectores físicos, niveles de voltaje, duración de un bit, velocidad de los datos físicos, temporización y otros datos similares son definidos por las especificaciones de esta capa.

3.2.8. Funciones de los Protocolos

Entre las funciones de los protocolos se encuentran:

1. Encapsulamiento:

   Cada capa tiene lo que se conoce como PDU (Unidad de Datos del Protocolo), que es el resultado de la capa anterior con información propia. A medida que los datos van bajando a travéz de las capas se agregan encabezados y eventualmente una cola a los datos con información correspondiente a cada capa. Estos agregados contienen información de control para asegurar la entrega de los datos y la correcta interpretación de los mismos en el receptor. Una vez que se reúna la información de todas las capas, se convierten en bits y son enviadas por el medio físico. En la Figura 3.2 puede verse como se van acoplando las PDUs a medida que se avanza hacia abajo en el Modelo OSI.

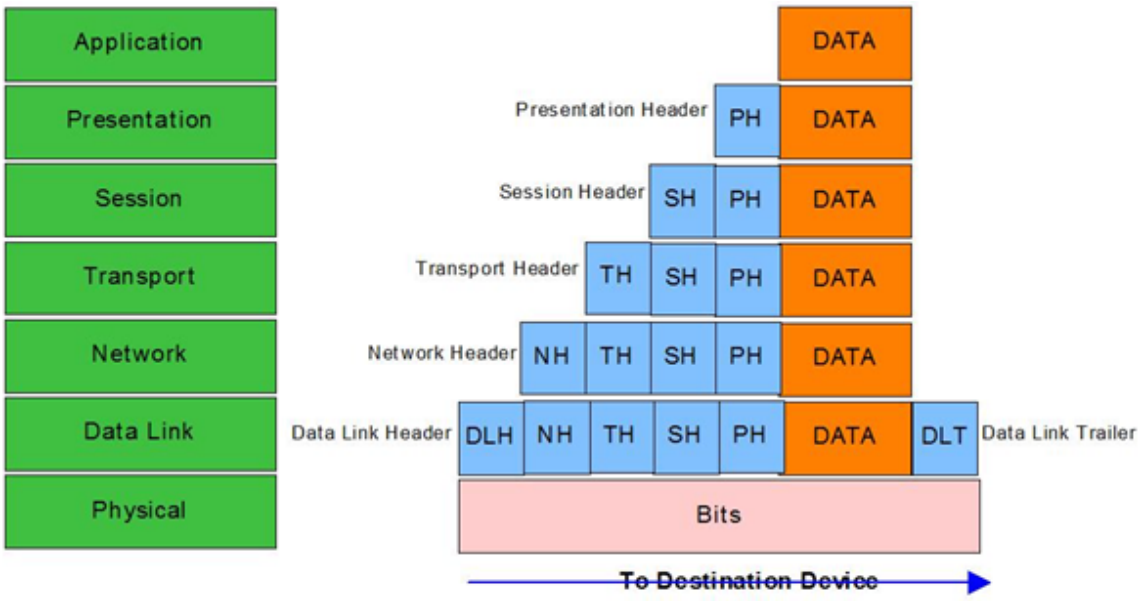

Figura 3.2: Encapsulamiento en el Modelo OSI, extraída de [10]

2. Establecimiento, control y cierre de la conexión.
3. Control de Flujo: Asegurar que la velocidad de los datos no sature las posibilidades particulares de cada capa.
4. Control de Errores: Detección y corrección de errores.
5. Multiplexación: Posibilidad de compartir el canal entre varias conexiones.
6. Encriptación y compresión.

3.3. TCP/IP

El conjunto de protocolos TCP/IP permite que computadoras de distintos tamaños, marcas, y diferentes sistemas operativos puedan lograr una comunicación entre sí [34]. Su desarrollo comenzó hacia los años 60', y en los 90'se convirtieron en los protocolos más utilizados en las redes. Conforma la base de Internet, la WAN[2] que conecta millones de dispositivos en todo el planeta.

Tiene un diseño en capas similar al del Modelo OSI (ver Sección 3.2), en donde cada capa provee una funcionalidad diferente para la comunicación. El modelo posee 4 capas, Aplicación, Transporte, Red y Enlace, como puede verse en la Figura 3.3.

3.3.1. Las Capas

1. Aplicación

   En esta capa se manejan los detalles de una aplicación particular. Aquí se encuentran protocolos muy conocidos tales como HTTP (Hipertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), SNMP (Simple Network Management Protocol), Telnet (para login remoto), DNS (Domain Name System).

2. Transporte

   Provee el servicio de envío de flujo de datos entre dos hosts. En esta capa se encuentran 2 protocolos muy importantes:

[2]Wide Area Network, Red de Área Amplia

Figura 3.3: TCP/IP

TCP (Transfer Control Protocol): Permite crear conexiones entre hosts para el envío de flujos de datos. Se ocupa de dividir los datos que vienen de la capa de aplicación en paquetes de un tamaño adecuado para realizar el envío en las capas inferiores. También se encarga del control de la recepción de los paquetes enviados, así también como el establecimiento de los tiempos de espera para asegurarse que el otro extremo reconoce los paquetes enviados. Realiza la multiplexación de la conexión, es decir, que permite compartir el canal entre varias conexiones.

Para establecer la conexión entre los 2 extremos, el servidor debe dejar en "escucha" una dirección en un puerto determinado. El cliente, envía un paquete SYN[3] inicial al servidor para iniciar la negociación, el servidor contesta con un paquete SYN-ACK[4] que es contestado por el cliente por un ACK. Una vez intercambiados estos 3 mensajes, el cliente puede empezar a realizar las peticiones al servidor. Esto se llama Three Way Handshake (Negociación de 3 vías), y puede verse en la Figura 3.4.

[3]SYNchronize
[4]ACKnowledgement

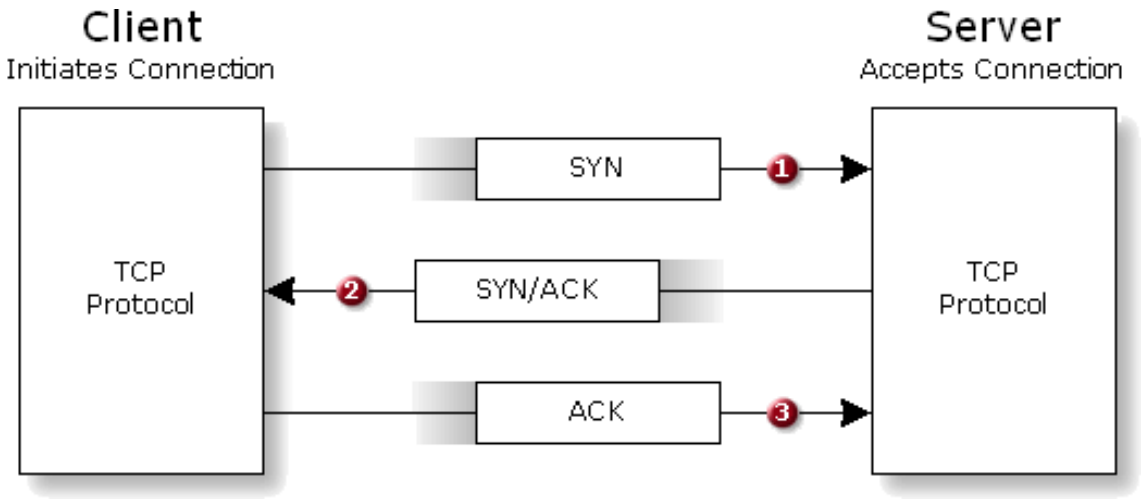

Figura 3.4: Three Way Handshake, extraído de https://www.grc.com/

UDP (User Datagram Protocol):

Se ocupa de enviar paquetes de datos llamados Datagramas de un extremo a otro, sin realizar una conexión previa, el mismo Datagrama posee suficiente información para llegar a destino. No brinda ninguna garantía de que el paquete llegue a su destino y tampoco hace control de flujo.

3. Red

   Esta capa, a veces llamada Capa de Internet, maneja el movimiento de los paquetes a través de la red. El enrutamiento y encaminamiento de los paquetes tiene lugar en esta capa. Protocolos como IP (Internet Protocol), ICMP (Internet Control Message Protocol) e IGMP (Internet Group Management Protocol) conforman esta capa.

4. Enlace

   Esta capa, a veces llamada capa de Enlace de Datos o capa de Interfaz de Red, incluye el Driver[5] del Sistema Operativo y la correspondiente Placa de Red del dispositivo. Juntos se encargan de todos los detalles del Hardware y de la interconexión física con el Medio (Cable, WiFi, Fibra Óptica, etc.).

[5]Controlador de Dispositivo

3.3.2. TCP/IP y el Modelo OSI

Similitudes

1. Ambos se dividen en capas.
2. Ambos tienen Capa de Aplicación, pero ofrecen servicios diferentes.
3. La Capa de Transporte y la de Red son similares.
4. La conmutación es por paquetes[6] (no por circuitos).

Diferencias:

1. En la Capa de Aplicación de TCP/IP se combinan las Capas de Presentación y de Sesión del Modelo OSI.
2. En la Capa de Enlace de TCP/IP se combinan las Capas de Enlace y de Física del Modelo OSI.
3. Al tener menos Capas, TCP/IP es más simple.
4. TCP/IP es el estándar de Internet, las redes no se desarrollan a partir del Modelo OSI, se utiliza como guía.

3.4. HTTP

Las siglas de este protocolo son por HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto). Es un protocolo de capa de aplicación (ver Sección 3.2) que sirve para distribuir información. Fué utilizado en la Web desde el año 1990 en su primera versión (0.9), en la que simplemente se podía transferir texto plano. Su evolución fue el estándar 1.0 en el que se mejoró el protocolo permitiendo que los mensajes usen el formato MIME - Multipurpose Internet Mail Extensions, se incorporaron metadatos acerca de la información transferida y modificadores en la semántica de petición/respuesta. La revisión del protocolo que se usa actualmente

[6]Método de envío de datos, cada paquete posee datos e información de control que indica la ruta a destino.

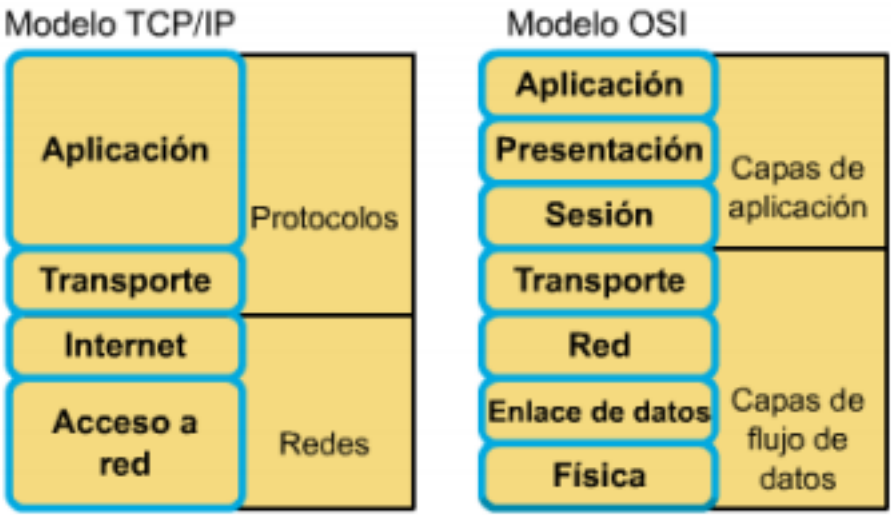

Figura 3.5: Comparación entre TCP/IP y OSI

es la 1.1, definida en la RFC 2616 [11]. Contiene nuevos metodos, headers y otras características. Las principales diferencias de esta última definición se pueden ver en [25].

El contenido web reside en Servidores, estos son los que se comunican utilizando este protocolo entre otros. Sirven RECURSOS, estos recursos pueden ser páginas HTML, imágenes, PDF's, video, etc, tanto contenido estático como dinámico (generado a demanda). Debido a la gran diversidad de contenido que provee un Servidor, es necesario identificar el tipo de recurso que se está enviando. Esto se hace utilizando una etiqueta llamada MIME-Type, que define el tipo de contenido a transferir.

Cada recurso del servidor tiene un nombre, para que los clientes puedan apuntar directamente al recurso deseado. Se nombra con una URL, que tiene el siguiente formato:

PROTOCOLO://SERVIDOR/PATH_AL_RECURSO/RECURSO

El funcionamiento básico es, el cliente envía una petición al Servidor (al puerto 80 por defecto) y este le responde. Esta comunicación se realiza a travéz de mensajes HTTP. Existen diferentes métodos que se pueden utilizar cuando se envía una petición al servidor, tales como

1. GET - El cliente solicita un recurso específico del servidor.

2. POST - El cliente envía datos que van a ser utilizados por el servidor.

3. HEAD - El cliente solicita sólo los Headers (se detallarán más adelante).

Estos son algunos de los métodos, hay otros tales como PUT, DELETE, etc. Según el método, el servidor opera de manera diferente. En la petición se envía el método, el recurso solicitado, la versión del protocolo utilizado, el host, el user-agent[7], entre otros. El Servidor, responde a la petición con una respuesta, que contiene un código de estado de 3 dígitos que le dice al cliente que la petición fue exitosa u otras, por ejemplo 200 (OK) o 404 (Documento no encontrado) de los más comunes.

Los mensajes de HTTP consisten en peticiones y respuestas, sus formatos son similares. Consisten en 3 partes:

1. Linea Inicial - Se indica que hacer en la petición o que fue lo que pasó en la respuesta.

2. Headers - Acá se pueden definir diferentes parámetros por cáda línea con la sintaxis "nombre:valor".

3. Cuerpo - Esta parte contiene los datos enviados, ya sea del cliente al servidor o viceversa.

El formato de una petición es el siguiente:

<método> <url del recurso> <versión>

<headers>

<cuerpo>

El formato de la respuesta es el siguiente:

<versión> <estado> <descripción del estado>

<headers>

<cuerpo>

[7]Quién está generando la petición, por ejemplo Mozilla o Safari (browser, proxy, etc.).

3.4.1. HEADERS

Los Headers, añaden informacion adicional a las peticiones y respuestas. El protocolo define varios Headers, pero se pueden inventar también, los servidores y clientes son libres de hacerlo. Hay diferentes tipos de Headers, entre los que se encuentran:

1. Headers Generales

   Pueden aparecer en peticiones y respuestas.

2. Headers de Peticiones.

   Proveen más información acerca de las peticiones.

3. Headers de respuesta.

   Proveen más información acerca de las respuestas.

4. Entity Headers (ENTIDAD?)

   Proveen información acerca del recurso del mensaje.

5. Headers de Extensión

   Permite agregar nuevos headers que no estén dentro de la especificación estándar [11].

La definición completa de los Headers se encuentra en la Sección 14 de [11].

3.5. HTTPS

Los usuarios de Internet, utilizan la web para hacer transacciones que requieren que el nivel de seguridad sea fuerte. Operaciones como realizar transacciones bancarias o hacer compras online, no se realizarían si los datos que el usuario envía al servidor viajan desprotegidos. Ante esta necesidad, se combina el protocolo HTTP con la tecnología de encriptación digital. HTTPS es la versión "segura" de HTTP, añade a este protocolo, una capa de cifrado utilizando SSL (el predecesor de TLS) sobre TCP, como se puede ver en la Figura 3.6. Se define en la RFC 2818 [7]. Se distingue el tráfico seguro del inseguro por la utilización de un número de puerto diferente.

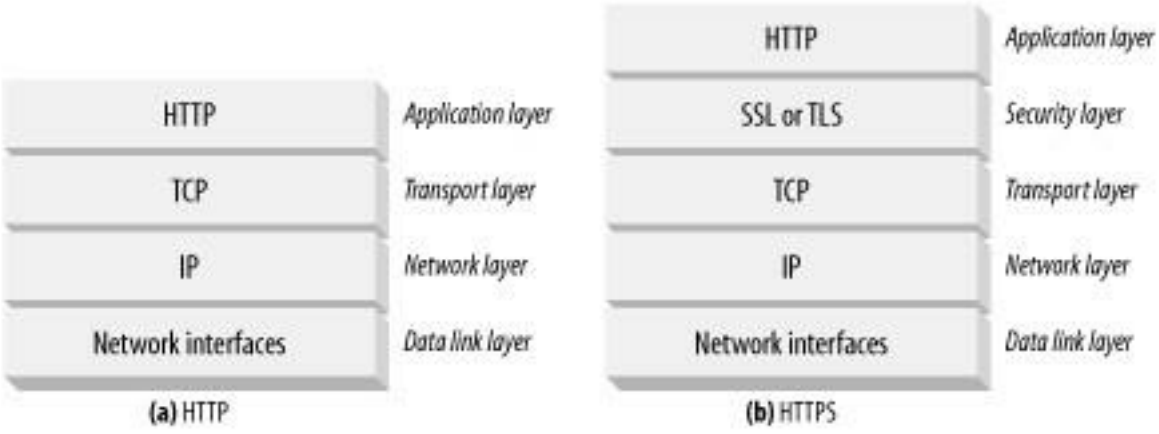

Figura 3.6: Comparación entre HTTP y HTTPS, extraído del libro [21]

Permite que los datos que viajan entre cliente y servidor vayan encriptados, esto se hace antes de enviar los datos por la red. Se distingue fácilmente porque el formato de la URL empieza con https:// y la conexión por defecto se establece con el puerto 443. Es decir, si el browser hace una petición a un servidor, en la cual, el esquema es https, se inicia la negociación para establecer la conexión segura con el mismo.

La conexión se hace con otro puerto diferente al de HTTP ya que SSL es un protocolo binario, completamente diferente. Si ambos llegaran al mismo puerto, los servidores interpretarian SSL como HTTP erróneo y cerrarían la conexión.

3.5.1. Criptografía

La criptografía es el arte y la ciencia de codificar y decodificar mensajes, alterando la representación lingu;istica de mensajes, buscando la confidencialidad y para prevenir la manipulación de los mismos. También se utiliza para probar quien fue el autor del mensaje o una transacción (no repudio).

Se basa en Algoritmos de Cifrado (Ciphers) que tiene un método para codificar el mensaje y otro para decodificarlo posteriormente. Comenzaron siendo algoritmos simples hasta que se empezaron a construir máquinas [8] para reforzar la seguridad de la encriptación haciendo operaciones más complejas. A causa de que estos algoritmos y máquinas podían llegar a manos no apropiadas, se incluyeron diversos métodos

[8]Ver Alemana Máquina Enigma, http://www.bbc.co.uk/history/topics/enigma

(llaves metálicas, diales de configuración), que actuan como entrada para que el algoritmo o máquina pudiera funcionar. De esta manera, aún teniendo el algoritmo o dispositivo, sin la clave no se puede decodificar.

En la Era Digital, los métodos (clave) para la entrada de los algoritmos de encriptación son simplemente números. Estos algoritmos son funciones que toman una porción de datos y la codifican/decodifican basados en el algoritmo de encriptación y la clave proporcionada.

La Criptografía puede ser Simétrica, es decir, que la clave para encriptar y desencriptar el mensaje es la misma. De esta manera, tanto el emisor como el receptor deben intercambiarse previamente la clave, antes de comenzar la comunicación. En la Criptografía de Clave Pública (ver Figura 3.7), las claves son Asimétricas, es decir, que la clave para encriptar difiere de la clave para desencriptar el mensaje. En este tipo de Criptografía, la clave para encriptar el mensaje es Pública, cualquiera que quiera establecer una conexión con seguridad, puede obtener la clave[9] para iniciar la comunicación. Por otro lado, la clave para desencriptar los mensajes es privada, pertenece al Servidor que es el único que puede decodificar los mensajes recibidos por el Cliente.

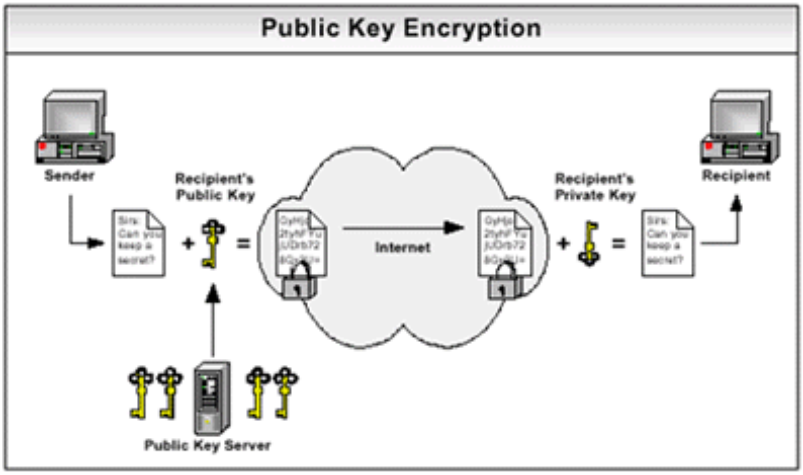

Figura 3.7: Criptografía de Clave Pública, extraído de http://www.novell.com/

[9] Almacenada en Servidores de Acceso Público

3.5.2. Conexión

El procedimiento para iniciar la conexión es el siguiente: El cliente abre una conexión con el puerto 443 al servidor. Una vez que la conexión TCP está establecida, el cliente y el servidor inicializan la capa SSL negociando algunos parámetros criptográficos e intercambiando llaves. Una vez concluida esta negociación, ya pueden empezar a intercambiar mensajes encriptados. Se puede ver el resumen en la Figura 3.8.

3.6. SPDY

3.6.1. Introducción

Desarrollado por Google a mediados del 2009, SPDY es un protocolo experimental cuya meta principal es reducir los tiempos de carga de los sitios enfocándose en las limitaciones de HTTP (ver 3.4). Específicamente se busca lo siguiente (extraído de [23]):

1. Reducir un 50 % el Tiempo de Carga de los Sitios.
2. Evitar la necesidad de que los desarrolladores tengan que realizar cambios a los sitios actuales.
3. Evitar cambios en la infraestructura de las redes y minificar la complejidad de implementación.
4. Liberar el desarrollo a la comunidad de código abierto.
5. Recopilar datos reales de rendimiento para validar o invalidar el protoclo experimental.

Ya para el 2012 el protocolo era soportado por Chrome[10], Mozilla Firefox[11], y Opera[12] y varios sitios populares (Google, Facebook, Twitter) ya ofrecían un cliente para la utilización del protocolo.

[10] http://www.google.com/intl/es_AR/chrome/browser/
[11] http://www.mozilla.org/es-AR/firefox/new/
[12] http://www.opera.com/

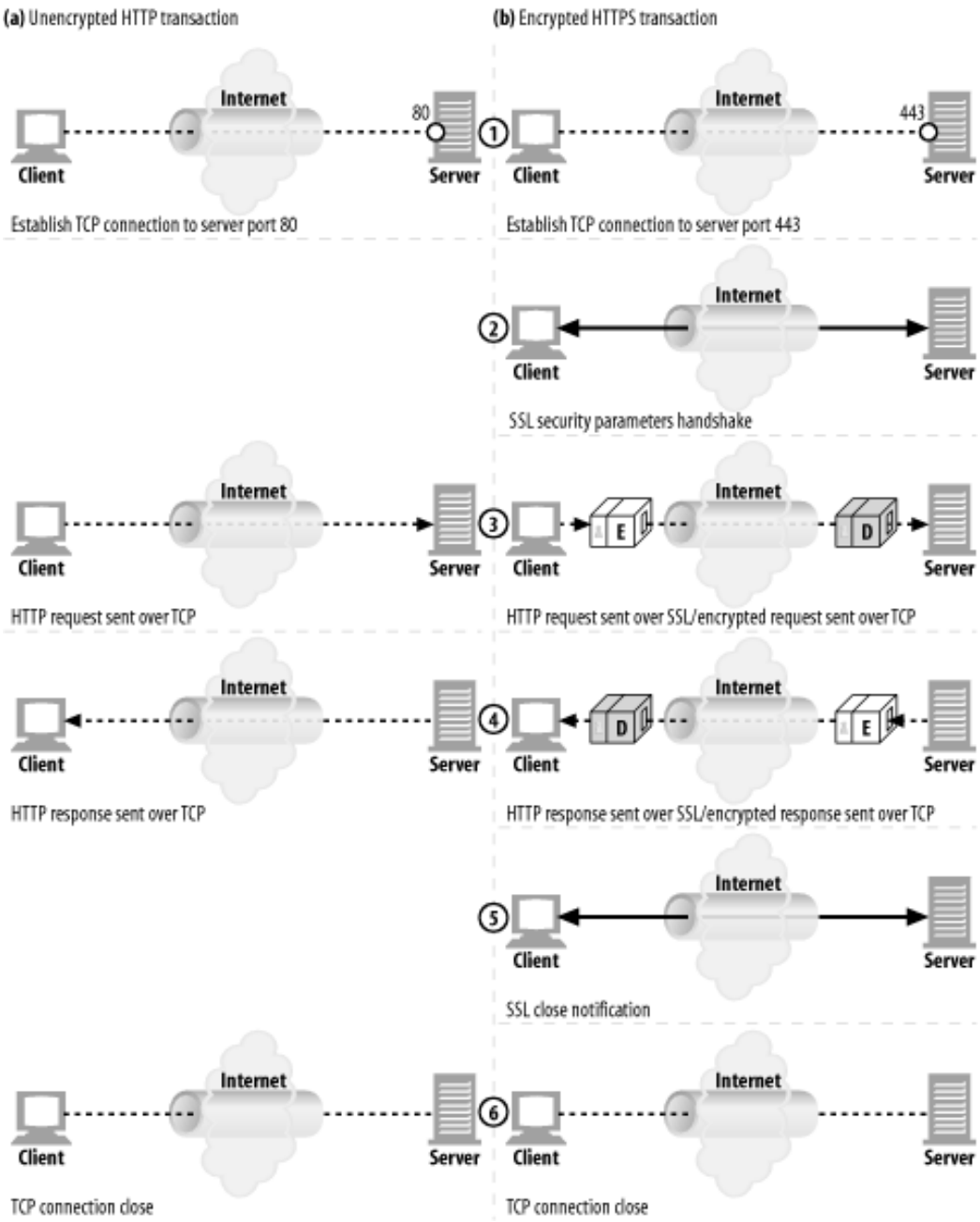

Figura 3.8: Establecimiento de Conexiones HTTP y HTTPS, extraído de [21]

3.6.2. El Protocolo

Es un protocolo de aplicación que añade una capa de sesión que funciona sobre SSL (ver Figura 3.9) que permite la transmisión de múltiples Streams[13] sobre una conexión TCP. Especifica un nuevo formato de trama para codificar y transmitir datos. Su especificación se puede ver en [12] y su draft se puede encontrar en [13].

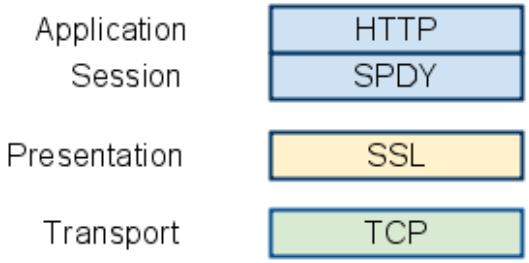

Figura 3.9: Protocolo de Aplicación SPDY, extraído de [12]

El protocolo HTTP no tiene estado, y, por cada recurso existe la necesidad de abrir una conexión nueva y cerrarla. Esto trae varios problemas. Por cada conexión nueva que se hace, se necesitan varios mensajes para establecer la conexión TCP, lo que trae varios RTT adicionales a la comunicación. Retrasos debido al "Slow Start"[14] de TCP. Clientes que evitan realizar múltiples conexiones con el mismo servidor (hasta 6 actualmente). A su vez, los servidores crean varios subdominios para almacenar el contenido para que los clientes puedan realizar las peticiones sin tener que evitar las múltiples conexiones a un mismo dominio.

El funcionamiento básico de SPDY 3.10 puede verse en la Figura 3.10. Inicia la conexión con el servidor realizando el ThreeWay Handshake de TCP (ver 2), se establece una conexión segura entre ambos extremos, y luego se pueden empezar a pedir los recursos a través de la misma conexión y en paralelo.

SPDY ofrece por sobre HTTP las siguientes mejoras:

1. Peticiones Multiplexadas. No existen límites de peticiones que se pueden realizar en una sesión de SPDY. A causa de que las peticiones son multiplexadas aumenta la eficiencia del protocolo TCP.

[13] Flujo de Datos.

[14] Se comienza enviando un volumen de datos pequeo hasta alcanzar cierto valor llamado Umbral de Congestión

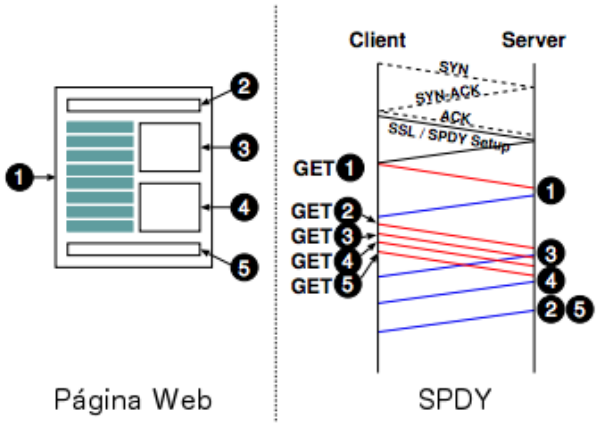

Figura 3.10: Funcionamiento de SPDY, extraído de [19]

2. Priorización de Peticiones. Los clientes pueden solicitar al servidor cuáles recursos quiere obtener antes que otros. Esto evita la congestión de recursos que no son críticos cuando todavía está pendiente el envío de algun recurso que tiene una prioridad mayor.

3. Compresión de Headers. A causa de que hoy en día los clientes envían mucha información redundante en forma de Headers, como la cantidad de peticiones para obtener un sitio promedio va desde 50 a 100, esta cantidad de información es relevante. Comprimir los Headers reduce el ancho de banda utilizado.

4. Server Push. Al permitir la comunicación bi-direccional a travéz de streams, cualquiera de los 2 (cliente o servidor) puede iniciar un stream hacia el otro. El servidor puede enviar un recurso al cliente antes de que este lo pida[15], esto reduce el tiempo de carga del sitio y disminuye la cantidad de peticiones del cliente.

5. Server Hint. El servidor puede "sugerirle" al cliente que pida un recurso en particular, ya que lo va a necesitar. De todas maneras, el servidor espera a que el cliente peticione el recurso en cuestión antes de enviarlo. Esto reduce el tiempo que tarda el cliente en descubrir cuales son los recursos que tiene que pedirle al servidor.

[15]El servidor conoce de antemano que el cliente va a necesitar el recurso en cuestión.

SPDY se enfoca en la manera en la que se transmiten los datos por la red, preserva toda la semántica del protocolo HTTP. De esta manera, para las aplicaciones se implementa de manera transparente, ya que reside entre la capa de aplicación y la de transporte. Esta sesión es similar al par petición-respuesta de HTTP. Es obligatoria la compresión del mensaje.

3.6.3. Alternativas Propuestas

Hubo varias alternativas que se propusieron para mejorar la performance de Internet,

Stream Control Transmission Protocol (SCTP)

Es una alternativa al Protocolo TCP, definido en la RFC 2960 [14], provee confiabilidad, control de flujo y secuenciación, opcionalmente permite el envío de mensajes sin un orden preestablecido. Permite Multihoming, que es la capacidad de que los extremos conectados puedan tener más de una dirección IP.

HTTP sobre SCTP

Fué una propuesta de utilizar HTTP sobre SCTP [16]

Structured Stream Transport (SST)

Es un Protocolo de Transporte experimental [20] diseñado para las aplicaciones que requieren de muchas conexiones asíncronas en paralelo, tales como la descarga de las diferentes partes que componen un sitio web y reproducir simultáneamente múltiples flujos de audio y video a la vez. No realiza el 3-way Hanshake en el inicio como TCP. Multiplexa múltiples Flujos de Datos de aplicaciones en una sola conexión de red. Soporta mensajes/datagramas de cualquier tamaño, no hay necesidad de limitar el tamaño de los envíos. Priorización de Flujos de Datos.

[16] http://tools.ietf.org/html/draft-natarajan-http-over-sctp-00

MUX y SMUX

MUX[17] y SMUX[18] son protocolos de capa intermedia (entre la capa de transporte y la capa de aplicación) que proporcionan la multiplexación de flujos de datos. Se propusieron al mismo tiempo que HTTP/1.1.

3.6.4. Estudios Relacionados

Se han realizado diversos estudios acerca de la performance de SPDY, que arrojan diferentes resultados. Estos resultados ayudan a ver en qué condiciones la utilización de SPDY realmente mejora la performance de la carga de un sitio.

Jitendra Padhye y Henrik Frystyk Nielsen, en su Paper "A comparison of SPDY and HTTP performance" [31], realizaron una comparación de ambos protocolos en un ambiente controlado. Con un sitio de prueba[19], variando el RTT y el Ancho de Banda, obtuvieron los resultados que se ven en los Gráficos 3.11 y 3.11.

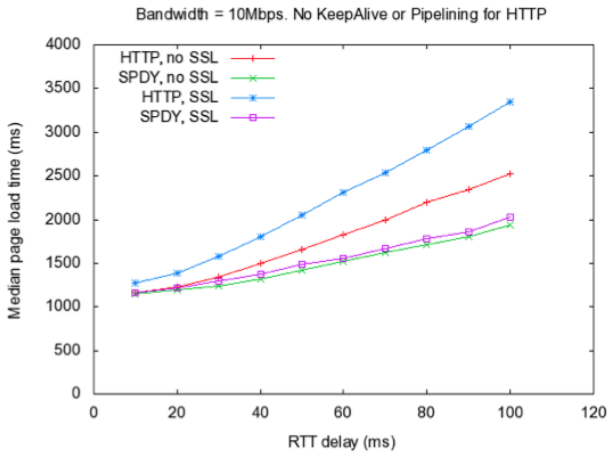

Figura 3.11: SPDY vs HTTP - 10Mbps, extraído de [12]

Claramente se observa que, con Anchos de Banda más limitados, el incremento de performance de SPDY es bajo (del 3 % al 8 %) frente a un Ancho de Banda de 10mb en el cual el incremento es de hasta 39 %. Lo cual es un indicador de que, en un ambiente en el cual la velocidad del enlace sea pobre (por ejemplo una red 3G),

[17] http://www.w3.org/Protocols/MUX/
[18] http://www.w3.org/TR/WD-mux
[19] index.html + 20 imágenes + 3 hojas de estilo

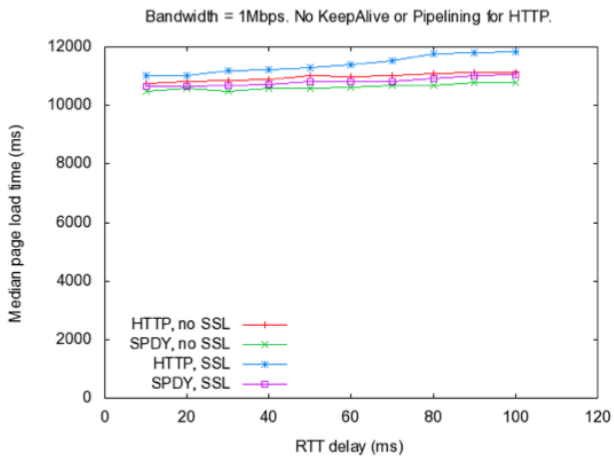

Figura 3.12: SPDY vs HTTP - 1Mbps, extraído de [12]

que causa que esté sujeto a una mayor probabilidad de pérdidas de paquetes, SPDY no funciona como lo esperado. Posiblemente a causa del alto costo de retransmisión del Flujo de Datos, ya que si el Stream se pierde, se debe retransmitir todo completo ya que usa una sola conexión entre los extremos para comunicarse.

Otro estudio interesante, relacionado con los resultados del estudio anterior, es "Towards a SPDYier Mobile Web?" [19]. En este paper se estudió el rendimiento de SPDY vs HTTP en redes de celulares. También concluyen que no hay una diferencia significativa entre los protocolos estudiados. en ese ambiente.

FALTAN:
SPDY Accelerator for Improving Web Access Speed
The Effect of Network and Infrastructural Variables on SPDY?s Performance
How Speedy is SPDY?

Capítulo 4

Problemáticas asociadas a la Web

4.1. Introducción

Debido al crecimiento de Internet visto en el Cápitulo 2 y a los problemas que presenta el protocolo HTTP en su implementación (visto en el Cápitulo 3), mejorar los tiempos de carga de los sitios web es clave.

VER http://munchweb.com/effect-of-website-speed

4.2. Performance

Cada aplicación se desarrolla con los requerimientos del modelo de negocio, el contexto, las expectativas de los usuarios y la complejidad de la tarea. A causa de esto, hay que planificar y diseñar, centrándose en el usuario, y su percepción del tiempo de procesamiento [23]. Nuestros tiempos de reacción son constantes, independientemente del tipo de aplicación o dispositivo utilizado, puede verse en la tabla siguiente:

Retardo	Percepción del Usuario
0-100ms	Instantánea
100-300ms	Retraso pequeño perceptible
300-1000ms	La máquina se encuentra procesando
1,000+ ms	Distracción en lo que se está realizando
10,000+ ms	La tarea es abandonada

Percepción del usuario ante los retardos, extraído de [23].

Para que la experiencia del usuario sea positiva, el sitio debe renderizarse, o al menos tener una respuesta visual de que el sitio se está cargando, en al menos 250 milisegundos.

En agosto del 2010, Mike Belsche[1] publicó un estudio [17] acerca de 2 factores importantes en el Tiempo de Carga de los Sitios, Ancho de Banda y RTT. El RTT[2] es el tiempo que demora un paquete en ir del emisor al receptor y volver al emisor nuevamente. Realizó 2 experimentos, el primero variando el Ancho de Banda y el segundo variando el RTT.

En el Gráfico 4.1 puede observarse el resultado del primer experimento, al aumentar el Ancho de Banda hay mejoras en el tiempo de respuesta en los valores más bajos, pero, a partir de cierto ancho de banda, el aumento no produce mejoras en el rendimiento.

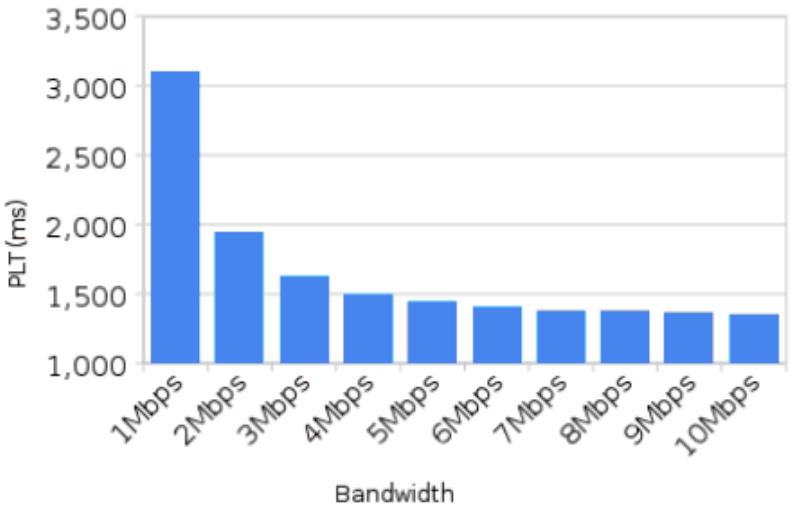

Figura 4.1: Latencia por Ancho de Banda, extraído de [17]

En cambio, en el Gráfico 4.2, se ve claramente que a medida que el RTT se va reduciendo, tambiíen lo hace el Tiempo de Carga del Sitio, incluso cuando ya el RTT es bajo.

[1] Ingeniero de Software, fué uno de los desarrolladores del protocolo SPDY (ver Sección 3.10) - https://www.belshe.com

[2] Round Trip Time

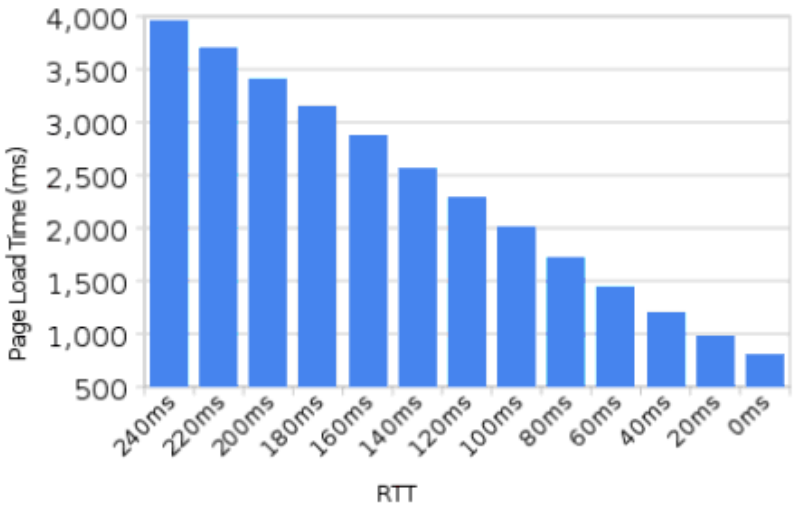

Figura 4.2: Tiempo de Carga de un sitio a medida que el RTT se decrementa, extraído de [17]

La conclusión del estudio de Belshe es que, si se duplica su Ancho de Banda sin reducir el RTT significantemente, sólo se obtiene una mejora mínima en la Navegación. Disminuir el RTT, sin importar el Ancho de Banda utilizado, siempre supone una mejora en la velocidad. Para acelerar Internet, o bien hay que ocuparse de reducir el RTT, o reducir la cantidad de RTT's requeridos para cargar un sitio.

Es importante también conocer, dónde es que el usuario pasa el tiempo esperando en la carga de un sitio web. Según el estudio de Steve Souders en su libro [33], el cliente tarda menos del 20 % para obtener el documento HTML, y el tiempo restante para recibir el resto de los componentes del sitio. Es importante enfocarse en el 80 %, 90 % restante, ya que el tiempo no se desperdicia en descargar el documento HTML ni en el procesamiento que realiza el servidor antes de enviarnos la petición. Esto se resume en la "Regla de Oro de la Performance" de dicho libro:

> *Solo el 10-20 % del tiempo de respuesta del usuario es consumido descargando el documento HTML. El otro 80-90 % se consume descargando todos los componentes de la página.*

4.3. Técnicas de optimización

Souders, plantea 14 reglas para la optimización de los sitios que se describen a continuación.

1. **Minimizar la cantidad de peticiones.**

   La idea básica es eliminar las peticiones al servidor, esto se hace utilizando varias técnicas enumeradas a continuación.

   a) Mapa de Imágenes - Permite asociar múltiples áreas en una imagen para que se puedan cliquear y realizar cierta acción (el ejemplo más básico es el de navegar un hipervínculo). En vez de utilizar múltiples imágenes para realizar un menú por ejemplo, se utilizaría una sola mapeada. De esta manera se ahorran las peticiones de las imágenes por 1 sola petición.

   b) CSS Sprites - Permite combinar várias imágenes en una sola, luego en el sitio, se define que parte de la imagen combinada desea mostrarse. Por ejemplo, en el sitio de Google cuando se realiza una búsqueda, se utiliza el Sprite que se ve en la Figura 4.3.

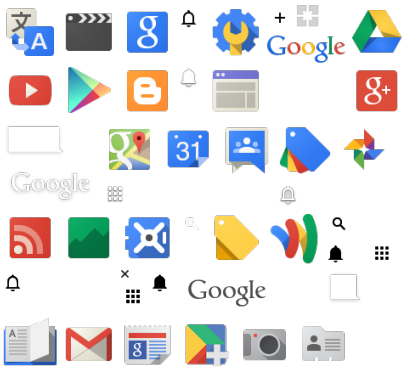

Figura 4.3: Sprite del sitio www.google.com

   A simple vista se ven 44 imágenes combinadas. La utilización de este Sprite reduciría todas estas peticiones a 1 sola.

   c) Imágenes Inline - Se pueden incluir imágenes en un sitio web sin necesidad de realizar una petición al servidor definiendolas con el esquema *data:* **??** dentro del código HTML de la página.

d) Combinar Scripts y Hojas de Estilo - Como la mayoría de los sitios actuales utilizan Javascript y CSS, es preferible que los Scripts se combinen en uno solo al igual que las Hojas de Estilo. En el caso de tener varios archivos, el navegador va a generar una petición por cada uno de ellos de no tenerlo almacenado en una copia local.

2. **Utilizar un CDN**

   La proximidad del cliente al servidor web tiene un impacto en el tiempo de respuesta de las peticiones. No es lo mismo solicitar un recurso localizado China estando en Argentina que uno ubicado dentro del mismo país. Por ende, si los contenidos están cerca[3] el tiempo de respuesta es menor. Debido a que solo el 10-20 % del tiempo de respuesta se dedica al HTML (visto al inicio de este capítulo), si el resto de los recursos del sitio se encuentran cerca del cliente, se mejorarían los tiempos de respuesta. Para esto es necesario dispersar estos recursos geográficamente.

   Un CDN[4] es una red de distribución de contenidos. Son servidores dispersados geográficamente que ofrecen réplicas de los recursos de un sitio particular para brindarlos al cliente desde el más cercano a su locación.

   Utilizar el servicio que brinda un CDN mejora los tiempos de respuesta de los usuarios.

3. **Añadir Headers para Cachés**

   Cuando un usuario visita el sitio por primera vez, realiza tantas peticiones HTTP como recursos tenga la página. Utilizando Headers Expires o Cache-Control[5] (Capítulo 3, Sección 3.4.1) se puede hacer que los recursos puedan ser almacenados en Cachés (Capítulo 6). Esto disminuye la cantidad de peticiones en una posterior visita a la página del mismo usuario.

   La performance del tiempo de respuesta del sitio mejora según la cantidad de "Hits"[6] que el usuario tenga de los componentes del sitio.

[3] En términos de distancia física.
[4] Content Delivery Networks
[5] Desde HTTP 1.1
[6] Necesidad de solicitar un recurso que ya tengo almacenado en la Caché

4. **Comprimir los componentes (Gzip)**

   Se puede reducir el tiempo de respuesta, disminuyendo el tamaño de la respuesta HTTP. Esto se puede realizar comprimiendo el recurso que se está solicitando. La reducción del tiempo es mayor en ambientes en donde el ancho de banda es bajo. El formato *Gzip*[7] [6] es el más popular y el más efectivo, el otro formato que se usa con menos frecuencia es *deflate* [2].

   El usuario tiene que enviar en la petición un Header *Accept-Encoding* indicando los métodos aceptados. Cuando el servidor recibe este Header, puede comprimir el recurso utilizando alguno de los métodos indicados por el usuario. Este devuelve la respuesta con un Header *Content-Encoding* indicando el tipo de compresión utilizado en el recurso que se está enviando.

   Los tipos de archivos que se deberían comprimir son aquellos de texto, tales como HTML, Scripts, CSS, etc. Aquellos formatos que ya se encuentran comprimidos como las imágenes o los archivos PDF no deberían comprimirse ya que se desperdicia tiempo de CPU del servidor[8] y además puede incluso incrementar el tamaño del archivo.

5. **CSS en la parte superior del HTML**

   Las Hojas de Estilo deben incluirse en la parte superior del HTML. De esta manera, los navegadores pueden ir renderizando la página a medida que van llegando las respuestas de las peticiones. Esto es importante en términos de usabilidad, para brindarle un medio visual [9] al usuario que está esperando el sitio.

6. **Scripts en la parte inferior del HTML**

   Cuando se realizan las peticiones al servidor, las descargas pueden hacerse en paralelo con ciertos límites[10], eso claramente es beneficioso, ya que al paralelizar las descargas, el tiempo es menor comparado con descargas secuenciales. En el caso de los Scripts, esta característica se deshabilita por 2 motivos, uno es

[7] http://www.gzip.org/
[8] Para realizar la compresión.
[9] Carga progresiva del sitio.
[10] 2 según la especificación HTTP 1.1, hasta 6 según el navegador

que si es script altera contenido de la página, el navegador debe esperar a recibirlo para mostrar el contenido correctamente. El otro motivo es que el navegador debe respetar el orden de ejecución de los scripts, si estos vinieran en paralelo, no se puede asegurar que su ejecución tenga el mismo orden en el que se solicitaron.

7. **Evitar expresiones en CSS**

   Las expresiones en CSS se encuentran deprecadas[11] en los navegadores modernos. Afectaban directamente a la performance de la renderización del sitio una vez que todos los componentes eran recibidos.

8. **Utilizar JavaScript y CSS de manera externa (no embebido en el HTML)**

   Al embeber los scripts y el estilo en el HTML, se minimiza la cantidad de peticiones al servidor, pero se aumenta el tamao del archivo HTML. Incluir archivos externos al HTML, permite a los navegadores y proxys que puedan almacenar en su caché el objeto. Esto es útil ya que si en todas las páginas del sitio se utilizan el mismo Javascript y CSS, al ir navegando las diferentes páginas del sitio, estos recursos ya están almacenados en el navegador. También disminuye el tiempo de respuesta en visitas posteriores.

9. **Reducir las búsquedas de DNS**[12]

   Cada servidor tiene una dirección IP asignada, esta dirección se encuentra asociada a un nombre de Dominio, esto se almacena en los DNS. Cuando se ingresa un nombre de un sitio en el navegador, este necesita la dirección IP asociada a ese Dominio. Para ello necesita solicitar esa asociación a un DNS Resolver, antes de empezar a solicitar los recursos debe esperar la respuesta del DNS. Esto al igual que los recursos se almacenan en una caché local del navegador. A raíz de esta cuestión, tener menos dominios en las URL's de los componentes de una página, requiere menos peticiones (DNS lookups) a los servidores que almacenan los dominios para solicitar las direcciones IP que corresponden a esos dominios.

[11] http://blogs.msdn.com/b/ie/archive/2008/10/16/ending-expressions.aspx
[12] Domain Name System.

10. **Minificar el JavaScript**

    La Minificación, es la práctica de remover caracteres innecesarios del código para reducir el tamaño del archivo. Se quitan los comentarios, los espacios en blanco (tabulaciones, espacios, saltos de línea). Una optimización alternativa tiene el nombre de Ofuscación, además de lo que hace la Minificación, renombra las variables del codigo por cadenas de texto más pequeñas. Este método disminuye aún más el tamaño de los archivos, que el método visto anteriormente. Con esta optimización se mejora el tiempo de respuesta ya que los recursos tienen menor tamaño.

11. **Evitar redirecciones**

    Una redirección es utilizada para enrutar a los usuarios de una URL a otra. Generalmente se utilizan para documentos HTML, pero, ocasionalmente, también cuando se peticionan componentes de la página. Los retrasos ocasionados por una redirección retrasan directamente el documento HTML completo. Insertar una redirección entre el usuario y el HTML retrasa todo en el sitio.

12. **Remover Scripts duplicados**

    Incluir archivos Javascript más de una vez en un sitio, afecta directamente la performance por 2 factores, peticiones innecesarias y tiempo de procesamiento innecesario del Script.

13. **Configurar Etags**

    Los Etags son cadenas de texto únicas que identifican una versión específica de un componente (ver Capítulo 6, Sección 6.4) . Provee un mecanismo para poder realizar una petición condicional al servidor, comparando el Etag de la copia local con el del servidor. En el caso de que el valor coincida con el del servidor, la respuesta devuelve un código (ver 3.4) que indica que el recurso está fresco (ver 6), de esta manera, se reduce notablemente el tamaño de la respuesta y el tiempo ya que viaja por la red solamente los Headers.

    Hay ciertas cuestiones referidas a la utilización de los Etags. Por un lado, en el caso de un cluster de servidores, los Etags generados por los mismos, no son iguales entre ellos. En este caso, si se realiza una petición condicional,

a pesar de que el componente es igual, al no coincidir los Etags, el recurso es devuelto al usuario. Otra cuestión es que la petición condicional *If-None-Match* toma precedencia ante *If-Modified-Since*, según la especificación del protocolo HTTP 1.1 si ambos Headers se encuentran en la petición, ambas condiciones se tienen que cumplir para devolver un código 304. Por todas estas cuestiones, se recomienda, o bien configurar los Etags de manera que las peticiones sean precisas, o quitarlos.

Capítulo 5

Proxy

5.1. Definición

Los servidores proxy son Intermediarios. Se encuentran entre el cliente y el servidor, actúan enviando los mensajes del cliente al servidor y viceversa. En una comunicacin normal, le cliente se comunica directamente con el servidor, en el caso de que en la red haya un proxy presente, el cliente se comunica con el proxy y este es el que se comunica con el servidor.

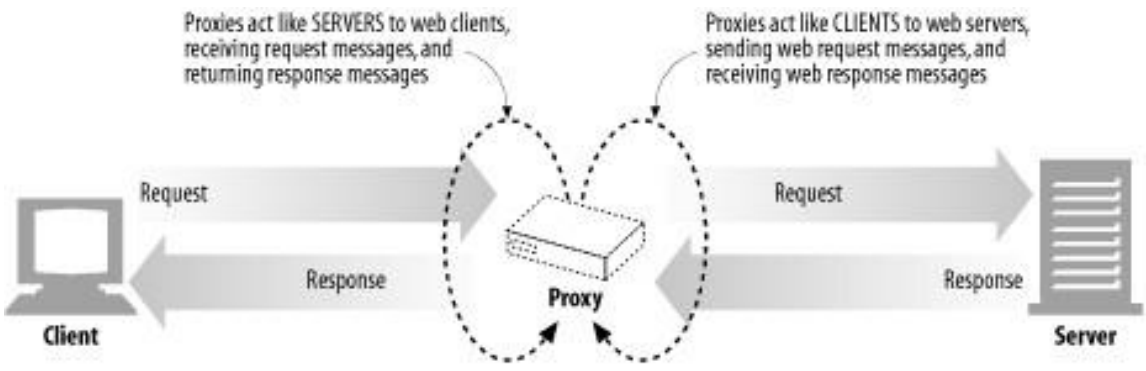

Figura 5.1: Esquema del funcionamiento de un Proxy, extraído de [21]

El proxy es un web server y también es cliente. Para recibir los pedidos de los clientes, tiene que actuar como un servidor y manejar correctamente las peticiones, conexiones y respuestas. A su vez, para poder conectarse a los destinos finales y recuperar los recursos que le son pedidos por el usuario del proxy, tiene que actuar

como un cliente, enviando peticiones y recibiendo las respuestas. Se puede ver el funcionamiento básico de un Proxy en la Figura 5.1.

El esquema de la Figura 5.1 muestra una conexión utilizando HTTP, las peticiones viajan directamente al Proxy y de ahí al servidor final. En el caso de HTTPS se hace de manera diferente. Primero el Cliente envía una petición con el Método CONNECT que incluye la dirección y el puerto del destino final. El proxy autentica la conexión, completa la negociación con el destino y responde al cliente con un mensaje que contiene el código 200 que indica que la conexión fué establecida. Luego, el proxy se convierte en un túnel que solo reenvía los paquetes del cliente hacia el servidor y viceversa (todo el tráfico se encuentra encriptado). Esto puede ilustrarse en la Figura 5.2. Los browsers actuales no soportan Proxies HTTP Seguros, es decir, que se establezca una sesión SSL del Cliente al Proxy y otra del Proxy al Servidor Final como puede verse en la Figura 5.3. Este último punto está en discusión, se propuso un borrador [4] al respecto.

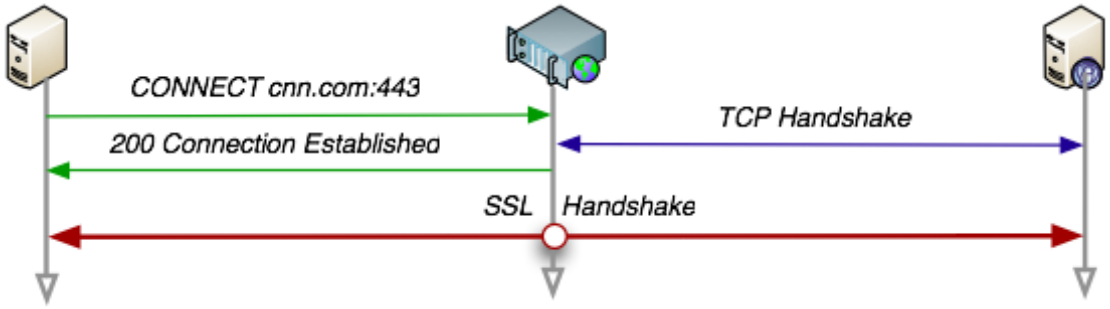

Figura 5.2: Túnel HTTPS sobre un Proxy, extraído de [22]

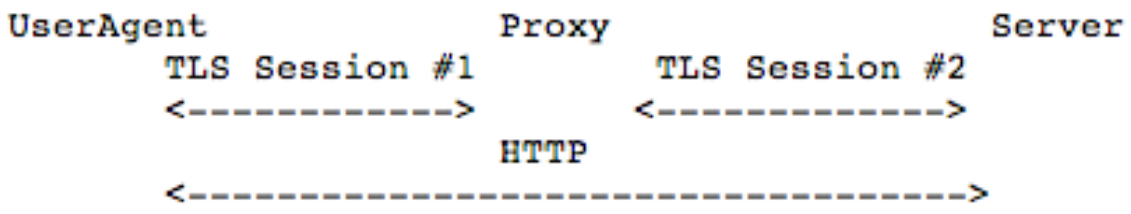

Figura 5.3: Túnel HTTPS sobre un Proxy, extraído de [4]

5.2. Usos de un Proxy

Hay diversas utilizaciones de los servidores proxy, entre ellas se destacan:

1. Proxy NAT[1] / Enmascaramiento

   El enmascaramiento IP o también llamado traducción de direciones de red, es el proceso por el cual las direcciones IP del origen/destino son reescritas, se sustituyen por otras. Es útil cuando se dispone de una única dirección pública que se comparte entre varios usuarios. El proxy se encarga de enmascarar las direcciones privadas de los clientes, traduciendolá a la dirección pública, para realizar las peticiones al exterior. Cuando recibe la respuesta de los servidores exteriores, se encarga de derivarla al usuario que inició el pedido.

2. Filtro

   El Proxy al recibir las peticiones de los clientes, puede permitir o denegar esas peticiones según ciertas políticas definidas. Por ejemplo, en una Institución Educativa se podría bloquear el acceso a ciertas redes sociales o sitios para adultos.

3. Cortafuegos[2]

   Al ser un intermediario, y muchas veces como puerta de acceso hacia internet, se pueden utilizar para aumentar el nivel de seguridad restringiendo ciertos protocolos o filtrando cierto contenido inseguro por ejemplo.

4. Web Caché

   Puede utilizarse para mantener copias locales de los recursos peticionados por los clientes, y al momento de recibir una petición por un recurso que se encuentra almacenado, se brinda esa copia al cliente. Al servir varios clientes esto incrementa la velocidad de respuesta. Esto se verá detalladamente en el Capítulo 6.

5. Proxy Reverso / Surrogate Server

[1] Network Address Translation
[2] Firewall

Puede actuar delante de un Servidor Web, atendiendo peticiones de los clientes como si fuera él Servidor mismo. Esto permite por ejemplo, generar una Red de Distribución de Contenidos[3]. Recibe una petición de un cliente, y, en vez de devolver el contenido directamente, inicia una comunicación con otros servidores para localizar el recurso pedido, de manera más eficiente. También protege a los Servidores Web, añadiendo una capa adicional de defensa. Puede distribuir la carga entre varios Servidores Web.

6. Content-Router

   Pueden redirigir las peticiones a diferentes Servidores, basados en las condiciones del Tráfico de Internet y el tipo de contenido.

7. Transcoder

   Puede modificar el contenido de los recursos antes de hacer el envío a los clientes. Por ejemplo, transformar imágenes BMP a JPEG para reducir el tamaño, comprimir archivos de texto, e incluso hacer una traducción del contenido a otro lenguaje.

8. Anonimizador

   Permite navegar de manera privada y anónima removiendo ciertos datos de los mensajes HTTP (Dirección IP, From Header, cookies, etc).

5.3. Ventajas y Desventajas

5.3.1. Ventajas

1. Control

   Al ser el intermediario, es el Proxy el único que va a realizar el trabajo de comunicación hacia el exterior, es el que tiene la potestad de limitar y restringir los derechos de los clientes, ya que todo pasa a través de él.

2. Optimización de los tiempos de carga

[3]CDN

En el caso de los Proxies que ofrecen servicio de Caché, se optimizan los tiempos de carga de los sitios, al tener almacenada copias locales que puede bridar directamente al usuario sin tener que realizar la petición al servidor final.

3. Optimización del uso de recursos

   El Proxy es el único que realiza el trabajo hacia afuera, es decir, que al utilizar una sola conexión, se maximiza el uso del ancho de banda.

4. Modificación de la Información

   Al tener acceso a los recursos que viajan a través de él, tiene la posibilidad de modificarlos.

5. Filtrado

   Recibir todas las peticiones del usuario, da la posibilidad de poder decidir que recursos o qué sitios son los que están habilitados para su recuperación. Podría manejarse una Lista Negra[4] o una Lista Blanca[5] por ejemplo.

6. Tráfico controlado

   Debido a que todo el tráfico pasa a travéz del Proxy, se puede registrar un gran volumen de información que luego puede usarse para Auditoría y Seguridad.

7. Anonimato

   Permite a los usuarios acceder a los servicios que brinda Internet, protegiendo la red interna. Al acceder al exterior identificándose como el mismo, es difícil para el recurso diferenciar quien es el que está realizando la petición.

5.3.2. Desventajas

1. Múltiples funciones

   Al estar a disposición de todas las peticiones de los usuarios, es posible que tenga que realizar algunas tareas no específicas de su función, por ejemplo los controles de acceso a sus servicios.

[4]Listado de sitios a los que NO se puede acceder
[5]Listado de sitios cuyo acceso está permitido

2. Carga de trabajo

   Al ser el intermediario de todos los usuarios, el Proxy tiene que realizar el trabajo de todos ellos. La carga de trabajo crece en la medida en que crezcan la cantidad de usuarios que consumen sus servicios.

3. Confianza del Usuario

   Pueden aparecer usuarios que no se sientan cómodos con que todo su tráfico sea interceptado.

4. Modificación en el Software de los usuarios

   La utilización de un Proxy, requiere que las aplicaciones que interactúan con Internet que los usuarios utilizan en la red interna, se configuren (Ver Sección 5.4) para que puedan tener acceso hacia el exterior a través del Proxy.

5. Servicios no disponibles

   Existen algunos aplicaciones que no funcionan con un Proxy de por medio, por ejemplo el servicio de mensajería Whatsapp[6].

6. Retardo

   Que la comunicación del cliente con el servidor final no sea directa, supone un retardo en las comunicaciones.

5.4. Configuración

Existen varias maneras de configurar un browser para navegar a través de un Proxy.

1. Manual: Se configura desde una opción específica del navegador.

2. Flags[7]: Un proxy puede configurarse con un flag específico en el lanzamiento del proceso, esto permite indicar el proxy previo a la ejecución del navegador. En el caso de Google Chrome / Chromium el flag utilizado es:

[6] http://www.whatsapp.com/
[7] Bandera

–proxy-server direccion:puerto

Este flag es el que se va a utilizar más adelante en la parte de experimentación del Proxy desarrollado. El listado de banderas completo se encuentra online[8].

3. Archivos PAC: Puede definirse un archivo de Auto-Configuración de Proxy, es un archivo Javascript en el que se pueden realizar acciones avanzadas como por ejemplo, identificar los protocolos y redireccionar al proxy indicado. Ejemplo de archivo PAC en el que se direcciona según el protocolo (HTTP, HTTPS a un Proxy diferente):

```
function FindProxyForURL(url, host) {
    if (url.substring(0,5) == "http:") {
        return "PROXY proxy.normal.com:8080";
    }else if (url.substring(0,6) =="https:") {
        return "PROXY proxy.seguro.com:8080";
    } else {
        return "DIRECT";
}
```

[8]http://peter.sh/experiments/chromium-command-line-switches/

Capítulo 6

Caché Web

6.1. Definición

Un caché web es un intermediario entre un cliente (o varios) y un Servidor (o varios). Observa las peticiones que se van realizando y va almacenando los recursos solicitados, de manera que, si hay otra petición de la misma URL, el intermediario puede brindar el recurso sin tener que solicitarlo al destino original [27].

El funcionamiento básico de un Caché es el siguiente. Cuando se recibe una petición del cliente, se busca el recurso en el almacenamiento local. En el caso de encontrarlo, se realiza la verificación de que el recurso esté "fresco", es decir, que la copia se encuentre actualizada. Luego, se envía como respuesta al cliente, la copia local. Cuando el recurso se devuelve desde la caché, se llama Caché HIT[1], y, en el caso de que el recurso no se encuentre, se llama Caché MISS[2].

Posee ciertos beneficios al proveer un mecanismo eficiente para la distribución de Información en la Web [39, p. 20].

1. Latencia

   Un Caché Web cercano a los clientes, reduce la latencia para los aciertos de caché[3]. Los retrasos en la transmisión son menores por la cercanía entre los dos

[1] Acierto
[2] Desacierto
[3] Cuando un recurso solicitado por el cliente, efectivamente se encuentra almacenado en la caché.

puntos de interacción. Adicionalmente, se reducen los retrasos por retransmisión y las esperas en los routers a causa de que hay menos enlaces intermedios involucrados.

2. Ancho de Banda

   Cuando se utilizan las copias de los recursos almacenados en la caché, se ahorra ancho de banda, ya que no hay que hacer la petición del recurso al destino final.

3. Carga del Servidor

   Reduce la carga de los servidores al disminuir la cantidad de peticiones que se realizan. Un servidor que esta con poca carga de trabajo es más rápido que uno que está muy ocupado, es decir, atendiendo gran cantidad de peticiones de diferentes clientes.

4. Si un servidor remoto no está disponible temporalmente, una copia del objeto pedido se puede recuperar de la caché.

6.2. Tipos

Hay diferentes tipos de Cachés Web, entre ellos se encuentran los siguientes:

1. Caché de Navegador

   Los navegadores tienen un caché incluido, la información se almacena de manera local en el dispositivo del usuario. Esta limitado a 1 solo usuario, y se obtiene un hit de caché, solamente cuando se visita nuevamente algun sitio que ya se haya visitado.

2. Proxy Cachés

   A diferencia de los de navegador, estos sirven a un número grande de usuarios. Al crecer la cantidad de usuarios, también crece la tasa de aciertos de la caché, ya que los usuarios suelen acceder a los mismos sitios (sitios de gran popularidad) [18].

3. Surrogates Servers

   Son intermediarios que actúan con la autoridad del servidor original para servir contenidos como si fueran el servidor mismo [28]. Generalmente se encuentran cerca de los servidores originales, sirviendo el contenido de los mismos, generalmente desde una caché interna.

   Se usan como Redes de Distribución de Contenidos (CDN), para tener replicaciones de los recursos de un servidor en diferentes lugares. Los recursos peticionados por los clientes, son revueltos por el CDN más cercano fisicamente. También se usan como aceleradores, simplemente almacenando en la caché las respuestas del servidor.

6.3. Politicas de Reemplazo / Algoritmos de Caché

Para el buen funcionamiento de la caché, hay que definir de qué manera se van reemplazando los contenidos almacenados en la misma. Para esto se utiliza una Política de Reemplazo de objetos, orientado a ciertas características de los objetos, tales como el tamaño, cantidad de peticiones de dicho objeto, tiempo almacenado en la caché, etc. Hay diversos algoritmos que se detallan a continuación.

1. Primero en Entrar Primero en Salir (FIFO[4])

   Este método es el más sencillo, se trata de una cola en la que el primer elemento en entrar es el primer elemento en salir, es decir, aquel que más tiempo haya estado en la caché es el que más probabilidades tiene de salir.

2. Aleatorio

   Realiza el reemplazo buscando aleatoriamente un objeto almacenado en la caché.

3. Menos Usado Recientemente (LRU[5])

   Este método, busca reemplazar al objeto que más tiempo haya estado en la caché sin ser peticionado, es decir, aquel objeto que menos peticiones haya

[4]First In First Out
[5]Last Recently Used

tenido en un lapso de tiempo amplio, es el que tiene más posibilidades de ser reemplazado.

4. Menor Menos Usado Recientemente (LRU-MIN)

   Es una optimización del algoritmo LRU, en el cual se busca reemplazar los objetos de mayor tamaño de la caché. quedando así objetos de menor tamaño almacenados.

5. Menos Frecuentemente Usado (LFU[6])

   Se evalúa la frecuencia con la que un elemento es solicitado, el que es menos solicitado es el que tiene más posibilidades de ser reemplazado.

6.4. Control de Caché

Cuando el cliente realiza una petición, y esta coincide con un elemento que tenemos almacenado en la Caché se debe evaluar si es viable enviar la copia local, o hacer la petición nuevamente, a esto se le llama Revalidación. Para esto se realizan ciertos controles (extraídos de [27]) en base a los Headers de HTTP (vistos en 3.4.1).

1. El Header *Expires*, nos indica cuanto tiempo va a estar "fresco"[7] el recurso. Si en el momento de la petición, todavía no expiró el recurso, puedo servirlo de la copia local. Tiene algunas limitaciones, como por ejemplo que el valor permitido es una fecha en GMT, lo que lleva a que los relojes del Servidor y del caché estén sincronizados, de otra manera no tiene sentido una comparación de fechas. Otro problema que se encuentra es que si se envía este header, pero no está actualizado, siempre va a enviar una fecha anterior a la actual lo que llevaría a generar siempre la petición del recurso.

2. Se puede utilizar el Header *Cache-control*[8] que brinda información variada. El formato es: **Cache-control: especificación1, especificación2, especificaciónN**.

[6] Least Frequently Used
[7] No va a tener cambios.
[8] Implementado a partir de la versión 1.1 de HTTP

a) *max-age* - Especifica en tiempo en el que el recurso puede ser considerado "fresco" en segundos. Es el similar a *Expires* pero con un valor específico a diferencia de una marca de tiempo.

b) *s-maxage* - Similar al anterior pero para los cachés compartidos[9].

c) *public* - Indica que las peticiones que requieren autenticación pueden almacenarse en la caché.

d) *private* - Permite que el recurso puede almacenarse en aquellos cachés que son solo para 1 solo usuario[10] y no en una caché compartida.

e) *no-cache* - Fuerza a que el recurso tenga que revalidarse antes de devolver la copia local de la caché al cliente.

f) *no-store* - Indica que el recurso no debe quedar almacenado en la caché bajo ninguna circunstancia.

g) *must-revalidate* - Si el recurso luego de validarlo en la caché no esta "fresco", debe revalidarse.

h) *proxy-revalidate* - Igual que *must-revalidate* pero aplicado a Proxies.

Cuando ambos Headers, *Expires* y *Cache-control* están presentes en la respuesta, *Cache-control* es el que toma mayor importancia.

Para realizar la Revalidación se utilizan ciertos Headers en la petición. Los Servidores utilizan 2 Headers para realizar las validaciones, uno es *Last-Modified*, que indica cuándo fué modificado por última vez el recurso. Cuando éste está presente, el cliente puede realizar una petición con el Header *If-Modified-Since*:fecha, para que el servidor devuelva el recurso unicamente si se cumple con esta condición, de no ser así, devuelve un mensaje con el código `304 Not Modified`, que indica que el recurso no fue modificado y se puede servir la copia local. Por otro lado, se utiliza un identificador llamado *ETag* que se genera cada vez que el recurso se modifica. Es una cadena de texto que identifica una versión específica del recurso. En este caso, el cliente puede utilizar el Header *If-None-Match*:ETag_local, que el servidor responde con el recurso en el caso de que el identificador enviado por el cliente no coincida con el del objeto peticionado.

[9] Por ejemplo un Proxy.
[10] Por ejemplo la caché de un Browser.

Capítulo 7

Proxy Adaptativo para Protocolos Web Avanzados

7.1. Introducción

A lo largo de los capítulos anteriores, hemos visto, el crecimiento de Internet, en el Capítulo 2, las deficiencias de los protocolos, en el Capítulo 3, y la problemática de los tiempos de carga y sus posibles optimizaciones en el Capítulo 4. También se ha visto SPDY como propuesta a resolver varios de los problemas actuales de la carga de los sitios, pero también, en ciertos contextos no funciona como lo esperado (ver Sección 3.6.4). Por todo ello, se desarrolla un Proxy (ver Capítulo 5), cuya funcionalidad principal, es seleccionar de los métodos habilitados que ofrezca un sitio (HTTP HTTPS, SPDY), cuál es el óptimo para recuperar el recurso. Posee otras funcionalidades tales como, ser MITM[1], Caché (ver Capítulo 6), cálculo de RTT, recuperación de los métodos habilitados para un sitio, cliente SPDY.

Para el desarrollo del Proxy, se escogió Python[2] como lenguaje principal, dada la facilidad del mismo y la gran documentación disponible en la Web. Para el almacenamiento de los datos de los sitios se escogió MongoDB[3], por su simple interacción con Python y su velocidad.

[1] Man in The Middle
[2] https://www.python.org/
[3] http://http://www.mongodb.org/

Todo el código del trabajo se encuentra alojado en un repositorio de Github[4], que es un repositorio para alojar proyectos de código abierto. El Proxy se puede navegar desde aquí [26].

7.2. Funcionamiento

El Proxy se inicia en una dirección y en un puerto, por ejemplo: *localhost:8080* se debe configurar el Navegador para que utilice la dirección del proxy para conectarse a Internet. Por ejemplo en Google Chrome se puede iniciar con un Flag para configurar el proxy:

chrome –proxy-server="localhost:8080"

Acepta tanto conexiones HTTP, como HTTPS en el mismo puerto. Al realizar la conexión por SSL con el cliente en vez de actuar de "puente" con el destino final, utiliza un Certificado SSL para realizar la autenticación.

Cuando recibe una petición HTTP, el método que llega al Proxy es *GET*, se consulta la Caché para ver si el recurso se encuentra almacenado para retornarlo desde la copia del Proxy, si no se encuentra, se pide al servidor correspondiente y se devuelve el recurso al cliente. Luego, se realiza el Análisis del Recurso (ver 7.2.1) en segundo plano.

Cuando recibe una petición HTTPS, el método que llega al Proxy es CONNECT, que es un método especial para solicitarle a un Proxy una conexión al servidor final. En este caso, el Proxy responde con el mensaje:

HTTP/1.1 200 Connection established

Para indicarle que se va a realizar la conexión. Esta primera etapa puede verse en la Figura 7.1.

Luego, se inicia la conexión SSL Cliente-Proxy y comienzan a recibirse las peticiones del Cliente, como puede verse en la Figura 7.2, las conexiones[5] se realizan individualmente (no como un Túnel HTTPS normal, ver Figura 5.2). Con cada una de las peticiones, se procede de manera similar a HTTP. Se analiza si el recurso

[4]https://www.github.com
[5]Tanto hacia el Cliente como hacia el Servidor.

Figura 7.1: Conexión inicial cuando el protocolo es HTTPS

se encuentra almacenado en la caché y se devuelve de la copia local, o se solicita el recurso al servidor final. En el caso de que el método seleccionado para devolver el recurso sea HTTS o SPDY, se establece una conexión segura entre el proxy y el servidor final.

Cabe destacar que, todo el tráfico que fluye entre el cliente y el proxy, y entre el proxy y el servidor final, viaja encriptado. Pero, todo el tráfico se encuentra desencriptado en el Proxy, siendo este, un proxy HTTPS de confianza (ver 5.1, discusión en [4]).

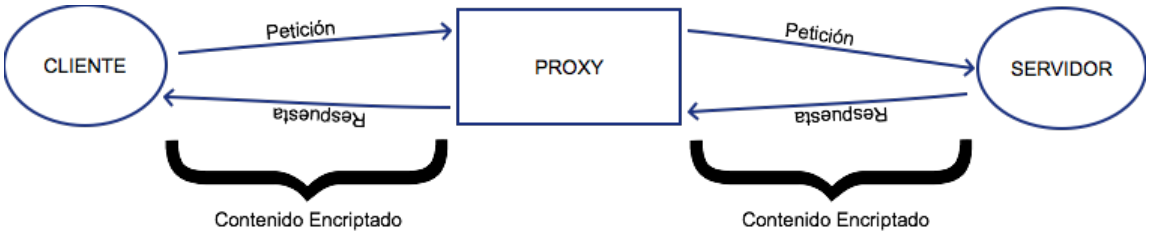

Figura 7.2: Procedimiento una vez realizada la conexión en los 2 extremos

7.2.1. Análisis de los Recursos

Cada vez que el Proxy recibe una petición del cliente, se analiza si el recurso se encuentra en la Caché, en el caso de que el recurso se encuentre almacenado, se devuelve desde la copia local del Proxy y se actualiza la Caché con el HIT (ver Capítulo 6) del elemento correspondiente. Caso contrario, se peticiona el recurso servidor final. Cada vez que el Proxy pide un recurso, se realiza un análisis del mismo en segundo plano:

1. Cálculo del tamaño del recurso.

2. Cálculo del RTT: se realiza el cálculo del RTT al host donde se aloja el recurso.

3. Obtención de Protocolos: Se obtienen los protocolos soportados (HTTP, HTTPS, SPDY) del host donde se aloja el recurso.

4. Cálculo de Peticiones a Generar: en el caso de que el recurso sea un HTML plano, se calcula cuantas peticiones generaría. Se realiza analizando el texto y extrayendo los recursos que esa página va a necesitar para su renderización.

Toda esta información, luego es consumida por el Árbol de Decisión para determinar que protocolo es el más óptimo para recuperar el recurso.

7.2.2. Árbol de Decisión

Una vez que se recibe la petición del cliente, y hay que peticionar el recurso al servidor final, se decide con qué método se debe traer el recurso. Se tomó como referencia, el árbol de "How Speedy is SPDY?" [38], que puede verse en la Figura 7.3.

Este árbol, se retroalimenta con la información que se extrae de los recursos a medida que el Proxy está en funcionamiento. Datos tales como tamaño, RTT, peticiones a generar, se obtienen del almacenamiento propio del Proxy.

7.3. Módulos Adicionales

7.3.1. Cálculo del RTT

Módulo que se encarga de realizar un Ping hacia el host del que se busca obtener la Latencia entre ambos nodos. El Proxy envía un solo paquete ICMP al servidor y se espera su respuesta. Una vez obtenido el resultado, se almacena en la base de datos local. También, ofrece la posibilidad de realizar peticiones masivas, leyendo URLs desde un archivo de texto. Esta opción es útil por ejemplo, para ejecutar antes de iniciar el Proxy, y así obtener mediciones a priori de hosts que serán visitados.

7.3.2. Protocolos Habilitados

Módulo que se encargar de identificar que protocolos soporta un sitio en cuestión. Los protocolos de los que se intenta averiguar si el sitio tiene disponible son HTTP,

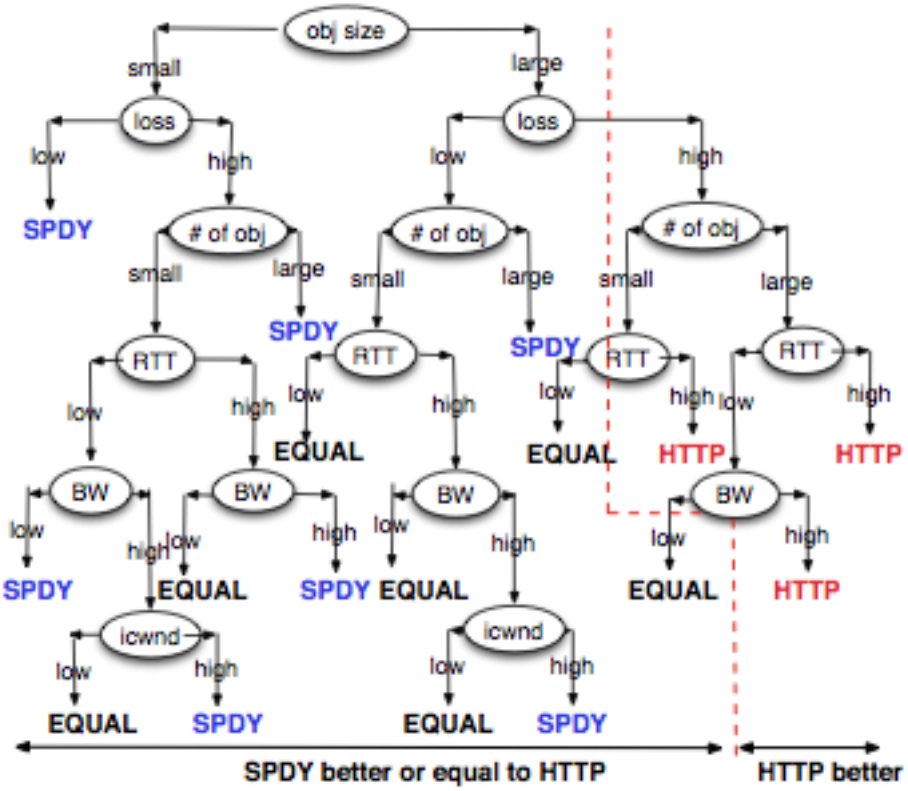

Figura 7.3: Árbol de Decisión, extraído de [38]

HTTPS y SPDY. El procedimiento es el siguiente:

1. HTTP:

   Se intenta realizar una conexión al puerto por defecto de HTTP, el puerto 80. En el caso de tener éxito, se marca HTTP como un protocolo soportado.

2. HTTPS:

   Se intenta realizar una conexión al puerto por defecto de HTTPS, el puerto 443. En el caso de tener éxito, se marca HTTPS como un protocolo soportado.

3. SPDY

   En el caso de SPDY, se utilizó el cliente desarrollado junto con el Proxy (ver 7.3.3). Al igual que con los otros protocolos, se intenta realizar una conexión segura con el servidor final, si la conexión tiene éxito, se intenta iniciar una

Sesión SPDY con el servidor. Si el inicio de la Sesión es correcto, se marca SPDY como un protocolo soportado.

Al igual que el módulo que calcula el RTT (ver 7.3.1), éste posee una opción para realizar la verificación de los protocolos habilitados, leyendo un archivo de texto con las URLs. Tal como el módulo antes descripto, es útil para poseer información previo a la utilización del Proxy.

7.3.3. Cliente SPDY

Python no trae soporte nativo para SPDY, por ello se buscó una libreria que pueda brindar la funcionalidad del protocolo. La seleccionada fue Spdylay [36] de Tatsuhiro Tsujikawa, que, si bien es para el lenguaje C, provee una extensión para utilizarla desde Python[6]. Utilizando esta librería, se diseño un Cliente para poder realizar y mantener una Sesión a travéz de una conexión TCP con un servidor que soporte SPDY. El módulo permite iniciar una conexión con un servidor y luego iniciar la Sesión SPDY. Luego, se pueden realizar peticiones a través de la misma conexión.

[6] http://tatsuhiro-t.github.io/spdylay/python.html

Capítulo 8

Experimentos y Resultados

8.1. Introducción

8.2. Extracción Top Alexa

Script, sitios utilizados (en anexo)

8.3. Experimento

Pseucódigo del experimento. Chrome-har-capturer Porque se usa el onLoad y no el onContentLoad.

8.4. Resultados

Capítulo 9

Conclusiones

Glosario

A

Ancho de Banda

Retraso en la transmisión de datos desde un punto a otro. 36

G

GMT - Greenwich Mean Time

Identificador Universal de Recursos. Es un estándar de tiempo que originalmente se refería al tiempo solar medio en el Real Observatorio de Greenwich, en Greenwich, cerca de Londres, Inglaterra, que en 1884 fue elegido por la Conferencia Internacional del Meridiano como el primer meridiano. 36

H

HTML - HyperText Markup Language

Lenguaje de Marcado de Hipertexto. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado cdigo HTML) para la definición de contenido de una página web.. 36

L

Latencia

Retraso en la transmisión de datos desde un punto a otro. 36

M

MIME - Multipurpose Internet Mail Extensions

Son una serie de convenciones o especificaciones dirigidas al intercambio a través de Internet de todo tipo de archivos (texto, audio, video, etc.) de forma transparente para el usuario. Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos.. 11, 36

N

Netcraft

Es una compañía de servicios de Internet basada en Bath, Inglaterra. Netcraft ofrece análisis de cuota de mercado de servidores y alojamiento web, incluyendo la detección del tipo de servidor web y de sistema operativo.. 36

P

PDF - Portable Document Format

Formato de Documento Portátil. Es un formato de almacenamiento de documentos digitales independiente de plataformas de software o hardware. Este formato es de tipo compuesto (imagen vectorial, mapa de bits y texto). Fue inicialmente desarrollado por la empresa Adobe Systems, oficialmente lanzado como un estándar abierto el 1 de julio de 2008 y publicado por la Organización Internacional de Estandarización como ISO 32000-1.. 36

R

RFC - Request For Comments

Petición de Comentarios. Son una serie de notas sobre Internet, y sobre sistemas que se conectan a internet, que comenzaron a publicarse en 1969. Cada una de ellas individualmente es un documento cuyo contenido es una propuesta oficial para un nuevo protocolo de la red Internet, que se explica con todo detalle para que en caso de ser aceptado pueda ser implementado sin ambiguedades.. 36

U

URI - Universal Resource Identifier

Identificador Universal de Recursos. Es una cadena de caracteres corta que identifica inequívocamente un recurso. Ver [15]. 36

URL - Localizador de Recursos Uniforme

Secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, vdeos, presentaciones digitales, etc. El formato general de un URL es: esquema://máquina/directorio/archivo. 36

Bibliografía

[1] The "data" url scheme. . URL http://tools.ietf.org/html/rfc2397.

[2] Deflate compressed data format specification. . URL http://www.ietf.org/rfc/rfc1951.txt.

[3] El modelo osi. URL http://www.exa.unicen.edu.ar/catedras/comdat1/material/ElmodeloOSI.pdf.

[4] Explicit trusted proxy in http/2.0. URL http://www.ietf.org/id/draft-loreto-httpbis-trusted-proxy20-01.txt.

[5] February 2014 web server survey. URL http://news.netcraft.com/archives/category/web-server-survey/.

[6] Gzip file format specification. . URL http://www.gzip.org/zlib/rfc-gzip.html.

[7] Http over tls. . URL http://tools.ietf.org/html/rfc2818.

[8] httparchive - the http archive tracks how the web is built. URL http://httparchive.org/.

[9] Internet growth statistics. URL http://www.internetworldstats.com/emarketing.htm.

[10] La encapsulación de datos, un concepto crítico. URL http://www.redescisco.net/v2/art/la-encapsulacion-de-datos-un-concepto-critico/.

[11] Rfc: Hypertext transfer protocol – http/1.1. . URL http://www.ietf.org/rfc/rfc2616.txt.

[12] Spdy: An experimental protocol for a faster web. . URL http://www.chromium.org/spdy/spdy-whitepaper.

[13] Spdy protocol - draft 1. . URL http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1.

[14] Stream control transmission protocol. . URL http://tools.ietf.org/html/rfc2960.

[15] Uniform resource identifier (uri): Generic syntax. . URL http://tools.ietf.org/html/rfc3986.

[16] Web caching: Making the most of your internet connection. URL http://www.web-cache.com.

[17] Mike Belshe. More bandwidth doesn't matter (much). Agosto 2010.

[18] Duska, Bradley, David Marwood, y Michael Feely. The measured access characteristics of world-wide-web client proxy caches. Diciembre 1997.

[19] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, y K.K. Ramakrishnan. Towards a spdy?ier mobile web? Diciembre 2013.

[20] Bryan Ford. Structured streams: a new transport abstraction. Agosto 2007.

[21] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, y Sailu Reddy. *HTTP: The Definitive Guide*. O'Reilly Media, 2002.

[22] Ilya Grigorik. Web-vpn: Secure proxies with spdy & chrome. URL https://www.igvita.com/2011/12/01/web-vpn-secure-proxies-with-spdy-chrome/.

[23] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly Media, 2013.

[24] ISO/IEC. Information technology - open systems interconnection - basic reference model: The basic model. Noviembre 1994.

[25] Balachander Krishnamurthy, Jeffrey C. Mogul, y David M. Kristol. Key differences between http/1.0 and http/1.1. 1999.

[26] Pablo Maximiliano Lulic. Proxy adaptativo para protocolos web avanzados. URL https://www.github.com/maxisoad/spdyproxypython.

[27] Mark Nottingham. Caching tutorial for web authors and webmasters. URL http://www.mnot.net/cache_docs/.

[28] Mark Nottingham y Xiang Liu. Edge architecture specification. URL http://www.w3.org/TR/edge-arch.

[29] Website Optimization. Average web page size triples since 2008. URL http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[30] International Standard Organization. The measured access characteristics of world-wide-web client proxy caches. Noviembre 1994.

[31] Jitendra Padhye y Henrik Frystyk Nielsen. A comparison of spdy and http performance. Julio 2012.

[32] Carlos E. Quesada Sánchez y Esteban Meneses. Polticas de reemplazo en la caché de web. Diciembre 2006.

[33] Steve Souders. *High Performance Web Sites*. O'Reilly Media, 2007.

[34] W. Richard Stevens. *TCP/IP Illustrated, Vol. 1: The Protocols*. Addison-Wesley, 1993.

[35] Terry Sullivan. How much is too much? URL http://www.pantos.org/atw/35654.html.

[36] Tatsuhiro Tsujikawa. Spdylay - spdy c library. URL http://tatsuhiro-t.github.io/spdylay/.

[37] W3C. Http - hypertext transfer protocol. URL http://www.w3.org/Protocols/.

[38] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, y David Wetherall. How speedy is spdy? Abril 2014.

[39] Duane Wessels. *Web Caching*. O'Reilly Media, 2001.

[40] Hubert Zimmermann. Osi reference model - the iso model of arquitecture for open systems interconnection. Abril 1980.