



لینکلر
پیشگیری
میانبر





HTTP The Definitive Guide

آشنایی با پروتکل HTTP

تهیه شده توسط تیم تولید محتوای وب سایت دنیای امنیت

احسان نیک آور





فهرست مطالب

۱۲ مقدمه ..
۱۳ Overview of HTTP – فصل اول
۱۴ HTTP: The Internet's Multimedia Courier
۱۵ Web Clients and Servers
۱۶ Resources
۱۷ Transactions
۱۸ Messages
۱۹ Connections
۲۰ Protocol Versions
۲۱ Architectural Components of the Web
۲۲ For More Information
۲۳ URLs and Resources – فصل دوم
۲۴ Navigating the Internet's Resources
۲۵ URL Syntax
۲۶ URL Shortcuts
۲۷ Shady Characters
۲۸ A Sea of Schemes
۲۹ The Future
۳۰ For More Information
۳۱ HTTP Messages – فصل سوم
۳۲ The Flow of Messages
۳۳ The Parts of a Message





۷۱	Methods
۷۸	Status Codes
۸۷	Headers
۹۶	For More Information
۹۷	فصل چهارم – Connection Management
۹۷	TCP Connections
۱۰۳	TCP Performance Considerations
۱۱۰	HTTP Connection Handling
۱۱۳	Parallel Connections
۱۱۶	Persistent Connections
۱۲۷	Pipelined Connections
۱۲۸	The Mysteries of Connection Close
۱۳۳	For More Information
۱۳۴	فصل پنجم – Web Servers
۱۳۴	Web Servers Come in All Shapes and Sizes
۱۳۷	A Minimal Perl Web Server
۱۴۰	What Real Web Servers Do
۱۴۱	Step 1: Accepting Client Connections
۱۴۳	Step 2: Receiving Request Messages
۱۴۶	Step 3: Processing Requests
۱۴۹	Step 4: Mapping and Accessing Resources
۱۵۳	Step 5: Building Responses
۱۵۶	Step 6: Sending Responses





۱۵۶	Step 7: Logging
۱۵۷	For More Information
۱۵۸	فصل ششم - Proxies
۱۵۹	Web Intermediaries
۱۶۰	Why Use Proxies?
۱۶۱	Where Do Proxies Go?
۱۷۴	Client Proxy Settings
۱۷۷	Tricky Things About Proxy Requests
۱۸۶	Tracing Messages
۱۹۳	Proxy Authentication
۱۹۴	Proxy Interoperation
۱۹۷	For More Information
۱۹۸	فصل هفتم - Caching
۱۹۸	Redundant Data Transfers
۱۹۸	Bandwidth Bottlenecks
۲۰۰	Flash Crowds
۲۰۱	Distance Delays
۲۰۲	Hits and Misses
۲۰۶	Cache Topologies
۲۱۰	Cache Processing Steps
۲۱۴	Keeping Copies Fresh
۲۲۲	Controlling Cachability
۲۲۷	Setting Cache Controls





۲۳۰	Detailed Algorithms
۲۳۸	Caches and Advertising
۲۴۰	For More Information
۲۴۱	فصل هشتم – Integration Points: Gateways, Tunnels, and Relays
۲۴۱	Gateways
۲۴۴	Protocol Gateways
۲۴۸	Resource Gateways
۲۵۱	Application Interfaces and Web Services
۲۵۱	Tunnels
۲۵۹	Relays
۲۶۱	For More Information
۲۶۲	فصل نهم – Web Robots
۲۶۲	Crawlers and Crawling
۲۷۳	Robotic HTTP
۲۷۷	Misbehaving Robots
۲۷۸	Excluding Robots
۲۹۱	Robot Etiquette
۲۹۴	Search Engines
۲۹۹	For More Information
۳۰۰	فصل دهم – HTTP-NG
۳۰۰	HTTP's Growing Pains
۳۰۱	HTTP-NG Activity
۳۰۱	Modularize and Enhance





۳۰۲	Distributed Objects
۳۰۳	Layer 1: Messaging
۳۰۴	Layer 2: Remote Invocation
۳۰۵	Layer 3: Web Application
۳۰۶	WebMUX
۳۰۷	Binary Wire Protocol
۳۰۸	Current Status
۳۰۹	For More Information
۳۱۰	فصل یازدهم – Client Identification and Cookies
۳۱۱	The Personal Touch
۳۱۲	HTTP Headers
۳۱۳	Client IP Address
۳۱۴	User Login
۳۱۵	Fat URLs
۳۱۶	Cookies
۳۱۷	For More Information
۳۱۸	فصل دوازدهم – Basic Authentication
۳۱۹	Authentication
۳۲۰	Basic Authentication
۳۲۱	The Security Flaws of Basic Authentication
۳۲۲	For More Information
۳۲۳	فصل سیزدهم – Digest Authentication
۳۲۴	The Improvements of Digest Authentication





۳۵۲	Digest Calculations
۳۶۰	Quality of Protection Enhancements
۳۶۲	Practical Considerations
۳۶۵	Security Considerations
۳۶۹	For More Information
۳۷۰	فصل چهاردهم – Secure HTTP
۳۷۱	Making HTTP Safe
۳۷۲	Digital Cryptography
۳۷۶	Symmetric-Key Cryptography
۳۷۹	Public-Key Cryptography
۳۸۱	Digital Signatures
۳۸۳	Digital Certificates
۳۸۷	HTTPS: The Details
۳۹۵	A Real HTTPS Client
۴۰۴	Tunneling Secure Traffic Through Proxies
۴۰۶	For More Information
۴۰۷	فصل پانزدهم – Entities and Encodings
۴۰۸	Messages Are Crates, Entities Are Cargo
۴۱۱	Content-Length: The Entity's Size
۴۱۴	Entity Digests
۴۱۵	Media Type and Charset
۴۱۹	Content Encoding
۴۲۳	Transfer Encoding and Chunked Encoding





۴۲۸	Time-Varying Instances
۴۳۴	Range Requests
۴۳۶	Delta Encoding
۴۴۱	For More Information
۴۴۲	فصل شانزدهم - Internationalization – Internationalization
۴۴۲	HTTP Support for International Content
۴۴۳	Character Sets and HTTP
۴۴۹	Multilingual Character Encoding Primer
۴۵۸	Language Tags and HTTP
۴۶۰	Internationalized URIs
۴۶۸	Other Considerations
۴۶۹	For More Information
۴۷۰	Content-Negotiation Techniques
۴۷۱	Client-Driven Negotiation
۴۷۲	Server-Driven Negotiation
۴۷۶	Transparent Negotiation
۴۸۱	Transcoding
۴۸۴	Next Steps
۴۸۴	For More Information
۴۸۶	فصل هجدهم - Web Hosting – Web Hosting
۴۸۶	Hosting Services
۴۸۸	Virtual Hosting
۴۹۷	Making Web Sites Reliable





۵۰۱	Making Web Sites Fast
۵۰۱	For More Information
۵۰۲	فصل نوزدهم – Publishing Systems
۵۰۲	FrontPage Server Extensions for Publishing Support
۵۰۸	WebDAV and Collaborative Authoring
۵۳۲	For More Information
۵۳۳	فصل بیستم – Redirection and Load Balancing
۵۳۳	Why Redirect
۵۳۴	Where to Redirect
۵۳۵	Overview of Redirection Protocols
۵۳۷	General Redirection Methods
۵۵۰	Proxy Redirection Methods
۵۶۰	Cache Redirection Methods
۵۶۴	Internet Cache Protocol
۵۶۶	Cache Array Routing Protocol
۵۷۰	Hyper Text Caching Protocol
۵۷۶	For More Information
۵۷۷	فصل بیست و یکم – Logging and Usage Tracking
۵۷۷	What to Log
۵۷۸	Log Formats
۵۸۸	Hit Metering
۵۹۰	A Word on Privacy
۵۹۱	For More Information







مقدمه

با توجه به استفاده روز افزون از وب و تکنولوژی‌های مربوط با آن، آشنایی با پروتکل HTTP که پایه و اساس ساختار وب را تشکیل می‌دهد، بسیار حائز اهمیت است.

این کتاب که ترجمه کتاب HTTP The Definitive Guide است، به معرفی بخش‌های مختلف این پروتکل می‌پردازد.

دانستن جزئیات مربوط به پروتکل HTTP برای علاقمندان به حوزه‌های وب، تست نفوذ وب، باگ بانتی، برنامه نویسان، محققین و دانشجویان فناوری اطلاعات بسیار مفید خواهد بود.

قطعاً اولین توصیه در هر شاخه از فناوری اطلاعات، مطالعه منبع اصلی است. ولی با توجه به اینکه مطالعه منابع اصلی برای برخی از دوستان دشوار بوده و حتی برداشتی از ترجمه می‌تواند جهت درک بهتر منبع اصلی تاثیر گذار باشد، تصمیم به ترجمه این کتاب گرفته شد.

هیچ کاری بدون نقص نبوده و امیدوارم کمی و کاستی‌های موجود در این کتاب را بر ما ببخشید و مطالب موجود در این کتاب برای شما مفید واقع شود.





فصل اول - Overview of HTTP

مروگرهای وب، سرورها و برنامه‌های کاربردی وب مرتبط، همگی از طریق HTTP یا Hypertext Transfer Protocol با یکدیگر صحبت می‌کنند. HTTP زبان رایج اینترنت جهانی مدرن است.

این فصل مروی مختصر بر HTTP است. خواهید دید که چگونه برنامه‌های کاربردی وب از HTTP برای برقراری ارتباط استفاده می‌کنند و ایده‌ای تقریبی از نحوه انجام کار HTTP دریافت خواهید کرد. به طور خاص، ما در مورد:

- کلاینت‌ها و سرورهای وب چگونه ارتباط برقرار می‌کنند.
- منابع (محتوای وب) از کجا می‌آیند.
- تراکنش‌های وب چگونه کار می‌کنند.
- قالب پیام‌های مورد استفاده برای ارتباط HTTP
- نحوه انتقال شبکه در ساختار TCP به شکل زیربنایی HTTP
- تغییرات مختلف پروتکل HTTP
- برخی از اجزای معمازی HTTP نصب شده در سراسر اینترنت

HTTP: The Internet's Multimedia Courier

میلیاردها تصویر JPEG، صفحات HTML، فایل‌های متنی، فیلم‌های MPEG، فایل‌های صوتی WAV، اپلتهای جاوا و موارد دیگر هر روز از طریق اینترنت در حال گردش هستند. HTTP بخش عمده‌ای از این اطلاعات را به سرعت، راحت و قابل اعتماد از سرورهای وب در سراسر جهان به مروگرهای وب روی دسکتاپ افراد منتقل می‌کند.

از آنجایی که HTTP از پروتکل‌های انتقال داده قابل اعتماد استفاده می‌کند، تضمین می‌کند که داده‌های شما در حین حمل و نقل آسیب نخواهند دید، حتی زمانی که از آن طرف کره زمین می‌آیند. این برای شما به عنوان یک کاربر، خوب است زیرا می‌توانید بدون نگرانی در مورد یکپارچگی اطلاعات به آن دسترسی داشته باشید.

انتقال قابل اعتماد همچنین برای شما به عنوان یک توسعه‌دهنده برنامه‌های اینترنتی خوب است، زیرا لازم نیست نگران از بین رفتن، کپی شدن یا تحریف ارتباطات HTTP در حین انتقال باشید. شما می‌توانید بدون نگرانی در مورد ایرادات و معایب اینترنت، روی برنامه نویسی جزئیات متمایز برنامه خود تمکن کنید.

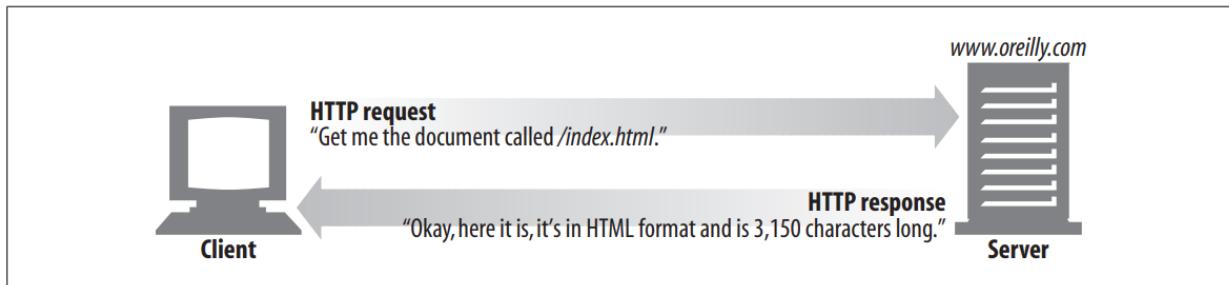
Web Clients and Servers

محتوای وب روی سرورهای وب قرار دارند. وب سرورها با پروتکل HTTP صحبت می‌کنند، بنابراین اغلب، آن‌ها را HTTP Server می‌نامند. این سرورهای HTTP داده‌های اینترنت را ذخیره می‌کنند و





در صورت درخواست مشتریان HTTP، داده‌ها را ارائه می‌دهند. کلاینت‌ها درخواست‌های HTTP را به سرورها ارسال می‌کنند و سرورها داده‌های درخواستی را در پاسخ‌های HTTP، همانطور که در شکل زیر ترسیم شده است، برمی‌گردانند. کلاینت‌های HTTP با هم اجزای اساسی شبکه جهانی وب را تشکیل می‌دهند.



شما احتمالاً هر روز از کلاینت‌های HTTP استفاده می‌کنید. رایج ترین سرویس گیرنده یک مرورگر وب است، مانند Firefox یا Microsoft Internet Explorer. مرورگرهای وب اشیاء HTTP را از سرورها درخواست می‌کنند و اشیاء را روی مانیتور شما نمایش می‌دهند.

هنگامی که صفحه ای مانند "http://www.oreilly.com/index.html" را مرور می‌کنید، مرورگر شما یک درخواست HTTP را به سرور www.oreilly.com ارسال می‌کند (شکل بالا). سرور سعی می‌کند شی مورد نظر (در این مورد، "index.html") را پیدا کند و در صورت موفقیت آمیز بودن، شیء را در یک پاسخ HTTP به همراه نوع شی، طول شی و اطلاعات دیگر برای مشتری ارسال می‌کند.

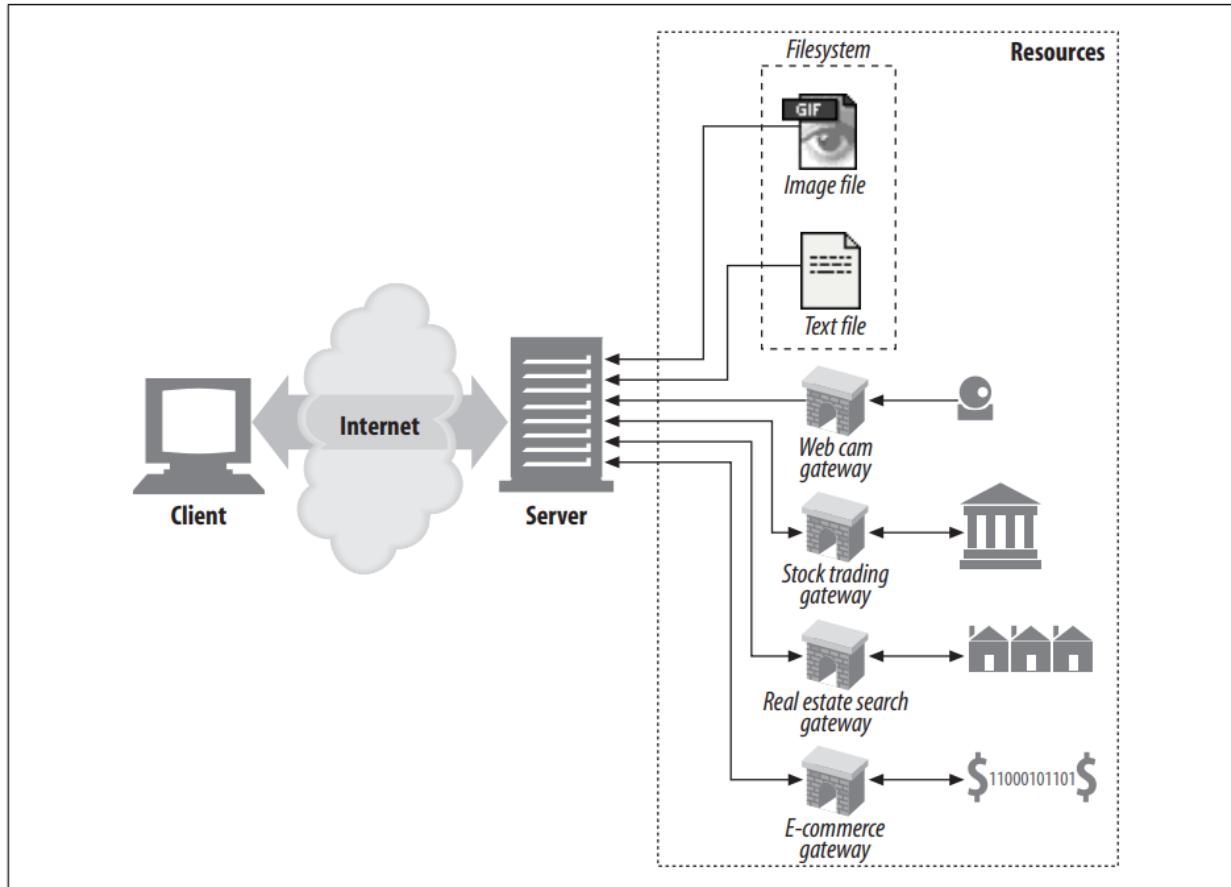
Resources

وب سرورها میزبان منابع وب هستند. ساده‌ترین نوع منبع وب یک فایل استاتیک در سیستم فایل وب سرور است. این فایل‌ها می‌توانند حاوی هر چیزی باشند: ممکن است فایل‌های متنی، فایل‌های HTML، فایل‌های Microsoft Word، فایل‌های تصویری JPEG، Adobe Acrobat، فایل‌های فیلم AVI یا هر فرمت دیگری که فکرش را بکنید.

با این حال، منابع لزوماً نباید فایل‌های ثابت باشند. منابع همچنین می‌توانند برنامه‌های نرم‌افزاری باشند که بر اساس تقاضا، محتوا تولید می‌کنند. این منابع محتوای پویا می‌توانند محتوا را بر اساس هویت شما، اطلاعاتی که درخواست کرده اید یا در زمان و شرایط خاص تولید کنند.



آن‌ها می‌توانند یک تصویر زنده از یک دوربین را به شما نشان دهند، به شما اجازه معامله سهام بدهند، امکان جست و جو در پایگاه داده‌های املاک را برای شما فراهم کنند یا خرید از فروشگاه‌های آنلاین را برای شما به ارمغان آورند.



یک فایل حاوی صفحه گسترده پیش بینی فروش شرکت شما، یک منبع یا Resource است. موتور جستجوی اینترنتی یک منبع است.

Media Types

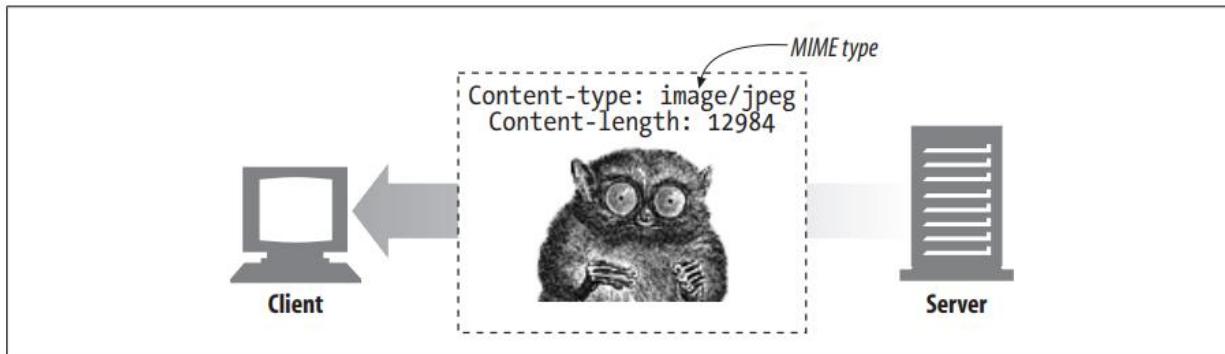
از آنجایی که اینترنت هزاران نوع داده مختلف را میزبانی می‌کند، HTTP هر شی که از طریق وب منتقل می‌شود را با یک برچسب فرمت داده به نام Multipurpose (MIME) Type به دقت برچسب گذاری می‌کند. آنقدر برای ایمیل خوب کار می‌کرد که HTTP آن الکترونیکی پیش می‌آیند طراحی شده بود. Internet Mail Extensions (Internet Mail Extensions) در ابتدا برای حل مشکلاتی که در انتقال پیام‌ها بین سیستم‌های مختلف پست

را برای توصیف و برچسب گذاری محتوای چندرسانه‌ای خود استفاده کرد.





سرورهای وب یک MIME Type را به داده‌های شی HTTP متصل می‌کنند. هنگامی که یک مرورگر وب یک شی را از سرور دریافت می‌کند، به MIME Type مرتبط نگاه می‌کند تا ببیند آیا می‌داند چگونه آن شی را مدیریت کند یا خیر. اکثر مرورگرها می‌توانند صدها نوع شی محبوب را مدیریت کنند: نمایش فایل‌های تصویری، تجزیه (Parsing) و قالب‌بندی (Formatting) فایل‌های HTML، پخش فایل‌های صوتی از طریق بلندگوهای رایانه، یا راهاندازی افزونه نرمافزار خارجی برای مدیریت فرمات‌های خاص.



یک برچسب متنی است که به عنوان یک نوع شی اصلی و یک نوع فرعی خاص نشان داده می‌شود که با یک اسلش از هم جدا شده‌اند. به عنوان مثال:

یک سند متنی با فرمت HTML با نوع text/html برچسب گذاری می‌شود.

یک سند متنی ASCII ساده با نوع text/plain برچسب گذاری می‌شود.

یک نسخه JPEG از یک تصویر، image/jpeg خواهد بود.

یک تصویر با فرمت GIF به صورت image/gif خواهد بود.

یک فیلم مبتنی بر Apple QuickTime به صورت video/quicktime خواهد بود.

یک فایل مربوط به مایکروسافت پاورپوینت، به صورت application/vnd.ms-powerpoint خواهد بود.

URIs

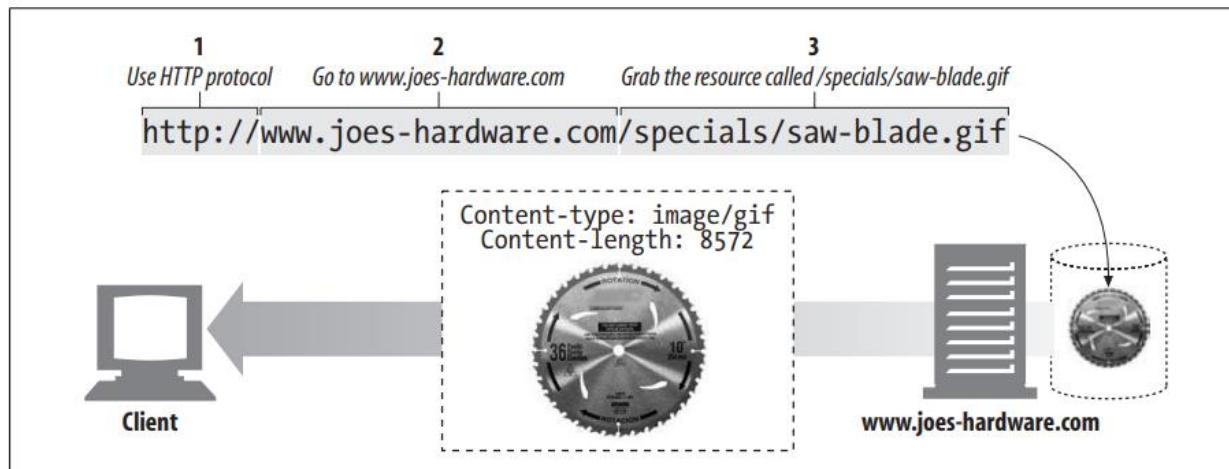
هر منبع یا resource وب سرور یک نام دارد، بنابراین کلاینت‌ها می‌توانند به منابع مورد علاقه خود اشاره کنند. نام این منبع یا resource سرور Uniform Resource Identifier یا URI نامیده می‌شود. URI ها مانند آدرس‌های پستی اینترنت هستند که به طور منحصر به فرد منابع اطلاعاتی را در سراسر جهان شناسایی و مکان‌یابی می‌کنند.

در اینجا یک URI برای یک منبع تصویر در سرور وب فروشگاه سخت افزار Joe آمده است:



:
<http://www.joes-hardware.com/specials/saw-blade.gif>

شکل زیر نشان می‌دهد که چگونه URI HTTP پروتکل GIF را برای دسترسی به منبع در سرور فروشگاه Joe مشخص می‌کند. با توجه به URI، HTTP می‌تواند شی را بازیابی کند. URIها در دو نوع URL و URN ظاهر می‌شوند.



باید اکنون نگاهی به هر یک از این نوع شناسه‌های منابع بیندازیم.

URLs

URL Uniform Resource Locator یا URL رایج ترین شکل شناسه منبع است. URL ها مکان خاصی از یک منبع را در یک سرور خاص توصیف می‌کنند. آن‌ها دقیقاً به شما می‌گویند که چگونه یک منبع را از یک مکان دقیق و ثابت دریافت کنید. شکل بالا نشان می‌دهد که چگونه یک URL دقیقاً محل قرارگیری یک منبع و نحوه دسترسی به آن را نشان می‌دهد.

همچنین در جدول زیر شما می‌توانید چند نمونه از URL ها را مشاهده نمایید:





URL	Description
http://www.oreilly.com/index.html	The home URL for O'Reilly & Associates, Inc.
http://www.yahoo.com/images/logo.gif	The URL for the Yahoo! web site's logo
http://www.joes-hardware.com/inventory-check.cgi?item=12731	The URL for a program that checks if inventory item #12731 is in stock
ftp://joe:tools4u@ftp.joes-hardware.com/locking-pliers.gif	The URL for the <i>locking-pliers.gif</i> image file, using password-protected FTP as the access protocol

اکثر URL ها از یک قالب استاندارد شده که از سه بخش اصلی تشکیل شده است، پیروی می‌کنند:

- بخش اول URL اصطلاحا Scheme نامیده می‌شود و پروتکل مورد استفاده برای دسترسی به منبع را توصیف می‌کند که این پروتکل معمولاً (http://) است.
- بخش دوم آدرس اینترنتی سرور می‌باشد (به عنوان مثال www.joes-hardware.com).
- بخش نهایی شامل یک منبع در وب سرور بوده که نام مشخصی دارد (به عنوان مثال، [saw-blade.gif](http://www.joes-hardware.com/specials/saw-blade.gif)).

امروزه تقریباً هر URI یک URL است.

URNs

نوع دوم از URI، یک URN یا Uniform Resource Name می‌باشد. یک URN به عنوان یک نام منحصر به فرد برای یک قطعه خاص از محتوا، مستقل از جایی که منبع در حال حاضر در آن قرار دارد، عمل می‌کند. این URN های مستقل از مکان، به منابع اجازه می‌دهند از مکانی به مکان دیگر حرکت کنند. URN ها همچنین امکان دسترسی به منابع را با چندین پروتکل دسترسی به شبکه و در عین حال حفظ همان نام فراهم می‌کنند.

برای مثال، URN زیر ممکن است برای نامگذاری سند استاندارد اینترنت "RFC 2141" صرف نظر از محل اقامت آن استفاده شود (حتی ممکن است در چندین مکان کپی شود):

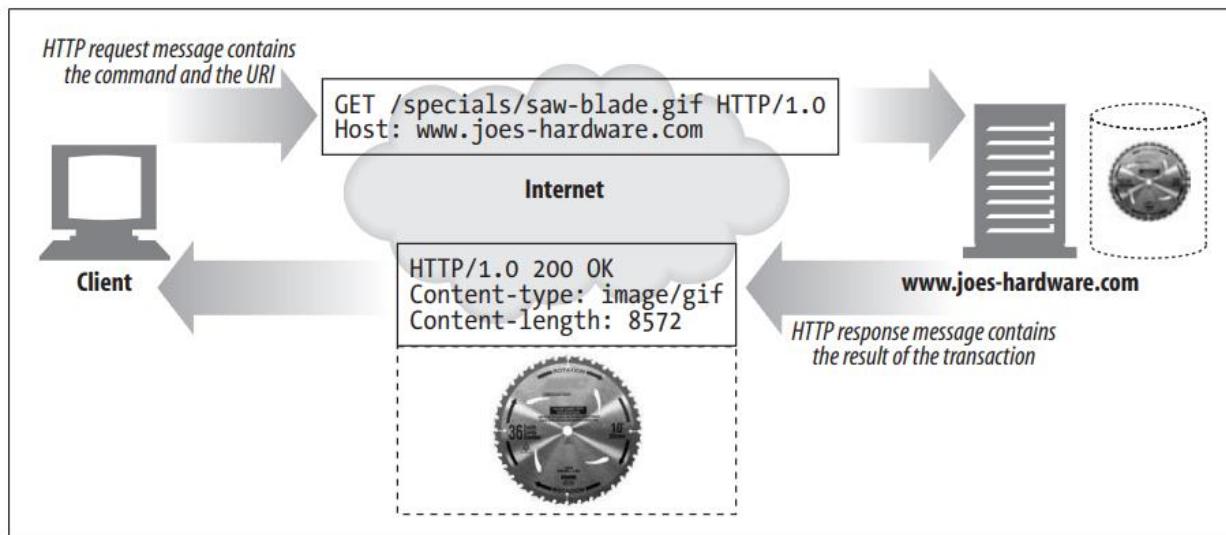
`urn:ietf:rfc:2141`

URN ها هنوز آزمایشی هستند و هنوز به طور گسترده مورد استفاده قرار نگرفته اند. برای کار موثر، URN ها نیاز به یک زیرساخت پشتیبانی برای حل مکان‌های منبع دارند. فقدان چنین زیرساختی نیز پذیرش آن‌ها را کند کرده است. اما URN ها نویدهای هیجان انگیزی برای آینده دارند.



Transactions

بیایید با جزئیات بیشتری به چگونگی استفاده کلانیت‌ها از HTTP برای تراکنش با سرورهای وب و منابع آن‌ها نگاه کنیم. تراکنش HTTP شامل یک فرمان درخواست (ارسال از کلاینت به سرور) و یک نتیجه پاسخ (ارسال از سرور به کلاینت) است. همانطور که در شکل زیر نشان داده شده است، این ارتباط با بلوک‌های داده‌ای به نام HTTP Messages انجام می‌شود.



Methods

HTTP از چندین دستور درخواست مختلف پشتیبانی می‌کند که HTTP Method نامیده می‌شوند. هر پیام درخواست HTTP یک Method دارد. این Method به سرور می‌گوید که چه عملی را انجام دهد (واکشی یک صفحه وب، اجرای یک فایل و غیره). جدول زیر پنج Method متداول HTTP را نشان می‌دهد.

HTTP method	Description
GET	Send named resource from the server to the client.
PUT	Store data from client into a named server resource.
DELETE	Delete the named resource from a server.
POST	Send client data into a server gateway application.
HEAD	Send just the HTTP headers from the response for the named resource.





Status Codes

هر پاسخ HTTP یک کد وضعیت را به همراه دارد. کد وضعیت یک عدد سه رقمی است که به کلاینت می‌گوید که آیا درخواست موفقیت آمیز بوده یا اقدامات دیگری لازم است. چند کد وضعیت رایج در جدول زیر نشان داده شده است.

HTTP status code	Description
200	OK. Document returned correctly.
302	Redirect. Go someplace else to get the resource.
404	Not Found. Can't find this resource.

HTTP همچنین یک "reason phrase" متنی توضیحی را با هر کد وضعیت عددی ارسال می‌کند. عبارت متنی فقط برای اهداف توصیفی گنجانده شده است.

کدهای وضعیت و reason phrase زیر توسط HTTP یکسان برخورд می‌شوند:

200 OK

200 Document attached

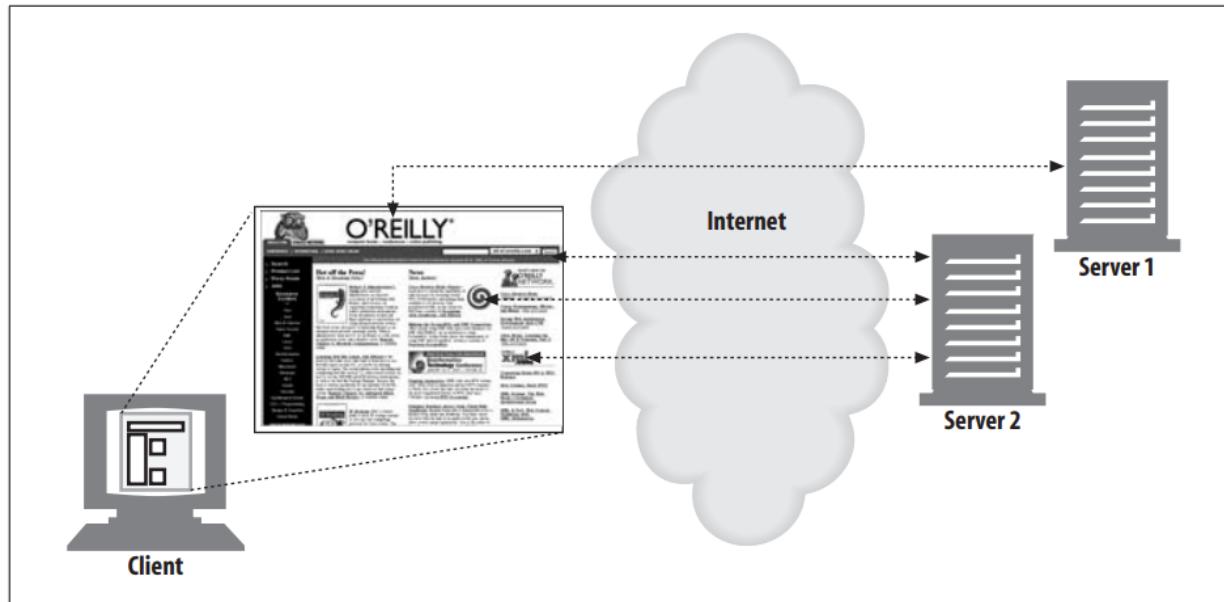
200 Success

200 All's cool, dude

Web Pages Can Consist of Multiple Objects

یک برنامه اغلب چندین تراکنش HTTP را برای انجام یک کار صادر می‌کند. به عنوان مثال، یک مرورگر وب، تعداد زیادی از تراکنش‌های HTTP را برای واکشی و نمایش یک صفحه وب غنی از گرافیک منتشر می‌کند. مرورگر، یک تراکنش را برای واکشی اسکلت HTML انجام می‌دهد که طرح‌بندی صفحه را توصیف می‌کند، سپس تراکنش‌های HTTP اضافی را برای هر تصویر تعبیه شده، صفحه گرافیکی، اپلت‌جاوا و غیره صادر می‌کند. همانطور که در شکل زیر نشان داده شده است این منابع جاسازی شده حتی ممکن است در سرورهای مختلف قرار داشته باشند. بنابراین، یک "صفحه وب" اغلب مجموعه‌ای از منابع است، نه یک منبع واحد.

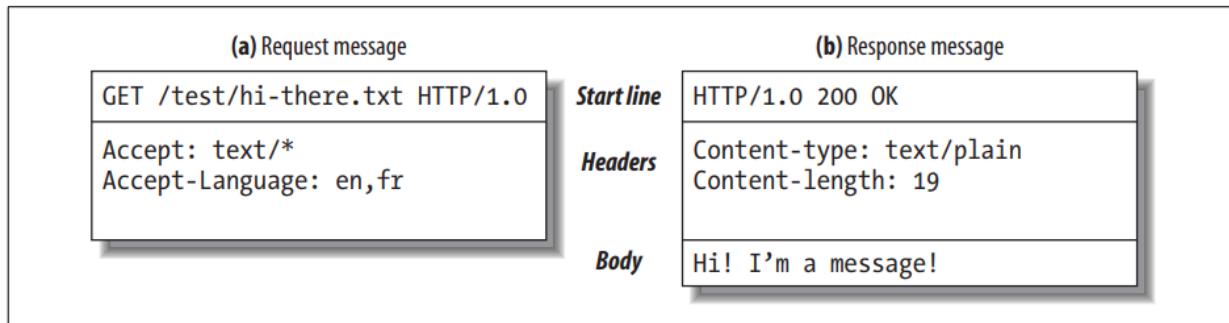




Messages

اکنون اجازه دهید نگاهی گذرا به ساختار پیام‌های درخواست و پاسخ HTTP بیندازیم. ما پیام‌های HTTP را در فصل‌های آتی با جزئیات دقیق مطالعه خواهیم کرد.

پیام‌های HTTP توالی ساده و خطی از کاراکترها هستند. از آنجا که آن‌ها متن ساده هستند، نه باینری، خواندن و نوشتمن برای انسان آسان است. شکل زیر پیام‌های HTTP را برای یک تراکنش ساده نشان می‌دهد.



پیام‌های HTTP که از وب کلاینت به سرورهای وب ارسال می‌شوند، پیام‌های درخواست یا Request نامیده می‌شوند. پیام‌هایی که از سرورها به کلاینت ارسال می‌شود، پیام‌های پاسخ یا Response نامیده می‌شوند. هیچ نوع دیگری از پیام‌های HTTP وجود ندارد. فرمتهای پیام‌های درخواست و پاسخ HTTP بسیار مشابه هستند.





پیام‌های HTTP از سه بخش تشکیل شده است:

Start Line

خط اول پیام، خط شروع است که نشان می‌دهد برای یک درخواست چه کاری باید انجام داد یا برای پاسخ چه اتفاقی افتاد.

Header Fields

صفر یا چند فیلد Header از خط شروع پیروی می‌کند. هر فیلد Header شامل یک نام و یک مقدار است که برای تجزیه آسان با یک دونقطه (:) از هم جدا شده اند. Header ها با یک خط خالی به پایان می‌رسند. افزودن فیلد Header به آسانی افزودن یک خط دیگر است.

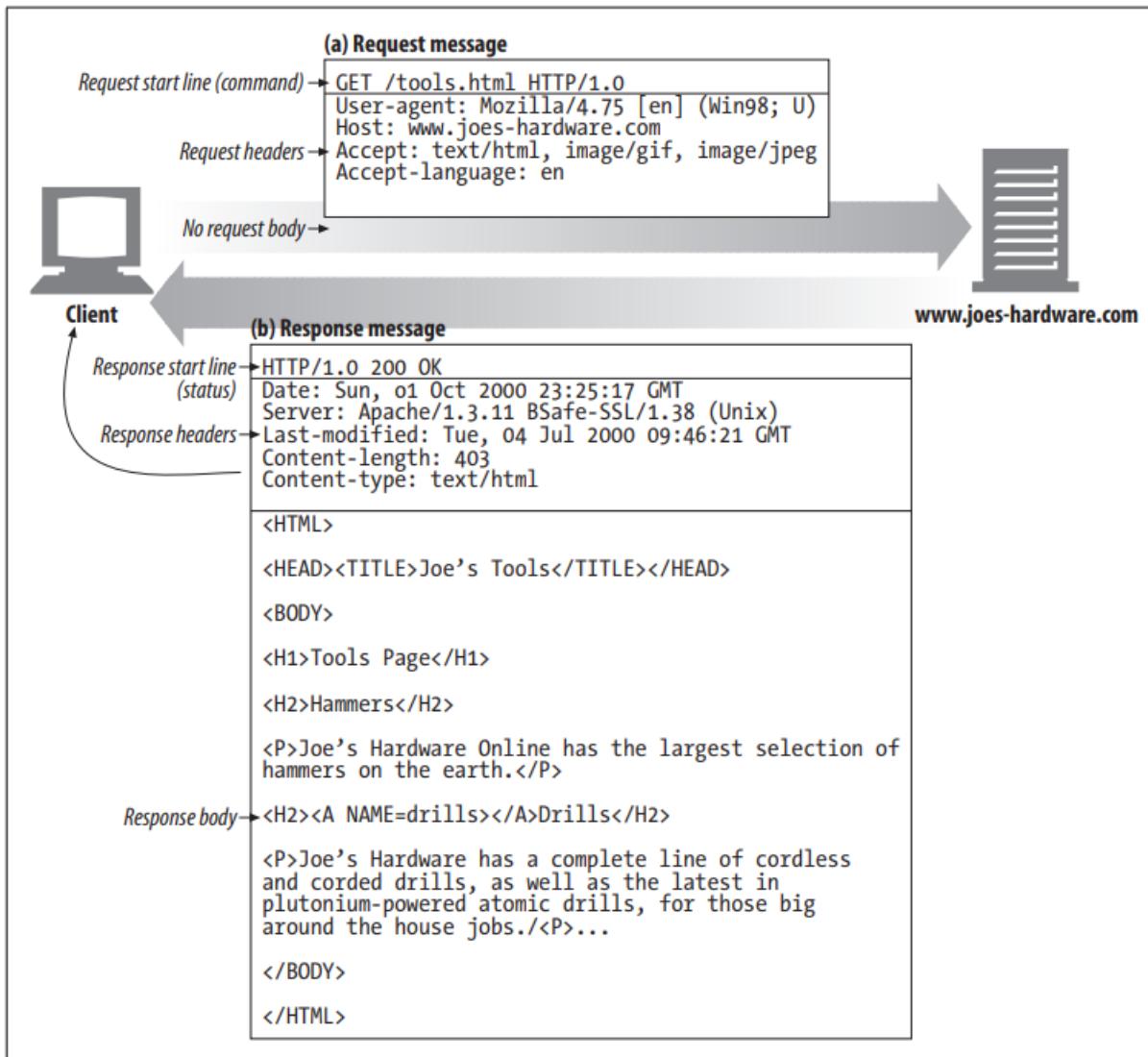
Body

بعد از خط خالی یک متن پیام اختیاری است که حاوی هر نوع داده‌ای است. Body های درخواست، داده‌ها را به سرور وب منتقل می‌کنند. Body های پاسخ، داده‌ها را به کلاینت منتقل می‌کنند. برخلاف Start Line و Header ها که متنی و ساختاری هستند، Body می‌تواند حاوی داده‌های باینری دلخواه باشد (مانند تصاویر، فیلم‌ها، آهنگ‌های صوتی، برنامه‌های نرم‌افزاری). البته بدنه می‌تواند حاوی متن نیز باشد.

Simple Message Example

شکل زیر پیام‌های HTTP را نشان می‌دهد که ممکن است به عنوان بخشی از یک تراکنش ساده ارسال شوند. در این مثال، مرورگر منبع <http://www.joes-hardware.com/tools.html> را درخواست می‌کند.





در این شکل، مرورگر یک پیام درخواست HTTP ارسال می‌کند. درخواست دارای یک متد GET در خط شروع است و منبع محلی مورد درخواست نیز tools.html است. درخواست نشان می‌دهد که ما در حال استفاده از نسخه ۱.۰ پروتکل HTTP هستیم. پیام درخواست هیچ متنی ندارد، زیرا برای دریافت (GET) یک سند ساده از یک سرور به هیچ داده درخواستی نیاز نیست.

سرور یک پیام پاسخ HTTP را ارسال می‌کند. پاسخ شامل شماره نسخه HTTP (HTTP/1.0)، یک کد وضعیت موفقیت (۲۰۰)، یک عبارت دلیل توصیفی (OK)، و یک بلوک از فیلدهای Header پاسخ است و در ادامه نیز Content-Length Body پاسخ که حاوی سند درخواستی است، قرار دارد. طول بدن پاسخ در هدر Content-Type معرفی شده است.





Connections

اکنون که شکل پیام‌های HTTP را ترسیم کرده‌ایم، اجازه دهید برای لحظه‌ای در مورد نحوه حرکت پیام‌ها از مکانی به مکان دیگر، در سراسر اتصالات TCP (Transmission Control Protocol) صحبت کنیم.

TCP/IP

HTTP یک پروتکل لایه کاربردی یا Application نگران جزئیات نابسامان ارتباطات شبکه نیست. در عوض، جزئیات شبکه را به TCP/IP، پروتکل محبوب حمل و نقل اینترنتی قابل اعتماد، واگذار می‌کند.

TCP موارد زیر را ارائه می‌دهد:

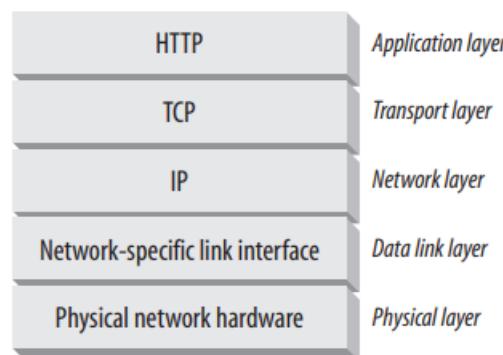
- انتقال داده بدون خطأ
- تحويل سفارشی (داده‌ها همیشه به ترتیبی که ارسال شده‌اند، می‌رسند)
- جریان داده‌های تقسیم‌بندی نشده (می‌تواند داده‌ها را در هر اندازه در هر زمانی بیرون بکشد)

خود اینترنت مبتنی بر TCP/IP است، یک مجموعه لایه‌ای محبوب از پروتکل‌های شبکه سوئیچینگ بسته که توسط رایانه‌ها و دستگاه‌های شبکه در سراسر جهان پشتیبانی شده و با این مدل با هم ارتباط برقرار می‌کنند. TCP/IP ویژگی‌ها و مشکلات شبکه‌ها و سخت‌افزار منفرد را پنهان می‌کند و به رایانه‌ها و شبکه‌ها از هر نوع اجازه می‌دهد تا به طور قابل اعتماد با هم صحبت کنند.

هنگامی که یک اتصال TCP برقرار می‌شود، پیام‌های رد و بدل شده بین رایانه‌های کلاینت و سرور هرگز از بین نمی‌روند، آسیب نمی‌بینند، یا امکان اینکه دریافت نشوند نیز وجود ندارد.

در شرایط شبکه، پروتکل HTTP روی TCP لایه بندی شده است. TCP از HTTP برای انتقال داده‌های پیام خود استفاده می‌کند. به همین ترتیب، TCP روی IP لایه بندی می‌شود (شکل زیر را ببینید).





Connections, IP Addresses, and Port Numbers

قبل از اینکه یک کلاینت HTTP بتواند پیامی را به سرور ارسال کند، باید با استفاده از آدرس‌های IP و شماره پورت، یک اتصال TCP/IP بین کلاینت و سرور برقرار کند.

راه اندازی یک اتصال TCP به نوعی مانند تماس با شخصی در یک دفتر شرکت است. ابتدا شماره تلفن شرکت را شماره گیری می‌کنید. این شما را به سازمان مناسب می‌رساند. سپس، داخلی خاص شخصی را که می‌خواهد با آن تماس بگیرید، شماره گیری می‌کنید.

در TCP، به آدرس IP رایانه سرور و شماره پورت TCP مرتبط با برنامه نرم افزاری خاصی که در حال اجرا بر روی سرور است، نیاز دارید.

همه این‌ها خوب است، اما چگونه می‌توان آدرس IP و شماره پورت سرور HTTP را در وله اول بدست آورد؟ پاسخ URL است. قبلًا اشاره کردیم که URL ها آدرس منابع هستند، بنابراین آن‌ها می‌توانند آدرس IP دستگاهی که منبع را در اختیار دارد به ما ارائه دهند. بباید نگاهی به چند URL بیاندازیم:

<http://207.200.83.29:80/index.html>

<http://www.netscape.com:80/index.html>

<http://www.netscape.com/index.html>

اولین URL دارای آدرس IP دستگاه، 207.200.83.29 و شماره پورت، 80 است.

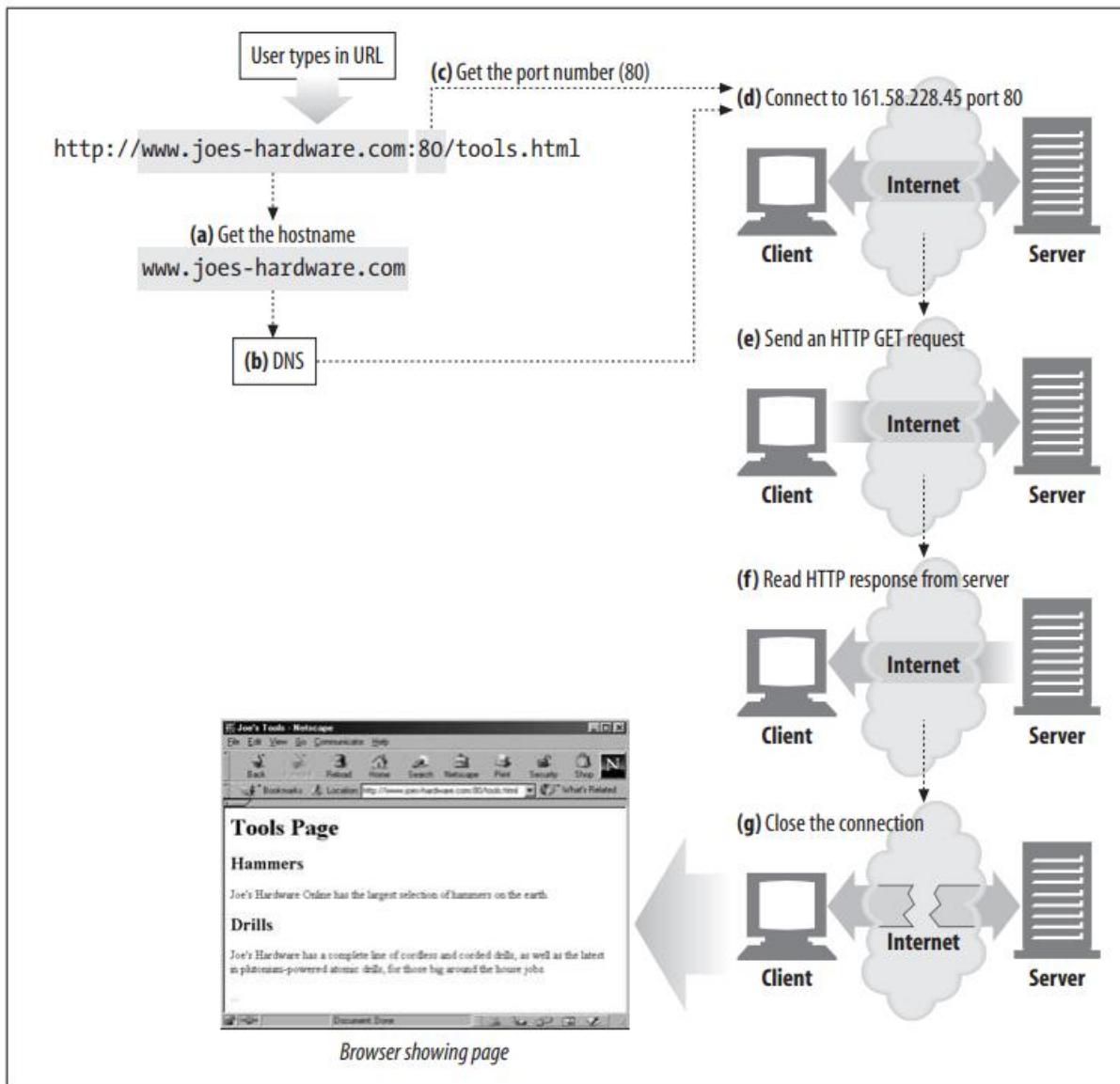
URL دوم آدرس IP عددی ندارد. یک نام دامنه یا نام میزبان ("www.netscape.com") دارد. نام میزبان فقط یک نام مستعار مناسب برای یک آدرس IP است. نام هاست را می‌توان به راحتی از طریق سرویسی به نام DNS یا Domain Name Service به آدرس‌های IP تبدیل کرد.





URL سوم شماره پورت ندارد. هنگامی که شماره پورت در یک URL HTTP وجود ندارد، به صورت پیش فرض پورت ۸۰ در نظر گرفته می‌شود.

با آدرس IP و شماره پورت، یک کلاینت می‌تواند به راحتی از طریق TCP/IP ارتباط برقرار کند. شکل زیر نشان می‌دهد که چگونه یک مرورگر از HTTP برای نمایش یک منبع ساده HTML که در یک سرور قرار دارد، استفاده می‌کند.



مراحلی که در این بخش انجام می‌شود به صورت زیر است:

(a) مرورگر نام میزبان سرور را از URL استخراج می‌کند.





- (b) مرورگر نام میزبان سرور را به آدرس IP سرور تبدیل می کند
- (c) مرورگر شماره پورت (در صورت وجود) را از URL استخراج می کند.
- (d) مرورگر یک اتصال TCP با وب سرور برقرار می کند.
- (e) مرورگر یک پیام درخواست HTTP به سرور ارسال می کند.
- (f) سرور یک پاسخ HTTP را به مرورگر ارسال می کند.
- (g) اتصال بسته می شود و مرورگر سند را نمایش می دهد.

A Real Example Using Telnet

از آنجایی که HTTP از TCP/IP استفاده می کند و مبتنی بر متن است، برخلاف استفاده از فرمتهای باینری مبهم، صحبت مستقیم با یک وب سرور ساده است.

ابزار Telnet شما را به یک درگاه TCP مقصد متصل می کند و خروجی پورت TCP را به شما نمایش می دهد. Telnet معمولاً برای ارتباط از طریق ترمینال به صورت ریموت استفاده می شود، اما به طور کلی می تواند به هر سرور TCP، از جمله سرورهای HTTP متصل شود.

می توانید از ابزار Telnet برای صحبت مستقیم با سرورهای وب استفاده کنید. Telnet به شما امكان می دهد یک اتصال TCP را به یک پورت در یک ماشین باز کنید و کاراکترها را مستقیماً در آن تایپ کنید. وب سرور با شما به عنوان یک کلاینت وب رفتار می کند و هر داده ای که در اتصال TCP برگردانده می شود روی صفحه، نمایش داده می شود.

بیایید از Telnet برای تعامل با یک وب سرور واقعی استفاده کنیم. ما از Telnet برای واکشی سندی با آدرس http://www.joes-hardware.com:80/tools.html استفاده خواهیم کرد.

بیایید مرور کنیم که چه اتفاقی باید بیفتد:

- ابتدا باید آدرس IP مربوط به www.joes-hardware.com را جستجو کرده و یک اتصال TCP به پورت ۸۰ آن دستگاه باز کنیم. Telnet این کار را برای ما انجام می دهد.
- هنگامی که اتصال TCP باز شد، باید درخواست HTTP را تایپ کنیم.
- هنگامی که درخواست کامل شد (با یک خط خالی مشخص می شود)، سرور باید محتوا را در یک پاسخ HTTP ارسال کند و اتصال را ببندد.





درخواست HTTP مثال ما برای <http://www.joes-hardware.com:80/tools.html> در مثال زیر نشان داده شده است. آنچه تایپ کردیم با خط پرنگ نشان داده شده است.

```
% telnet www.joes-hardware.com 80
Trying 161.58.228.45...
Connected to joes-hardware.com.
Escape character is '^].
GET /tools.html HTTP/1.1
Host: www.joes-hardware.com

HTTP/1.1 200 OK
Date: Sun, 01 Oct 2000 23:25:17 GMT
Server: Apache/1.3.11 BSafe-SSL/1.38 (Unix) FrontPage/4.0.4.3
Last-Modified: Tue, 04 Jul 2000 09:46:21 GMT
ETag: "373979-193-3961b26d"
Accept-Ranges: bytes
Content-Length: 403
Connection: close
Content-Type: text/html

<HTML>
<HEAD><TITLE>Joe's Tools</TITLE></HEAD>
<BODY>
<H1>Tools Page</H1>
<H2>Hammers</H2>
<P>Joe's Hardware Online has the largest selection of hammers on the earth.</P>
<H2><A NAME=drills></A>Drills</H2>
<P>Joe's Hardware has a complete line of cordless and corded drills, as well as the latest
in plutonium-powered atomic drills, for those big around the house jobs.</P> ...
</BODY>
</HTML>
Connection closed by foreign host.
```

نام میزبان را جستجو می‌کند و یک اتصال به وب سرور www.joes-hardware.com باز می‌کند که در پورت ۸۰ گوش می‌دهد. سه خط بعد از دستور از Telnet شامل خروجی می‌شود و به ما می‌گوید که یک اتصال برقرار کرده است.

سپس دستور درخواست اصلی خود را که عبارت "GET /tools.html HTTP/1.1" می‌باشد، تایپ می‌کنیم و یک هدر Host که نام میزبان اصلی را ارائه می‌دهد و به دنبال آن یک خط خالی ارسال می‌کنیم و از سرور می‌خواهیم که منبع "tools.html" را از سرور www.joes-hardware.com برای ما دریافت کند. پس از آن، سرور با یک خط پاسخ، چندین Header پاسخ، یک خط خالی و در نهایت Body سند HTML به ما پاسخ می‌دهد.





مراقب باشید که Telnet کلاینت‌های HTTP را به خوبی تقلید می‌کند اما به عنوان یک سرور خوب کار نمی‌کند. و اسکریپت نویسی به صورت خودکار توسط Telnet گزینه مناسبی نیست. برای یک ابزار انعطاف‌پذیرتر، ممکن است بخواهید nc (netcat) را بررسی کنید. ابزار nc به شما این امکان را می‌دهد تا به راحتی ترافیک مبتنی بر TCP و UDP، از جمله HTTP را دستکاری و اسکریپت مناسب خود را برای بررسی آن ایجاد کنید.

Protocol Versions

امروزه چندین نسخه از پروتکل HTTP در حال استفاده است. برنامه‌های کاربردی HTTP باید سخت کار کنند تا به طور قوی تغییرات مختلف پروتکل HTTP را مدیریت کنند. نسخه‌هایی که امروزه در حال استفاده هستند، عبارتند از:

HTTP/0.9

نسخه اولیه HTTP در سال ۱۹۹۱ ایجاد شده و با نام HTTP/0.9 شناخته می‌شود. این پروتکل حاوی بسیاری از مشکلات جدی طراحی است و باید فقط برای تعامل با کلاینت‌های قدیمی استفاده شود. HTTP/0.9 فقط از متدهای GET پشتیبانی می‌کند و از MIME Type محتوای چندرسانه‌ای، هدرهای HTTP یا شماره نسخه پشتیبانی نمی‌کند. HTTP/0.9 در ابتدا برای واکشی (fetch) اشیاء ساده HTML تعریف شد و پس از آن با جایگزین گردید.

HTTP/1.0

پروتکل HTTP نسخه 1.0 اولین نسخه HTTP بود که به طور گسترده به کار گرفته شد. HTTP/1.0 اعداد مربوط به نسخه، هدرهای اضافی و مدیریت اشیاء چند رسانه‌ای را اضافه کرد. HTTP/1.0 پشتیبانی از صفحات وب گرافیکی جذاب و فرم‌های تعاملی را عملی کرد که این کار به ترویج پذیرش گسترده وب جهانی کمک نمود.

HTTP/1.0+

بسیاری از کلاینت‌های وب و سرورهای محبوب در اواسط دهه ۱۹۹۰ به سرعت ویژگی‌هایی را به HTTP اضافه کردند تا نیازهای وب جهانی که به سرعت در حال گسترش بوده و همچنین از نظر تجاری موفق می‌باشد، را برآورده سازند. بسیاری از این ویژگی‌ها، از جمله اتصالات طولانی‌مدت "keepalive"، پشتیبانی از میزبانی مجازی یا Virtual Hosting و پشتیبانی از اتصال پراکسی، به HTTP اضافه شدند و به استانداردهای غیررسمی تبدیل شدند. این نسخه غیررسمی و توسعه یافته HTTP اغلب با عنوان HTTP/1.0+ شناخته می‌شود.





HTTP/1.1

HTTP/1.1 بر اصلاح عیوب معماری در طراحی HTTP، معرفی بهینه‌سازی‌های عملکرد قابل توجه و حذف ویژگی‌های اشتباه تمرکز داشت. HTTP/1.1 همچنین شامل پشتیبانی از برنامه‌های کاربردی وب پیچیده‌تر و استقرارهایی بود که در اواخر دهه ۱۹۹۰ در حال انجام بودند. HTTP/1.1 نسخه فعلی HTTP است.

HTTP-NG (a.k.a. HTTP/2.0)

HTTP-NG یک پیشنهاد اولیه برای یک جانشین معماری برای HTTP/1.1 است که بر بهینه سازی عملکرد قابل توجه و یک چارچوب قدرتمندتر برای اجرای منطق سرور از راه دور تمرکز دارد. تلاش تحقیقاتی HTTP-NG در سال ۱۹۹۸ به پایان رسید و در زمان نگارش این کتاب، هیچ برنامه‌ای برای پیشبرد این پیشنهاد به عنوان جایگزینی برای HTTP/1.1 وجود ندارد.

Architectural Components of the Web

در این فصل که یک Overview از پروتکل HTTP بود، ما بر نحوه ارسال پیام‌های دو برنامه وب (مرورگرهای وب و سرورهای وب) برای اجرای تراکنش‌های اساسی تمرکز کردیم. بسیاری از برنامه‌های کاربردی وب دیگر وجود دارند که شما با آن‌ها در اینترنت تعامل دارید. در این بخش، چندین برنامه مهم دیگر را بیان خواهیم کرد، از جمله آن‌ها عبارتند از:

Proxies: واسطه‌های HTTP که بین کلاینت‌ها و سرورها قرار می‌گیرند.

Caches: انبارهای HTTP که کپی‌های صفحات وب محبوب را نزدیک کلاینت‌ها نگه می‌دارند.

Gateways: وب سرورهای ویژه‌ای که به برنامه‌های کاربردی دیگر متصل می‌شوند.

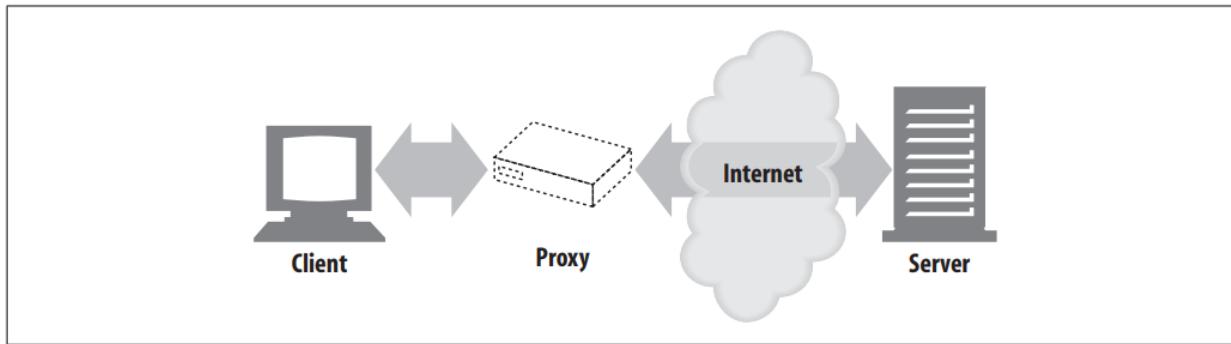
Tunnels: پراکسی‌های ویژه‌ای که کورکورانه (blindly) ارتباطات HTTP را هدایت می‌کنند.

Agents: کلاینت‌های وب نیمه هوشمند که درخواست‌های HTTP خودکار را انجام می‌دهند.

Proxies

همانطور که در شکل زیر نشان داده شده است، یک پروکسی بین یک کلاینت و یک سرور قرار می‌گیرد و تمام درخواست‌های HTTP کلاینت را دریافت می‌کند و درخواست‌ها را به سرور ارسال می‌کند (شاید پس از اصلاح درخواست‌ها). این برنامه‌ها به عنوان یک پروکسی برای کاربر عمل می‌کنند و از طرف کاربر به سرور دسترسی دارند.

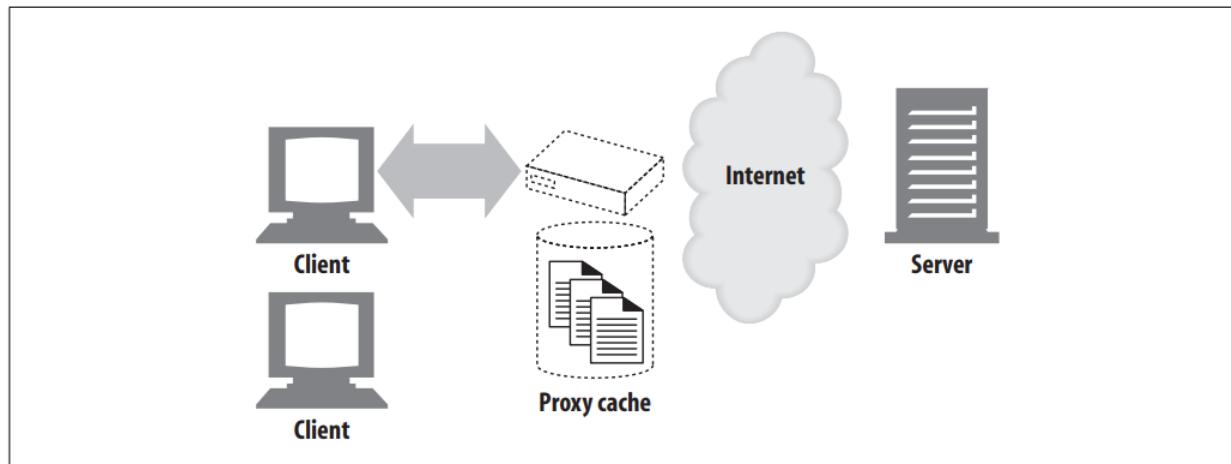




پروکسی‌ها اغلب برای امنیت استفاده می‌شوند و به عنوان واسطه‌های قابل اعتماد عمل می‌کنند که از طریق آن‌ها تمام ترافیک وب جریان می‌یابد. پروکسی‌ها همچنین می‌توانند درخواست‌ها و پاسخ‌ها را فیلتر کنند. به عنوان مثال، از این ویژگی می‌توان برای شناسایی ویروس‌هایی که در دانلودهای شرکت وجود دارند و یا فیلتر کردن محتوای بزرگ‌سالان استفاده نمود. در فصل‌های آتی به مباحث پروکسی پرداخته خواهد شد.

Caches

کش وب یا caching proxy نوع خاصی از سرور پروکسی HTTP است که کپی‌هایی از اسناد محبوب را که از طریق پروکسی عبور می‌کنند نگه می‌دارد. مشتری بعدی که همان سند را درخواست می‌کند می‌تواند از نسخه شخصی Cache ارائه شود.



یک کلاینت ممکن است بتواند یک سند را خیلی سریع‌تر از یک Cache در نزدیکی دانلود کند تا از یک وب سرور دور HTTP امکانات زیادی را برای موثرتر کردن Cache و تنظیم تازگی و حفظ حریم خصوصی محتوای آن تعریف می‌کند. ما در فصل ۷ فناوری Caching را پوشش می‌دهیم.

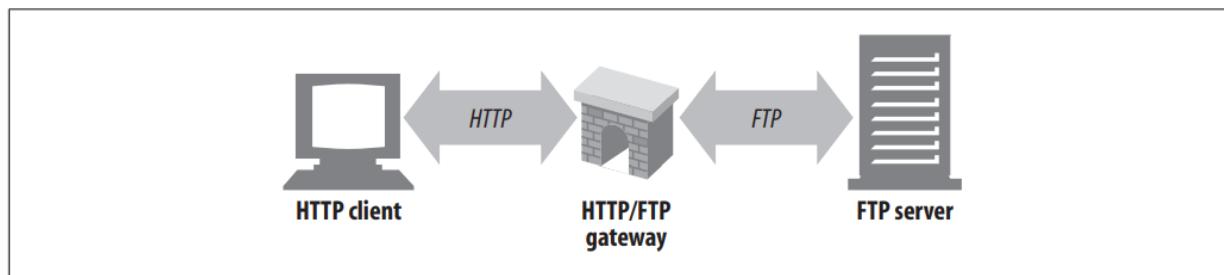




Gateways

Gateway ها سرورهای خاصی هستند که به عنوان واسطه برای سرورهای دیگر عمل می‌کنند. آن‌ها اغلب برای تبدیل ترافیک HTTP به پروتکل دیگری استفاده می‌شوند. یک Gateway همیشه درخواست‌ها را به گونه‌ای دریافت می‌کند که گویی سرور اصلی برای منبع است. کلاینت ممکن است متوجه نباشد که با یک Gateway ارتباط برقرار می‌کند.

به عنوان مثال، یک HTTP/FTP Gateway برای URI های FTP از طریق درخواست‌های HTTP دریافت می‌کند اما اسناد را با استفاده از پروتکل FTP واکشی (fetch) می‌کند (شکل زیر را ببینید). سند حاصل در یک پیام HTTP بسته بندی شده و برای مشتری ارسال می‌شود. در فصل ۸ در مورد Gateway ها بحث می‌کنیم.

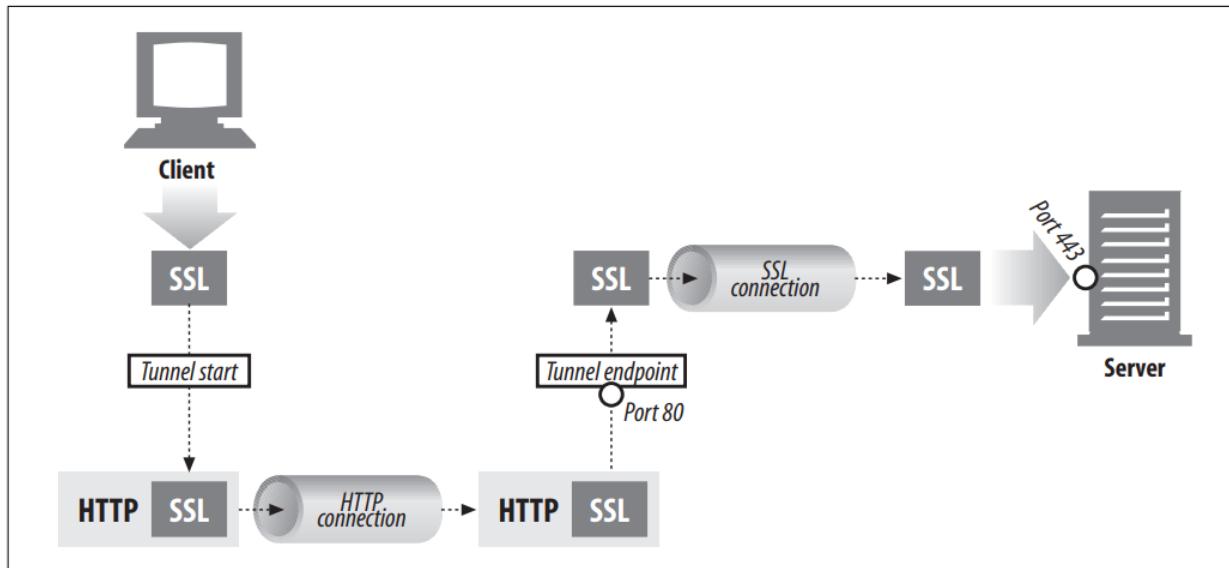


Tunnels

تونل‌ها برنامه‌های HTTP ای هستند که پس از راه اندازی، داده‌های خام را به صورت کورکورانه بین دو اتصال رله می‌کنند. تونل‌های HTTP اغلب برای انتقال داده‌های غیر HTTP بر روی یک یا چند اتصال HTTP، بدون نگاه کردن به داده‌ها استفاده می‌شوند.

یکی از کاربردهای محبوب تونل‌های SSL، حمل ترافیک HTTP رمزگذاری شده از طریق یک اتصال HTTP است. همانطور که در شکل زیر نشان داده شده است، یک تونل HTTP/SSL یک درخواست HTTP برای ایجاد یک اتصال خروجی به یک آدرس و پورت مقصد دریافت می‌کند، سپس به تونل کردن ترافیک رمزگذاری شده روی کانال HTTP ادامه می‌دهد تا بتواند کورکورانه به سرور مقصد منتقل شود.





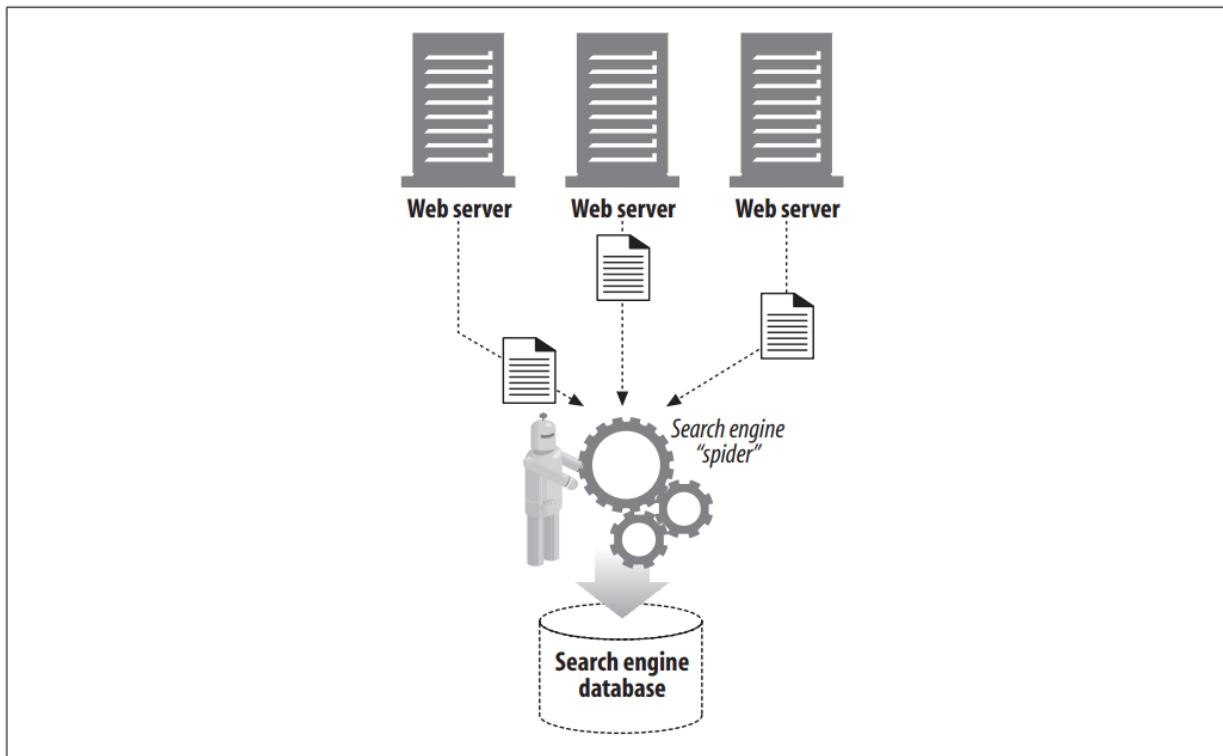
Agents

به عنوان مثال، User Agent های خودکاری (machine-automated user agents) وجود دارند که به طور مستقل در وب سرگردان هستند و تراکنشهای HTTP و واکشی محتوا را بدون نظارت انسان انجام می‌دهند. هر برنامه‌ای که درخواستهای وب را صادر می‌کند یک HTTP Agent است. تا کنون، ما فقط در مورد یک نوع HTTP Agent صحبت کرده ایم: مرورگرهای وب. اما بسیاری از انواع دیگری از User Agent ها وجود دارد.

به عنوان مثال، User Agent های خودکاری (machine-automated user agents) وجود دارند که به طور مستقل در وب سرگردان هستند و تراکنشهای HTTP و واکشی محتوا را بدون نظارت انسان انجام می‌دهند. این Agent های خودکار اغلب نامهای مختلفی مانند web robots یا spiders دارند. (شکل زیر را ببینید).

Spider ها در وب سرگردان هستند تا آرشیوهای مفیدی از محتوای وب ایجاد کنند، به عنوان مثال می‌توان به پایگاه داده یک موتور جستجو یا یک کاتالوگ محصول برای یک ربات مقایسه خرید، اشاره کرد.





For More Information

فصل‌های بعدی این کتاب HTTP را با جزئیات بیشتری بررسی می‌کند، اما ممکن است متوجه شوید که برخی از منابع زیر حاوی پیش‌زمینه‌های مفیدی درباره موضوعات خاصی است که در این فصل به آن پرداختیم.

HTTP Pocket Reference

<http://www.w3.org/Protocols/>

<http://www.ietf.org/rfc/rfc2616.txt>

<http://www.ietf.org/rfc/rfc1945.txt>

<http://www.w3.org/Protocols/HTTP/AsImplemented.html>

<http://www.w3.org/Protocols/WhyHTTP.html>

<http://www.w3.org/History.html>

<http://www.w3.org/DesignIssues/Architecture.html>

<http://www.ietf.org/rfc/rfc2396.txt>

<http://www.ietf.org/rfc/rfc2141.txt>





<http://www.ietf.org/rfc/rfc2046.txt>

<http://www.wrec.org/Drafts/draft-ietf-wrec-taxonomy-06.txt>





فصل دوم – URLs and Resources

اینترنت را به عنوان شهری غولپیکر و در حال گسترش، پر از مکان‌های دیدنی و کارهایی برای انجام در نظر بگیرید. شما و سایر ساکنان و گردشگران این جامعه پر رونق از نام‌گذاری استاندارد برای جاذبه‌ها و خدمات گستردۀ شهر استفاده می‌کنید. از آدرس‌های خیابان برای موزه‌ها، رستوران‌ها و خانه‌های مردم استفاده می‌کنید. از شماره تلفن‌های آتش نشانی، منشی رئیس و مادرتان استفاده می‌کنید.

هر چیزی یک نام استاندارد دارد تا به مرتب کردن منابع شهر کمک کند. کتاب‌ها دارای شماره شابک، اتوبوس‌ها شماره مسیریابی، حساب‌های بانکی دارای شماره حساب و افراد دارای شماره تامین اجتماعی یا کد ملی هستند. فردا در گیت ۳۱ فرودگاه با شرکای تجاری خود ملاقات خواهید کرد. هر روز صبح سوار قطار خط قرمز می‌شوید و از ایستگاه میدان کنдал خارج می‌شوید.

و از آنجا که همه در مورد استانداردهای این نام‌های مختلف توافق داشتند، می‌توانیم به راحتی گنجینه‌های شهر را با یکدیگر به اشتراک بگذاریم. می‌توانید به راننده تاکسی بگویید که شما را به خیابان مک‌آلیستر ۲۴۶ برساند و او متوجه منظور شما خواهد شد (حتی اگر مسیر طولانی را طی کند).

URL یا Uniform Resource Locator ها، نام‌های استاندارد شده برای منابع اینترنت هستند. URL ها به بخش‌هایی از اطلاعات الکترونیکی اشاره می‌کنند و به شما می‌گویند کجا قرار دارند و اینکه چگونه باید با آن‌ها تعامل کنید.

در این فصل، موارد زیر بررسی خواهد شد:

- ساختار URL و معنای اجزای مختلف URL و انجام آن
- میانبرهای URL که بسیاری از کلاینت‌های وب از آن‌ها پشتیبانی می‌کنند، از جمله URL های نسبی و URL های گستردۀ
- URL Encoding و قوانین مربوط به کاراکترها
- طرح‌های URL رایج که از انواع سیستم‌های اطلاعات اینترنتی پشتیبانی می‌کنند
- آینده URL ها، از جمله نام منابع یکسان (URN) - چارچوبی برای پشتیبانی از اشیایی که از مکانی به مکان دیگر حرکت می‌کنند و در عین حال نام‌های پایدار را حفظ می‌کنند.





Navigating the Internet's Resources

URL ها مکان‌های مربوط به منابعی هستند که مرورگر شما برای یافتن اطلاعات به آن‌ها نیاز دارد. آن‌ها به افراد و برنامه‌ها اجازه می‌دهند میلیاردها منبع داده را در اینترنت پیدا کنند، استفاده کنند و به اشتراک بگذارند.

URL ها نقطه دسترسی افراد به HTTP و پروتکل‌های دیگر هستند: یک شخص، مرورگر را باز نموده و به URL مورد نظر اشاره می‌کند و در پشت صحنه، مرورگر پیام‌های پروتکل مناسب را برای دریافت منبع مورد نظر ارسال می‌کند.

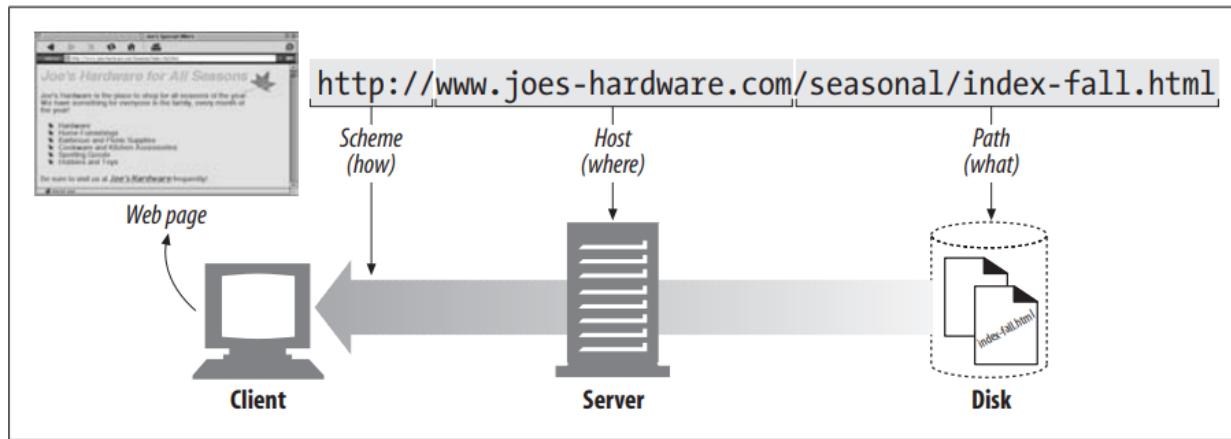
URL ها در واقع زیرمجموعه‌ای از یک کلاس عمومی‌تر از شناسه منبع هستند که uniform resource identifier یا URI نامیده می‌شود. URI ها یک مفهوم کلی هستند که از دو زیر مجموعه اصلی URL و URN تشکیل شده است. URL ها منابع را با توصیف محل قرارگیری منابع، شناسایی می‌کنند، در حالی که URN ها (که در ادامه این فصل به آن‌ها خواهیم پرداخت) منابع را بدون توجه به محل فعلی آن‌ها با نام شناسایی می‌کنند. مشخصات HTTP از مفهوم کلی تر URI‌ها به عنوان شناسه‌های منبع خود استفاده می‌کند. اما در عمل، برنامه‌های HTTP فقط با URL سروکار دارند. در سرتاسر این کتاب، گاهی اوقات به URI و URL به جای هم اشاره می‌کنیم، اما تقریباً همیشه در مورد URL ها صحبت می‌کنیم.

آدرس زیر را در نظر بگیرید:

<http://www.joes-hardware.com/seasonal/index-fall.html>

- اولین قسمت از URL که برابر با HTTP است، به عنوان URL scheme شناخته می‌شود. این طرح به یک کلاینت وب، نحوه دسترسی به منبع را می‌گوید. در این مورد، URL می‌گوید از پروتکل HTTP استفاده گردد.
- قسمت دوم URL (www.joes-hardware.com) مکان سرور است. این به کلاینت وب می‌گوید که منبع در کجا میزبانی می‌شود.
- قسمت سوم URL (/seasonal/index-fall.html) مسیر منبع است. مسیر نشان می‌دهد که چه منبع محلی خاصی روی سرور درخواست شده است.





URL‌ها می‌توانند شما را به منابع موجود از طریق پروتکل‌هایی غیر از HTTP هدایت کنند. آن‌ها می‌توانند شما را به هر منبعی در اینترنت، مانند ایمیل یک شخص هدایت کنند:

<mailto:president@whitehouse.gov>

همچنین FTP از جمله پروتکل‌های دیگری است که به منظور انتقال فایل از آن استفاده می‌شود:

<ftp://ftp.lots-o-books.com/pub/complete-price-list.xls>

علاوه بر موارد مذکور، می‌توان به فیلم‌هایی که از سرورهای پخش ویدئو میزبانی می‌شوند نیز اشاره کرد:

rtsp://www.joes-hardware.com:554/interview/cto_video

URL‌ها راهکاری برای نامگذاری یکسان منابع را فراهم می‌کنند. اکثر URL‌ها ساختاری مشابه زیر دارند:

"scheme://server location/path"

بنابراین، برای هر منبع موجود و هر روشی برای به دست آوردن آن منابع، شما یک راه واحد برای نامگذاری هر منبع دارید تا هر کسی بتواند از آن نام برای پیدا کردن آن استفاده کند.

The Dark Days Before URLs

قبل از وب و آدرس‌های اینترنتی، مردم برای دسترسی به داده‌های توزیع شده در سراسر شبکه به مجموعه‌ای از برنامه‌های کاربردی تکیه می‌کردند. اکثر مردم به اندازه کافی خوش شانس نبودند که همه برنامه‌های کاربردی مناسب را داشته باشند یا برای استفاده از آن‌ها به اندازه کافی زرنگ و صبور نبودند.

قبل از آمدن URL‌ها، اگر می‌خواستید فایل complete-catalog.xls را با یکی از دوستان خود به اشتراک بگذارید، باید چیزی شبیه به این می‌گفتید:





"از FTP برای اتصال به <ftp://ftp.joes-hardware.com/anonymous> استفاده کنید. به عنوان <ftp://ftp.joes-hardware.com/anonymous> (استفاده از عبارت anonymous هم به عنوان نام کاربری و هم کلمه عبور) وارد شوید. به دایرکتوری pub بروید. به حالت باینری بروید. اکنون فایلی با نام [complete-catalog.xls](ftp://ftp.joes-hardware.com/complete-catalog.xls) را در فایل سیستم محلی خود دانلود کرده و آن را مشاهده کنید."

امروزه، مرورگرهایی مانند Microsoft Internet Explorer و Firefox بسیاری از این قابلیت‌ها را برای شما فراهم می‌کند. این اپلیکیشن‌ها با استفاده از URL‌ها می‌توانند به بسیاری از منابع دسترسی پیدا کنند. به جای دستورالعمل‌های پیچیده بالا، می‌توانید بگویید:

"مرورگر خود را باز نموده و به آدرس <ftp://ftp.lots-o-books.com/pub/complete-catalog.xls> بروید."

URL‌ها ابزاری برای برنامه‌های کاربردی فراهم کرده‌اند تا از نحوه دسترسی به یک منبع آگاه شوند. در واقع، احتمالاً بسیاری از کاربران از پروتکل‌ها و روش‌های دسترسی مرورگرهایشان برای دریافت منابعی که درخواست می‌کنند بی‌اطلاع هستند.

با مرورگرهای وب، دیگر نیازی به خبرخوان برای خواندن اخبار اینترنتی یا کلاینت FTP برای دسترسی به فایل‌های روی سرورهای FTP ندارید. برای ارسال و دریافت پیام‌های ایمیل به برنامه پست الکترونیکی نیاز ندارید. نشانی‌های وب به ساده‌سازی دنیای آنلاین کمک کرده‌اند و به مرورگر اجازه می‌دهند در مورد نحوه دسترسی و مدیریت منابع هوشمندانه عمل کند. برنامه‌ها می‌توانند از URL‌ها برای ساده‌سازی دسترسی به اطلاعات استفاده کنند.

لازم به ذکر است که WinSCP نیز نمونه‌ای از ابزارهایی است که به منظور اتصال به یک سرور از آن استفاده می‌شود.

URL‌ها برای شما و مرورگرتان تمام آنچه را که برای یافتن یک اطلاعات نیاز دارید، فراهم می‌کنند. آن‌ها منبع خاصی را که می‌خواهید، جایی که در آن قرار دارد و نحوه دریافت آن را تعریف می‌کنند.

URL Syntax

URL‌ها وسیله‌ای برای مکان یابی هر منبعی در اینترنت هستند، اما این منابع را می‌توان با scheme‌های مختلف (مانند SMTP، FTP، HTTP) در دسترس قرار داد و ساختار URL از طرح دیگر متفاوت خواهد بود.





آیا این بدان معناست که هر URL scheme متفاوت، ساختار و Syntax ای کاملاً متفاوت دارد؟ در عمل، خیر. اکثر URL ها به یک URL پایبند هستند و همپوشانی قابل توجهی در سبک و Syntax بین scheme های مختلف وجود دارد.

اکثر URL scheme ها، ساختار URL خود را بر اساس این قالب کلی قرار می‌دهند:

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

تقریباً هیچ URL حاوی همه این اجزا نیست. سه بخش مهم URL عبارتند از: host scheme و path. جدول زیر اجزای مختلف را به صورت خلاصه نشان می‌دهد.

Component	Description	Default value
scheme	Which protocol to use when accessing a server to get a resource.	None
user	The username some schemes require to access a resource.	anonymous
password	The password that may be included after the username, separated by a colon (:).	<Email address>
host	The hostname or dotted IP address of the server hosting the resource.	None
port	The port number on which the server hosting the resource is listening. Many schemes have default port numbers (the default port number for HTTP is 80).	Scheme-specific
path	The local name for the resource on the server, separated from the previous URL components by a slash (/). The syntax of the path component is server- and scheme-specific. (We will see later in this chapter that a URL's path can be divided into segments, and each segment can have its own components specific to that segment.)	None
params	Used by some schemes to specify input parameters. Params are name/value pairs. A URL can contain multiple params fields, separated from themselves and the rest of the path by semicolons (;).	None
query	Used by some schemes to pass parameters to active applications (such as databases, bulletin boards, search engines, and other Internet gateways). There is no common format for the contents of the query component. It is separated from the rest of the URL by the "?" character.	None
frag	A name for a piece or part of the resource. The frag field is not passed to the server when referencing the object; it is used internally by the client. It is separated from the rest of the URL by the "#" character.	None

به عنوان مثال، <http://www.joes-hardware.com:80/index.html> را در نظر بگیرید. scheme برابر با "http" است، host برابر با "www.joes-hardware.com" است، port برابر با "80" و path برابر با "index.html/" است.





Schemes: What Protocol to Use

این scheme واقعاً شناسه اصلی نحوه دسترسی به یک منبع معین است و به برنامه‌ای که URL را تفسیر می‌کند، می‌گوید که چه پروتکلی باید صحبت کند. در URL ساده HTTP ما، scheme برابر با "http" است.

scheme باید با یک کاراکتر الفبایی شروع شود و با اولین کاراکتر ":" از بقیه URL جدا می‌شود. نام scheme حساس به حروف کوچک و بزرگ نمی‌باشد، بنابراین URL های "http://www.joes-hardware.com" و "HTTP://www.joes-hardware.com" معادل هم هستند.

Hosts and Ports

برای یافتن یک منبع در اینترنت، یک برنامه کاربردی باید بداند که چه ماشینی، منبع مورد نظر را میزبانی می‌کند و در کجا می‌تواند سروری را که به منبع مورد نظر دسترسی دارد، پیدا کند. اجزای host و پورت URL این دو اطلاعات را ارائه می‌دهند.

Host Component، ماشین میزبان در اینترنت را که به منبع دسترسی دارد شناسایی می‌کند. نام را می‌توان به عنوان hostname، مانند بالا ("www.joes-hardware. com") یا به عنوان یک آدرس IP ارائه کرد. به عنوان مثال، دو URL زیر به یک منبع اشاره می‌کنند - اولی به سرور با hostname و دومی به آدرس IP آن اشاره می‌کند:

`http://www.joes-hardware.com:80/index.html`

`http://161.58.228.45:80/index.html`

Port Component، پورت شبکه‌ای را که سرور به آن گوش می‌دهد، شناسایی می‌کند. برای HTTP که از TCP استفاده می‌کند، پورت پیش فرض ۸۰ است.

Usernames and Passwords

اجزای جالب‌تر، نام کاربری و رمز عبور است. بسیاری از سرورها قبل از اینکه بتوانید از طریق آن‌ها به داده‌ها دسترسی داشته باشید به یک نام کاربری و رمز عبور نیاز دارند. سرورهای FTP یک مثال رایج در این مورد هستند: در اینجا چند نمونه هستند:

`ftp://ftp.prep.ai.mit.edu/pub-gnu`

`ftp://anonymous@ftp.prep.ai.mit.edu/pub-gnu`

`ftp://anonymous:my_passwd@ftp.prep.ai.mit.edu/pub-gnu`





http://joe:joespasswd@www.joes-hardware.com/sales_info.txt

مثال اول هیچ جزء مربوط به نام کاربری یا رمز عبوری ندارد، فقط Scheme، Host و Path استاندارد وجود دارد. اگر برنامه‌ای از URL Scheme ای مانند FTP استفاده می‌کند که به نام کاربری و رمز عبور نیاز دارد، معمولاً یک نام کاربری و رمز عبور پیش‌فرض را در صورت عدم ارائه آن وارد می‌کند.

به عنوان مثال، اگر یک URL FTP بدون تعیین نام کاربری و رمز عبور به مرورگر خود بدهید، "anonymous" به عنوان نام کاربری برای شما درج شده و همچنین یک رمز عبور پیش‌فرض نیز برای شما ارسال می‌شود (مرورگر IE، عبارت Mozilla Navigator و IEUser را ارسال می‌کند).

در مثال دوم همانطور که در URL مشاهده می‌شود، یک نام کاربری به عنوان "anonymous" مشخص شده است. این نام کاربری، همراه با مؤلفه Host، دقیقاً شبیه یک آدرس ایمیل است. کاراکتر "@" اجزای نام کاربری و رمز عبور را از بقیه URL جدا می‌کند.

در مثال سوم، هم نام کاربری ("my_passwd") و هم رمز عبور ("anonymous") مشخص شده اند که با کاراکتر ":" از هم جدا شده اند.

Paths

مؤلفه مسیر URL مشخص می‌کند که منبع در کجا می‌باشد. مسیر اغلب شبیه یک مسیر سیستم فایل سلسله مراتبی است. مثلا:

<http://www.joes-hardware.com:80/seasonal/index-fall.html>

مسیر یا Path در این URL، "/seasonal/index-fall.html" است که شبیه یک مسیر سیستم فایل در یک فایل سیستم یونیکس است. مسیر یا Path اطلاعاتی است که سرور برای مکان یابی منبع نیاز دارد. مؤلفه مسیر برای URL های HTTP را می‌توان به بخش‌هایی که هر کدام با کاراکترهای "/" از هم جدا شده اند تقسیم کرد (دوباره، مانند مسیر فایل در یک سیستم فایل یونیکس). هر قطعه مسیر می‌تواند پارامترهای خاص خود را داشته باشد.

Parameters

برای بسیاری از Scheme ها، یک میزبان ساده و مسیر رسیدن به شی کافی نیست. جدا از اینکه سرور به چه پورتی گوش می‌دهد و حتی اینکه آیا با نام کاربری و رمز عبور به منبع دسترسی دارید یا خیر، بسیاری از پروتکل‌ها برای کار کردن، به اطلاعات بیشتری نیاز دارند.





برنامه‌های کاربردی که URL ها را تفسیر می‌کنند برای دسترسی به منبع، به این پارامترهای پروتکل نیاز دارند. در غیر این صورت، سرور طرف مقابل ممکن است پاسخ درستی به درخواست نداده و یا بدتر از آن، ممکن است منجر به اشتباه در سرویس دهی شود. به عنوان مثال، پروتکلی مانند FTP را در نظر بگیرید که دارای دو حالت انتقال، باینری و متن است. شما نمی‌خواهید تصویر باینری شما در حالت متنی منتقل شود، زیرا تصویر باینری می‌تواند بهم ریخته شود.

برای ارسال پارامترهای ورودی به برنامه‌ها جهت برقراری ارتباط صحیح با سرور، URL‌ها دارای یک مؤلفه با نام پارامتر هستند. این مؤلفه فقط فهرستی از جفت حالت name/value در URL است که از بقیه URL (و از یکدیگر) با مقدار «;» جدا شده‌اند. آن‌ها هر گونه اطلاعات اضافی را که برای دسترسی به منبع نیاز دارند به برنامه‌ها ارائه می‌کنند. به عنوان مثال:

`ftp://prep.ai.mit.edu/pub-gnu?type=d`

در این مثال، یک پارامتر type=d وجود دارد که نام آن "type" و مقدار آن "d" است.

همانطور که قبلاً ذکر کردیم، مؤلفه مسیر برای URL های HTTP را می‌توان به بخش‌های جزئی تر تقسیم کرد. هر بخش می‌تواند پارامترهای خاص خود را داشته باشد. به عنوان مثال:

`http://www.joes-hardware.com/hammers;sale=false/index.html;graphics=true`

در این مثال دو بخش مسیر با نام‌های hammers و index.html وجود دارد. بخش hammers دارای پارامتر param بوده و مقدار آن false است. بخش index.html دارای پارامتری به نام graphics بوده و مقدار آن true می‌باشد.

Query Strings

از برخی منابع، مانند سرویس‌های پایگاه داده، می‌توان سؤال یا پرس و جو (Query) کرد تا نوع منبع درخواستی را محدود کند.

فرض کنید فروشگاه سخت افزار Joe Lیستی از موجودی‌های فروخته نشده را در یک پایگاه داده نگهداری می‌کند و امکان پرس و جو به منظور شناسایی محصولات موجود را فراهم می‌کند. URL زیر ممکن است برای پرس و جو از پایگاه داده وب استفاده شود تا ببیند آیا محصول شماره ۱۲۷۳۱ موجود است یا خیر:

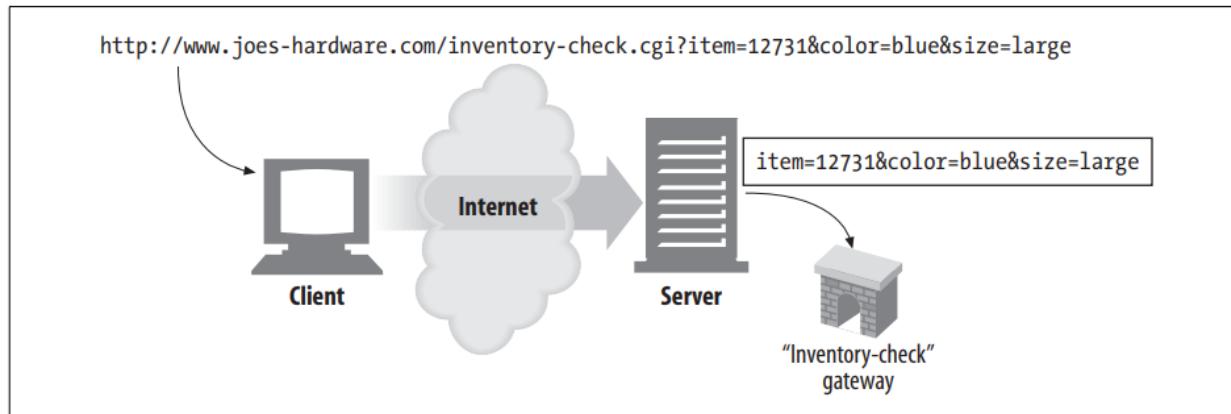
`http://www.joes-hardware.com/inventory-check.cgi?item=12731`





آدرس بالا، شبیه به سایر URL هایی است که تا به حال مشاهده نموده ایم. موضوع جدید مواردی است که در سمت راست علامت سوال (?) قرار دارد. به این بخش مؤلفه پرس و جو (Query) می‌گویند.

شکل زیر نمونه‌ای از یک مؤلفه پرس و جو را نشان می‌دهد که به سروری منتقل می‌شود. پرس و جو بررسی می‌کند که آیا یک کالای خاص با شناسه ۱۲۷۳۱، در اندازه بزرگ و رنگ آبی، موجود هست یا خیر.



هیچ الزامی برای قالب مؤلفه پرس و جو وجود ندارد، به جز اینکه برخی از کاراکترها غیرقانونی هستند، همانطور که در ادامه این فصل خواهیم دید. طبق قرارداد، بسیاری از **Gateway** ها انتظار دارند که رشته پرس و جو یا Query String به صورت مجموعه‌ای از "name=value" که با کاراکترهای "&" از هم جدا شده‌اند، قالب‌بندی شود:

`http://www.joes-hardware.com/inventory-check.cgi?item=12731&color=blue`

در این مثال، دو جفت name/value در مؤلفه پرس و جو وجود دارد که شامل item=12731 و color=blue می‌باشد.

Fragments

برخی از انواع منابع مانند HTML را می‌توان فراتر از سطح منبع تقسیم بندی کرد. به عنوان مثال، برای یک سند متنی بزرگ با بخش‌های مختلف موجود در آن، URL منبع به کل سند متنی اشاره می‌کند، اما در حالت ایده‌آل می‌توانید بخش‌های درون منبع را نیز مشخص کنید.

برای ارجاع دادن به بخش‌ها یا قطعات یک منبع، URL ها از یک مؤلفه با نام **Frag** برای شناسایی قطعات در یک منبع پشتیبانی می‌کنند. به عنوان مثال، یک URL می‌تواند به یک تصویر یا بخش خاص در یک سند HTML اشاره کند.



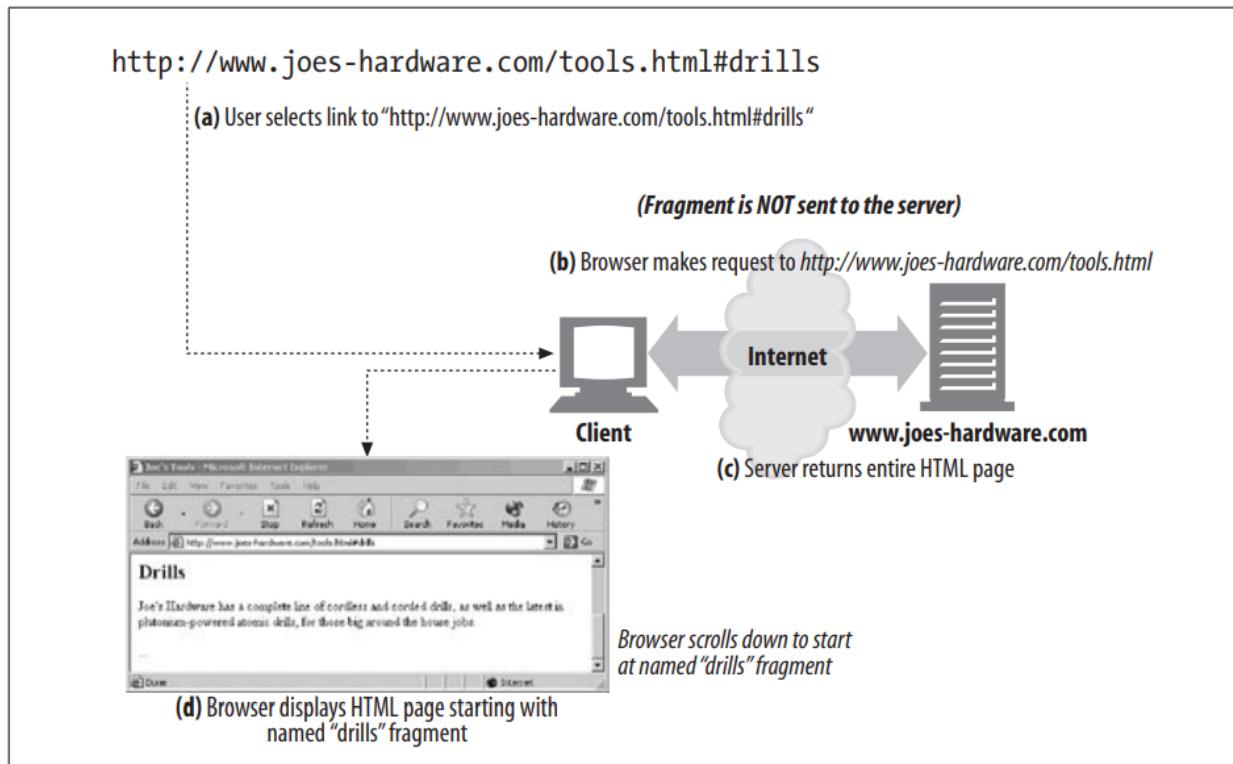


یک Frag در سمت راست یک URL قرار می‌گیرد و قبل از آن یک کاراکتر # وجود دارد. به عنوان مثال:

<http://www.joes-hardware.com/tools.html#drills>

در این مثال، Frag به بخشی از صفحه وب tools.html اشاره می‌کند که این قسمت "drills" نام دارد.

از آنجایی که سرورهای HTTP معمولاً فقط با کل اشیاء سروکار دارند، نه با قطعاتی از اشیاء، کلاینتها قطعات یا Frag ها را به سرور منتقل نمی‌کنند (شکل زیر را ببینید). پس از اینکه مرورگر شما کل منبع را از سرور دریافت کرد، سپس از Frag برای نمایش بخشی از منبع مورد علاقه شما استفاده می‌شود.



URL Shortcuts

کلاینت‌های وب چند میانبر URL را در ک نموده و از آن‌ها استفاده می‌کنند. بسیاری از مرورگرهای همچنین از automatic expansion مربوط به URL‌ها پشتیبانی می‌کنند، جایی که کاربر می‌تواند یک بخش کلیدی URL را تایپ کند و مرورگر مابقی آن را تکمیل خواهد کرد.





Relative URLs

URL ها به دو نوع مطلق (absolute) و نسبی (relative) تقسیم بندی می‌شوند. تا کنون، ما فقط به URL های مطلق نگاه کرده ایم. با یک URL مطلق، شما تمام اطلاعات مورد نیاز برای دسترسی به یک منبع را دارید.

از سوی دیگر، URL های نسبی، ناقص هستند. برای به دست آوردن تمام اطلاعات مورد نیاز برای دسترسی به یک منبع از یک URL نسبی، باید آن را نسبت به URL دیگر یا `base` آن تفسیر کنید.

URL های نسبی یک نماد کوتاه‌نویسی مناسب برای URL ها هستند. اگر با زبان HTML کار کرده باشید، از این آدرس‌ها به عنوان یک میانبر عالی استفاده کرده اید. مثال زیر یک سند HTML را نشان می‌دهد که در آن از URL نسبی استفاده شده است.

```
<HTML>
<HEAD><TITLE>Joe's Tools</TITLE></HEAD>
<BODY>
<H1> Tools Page </H1>
<H2> Hammers <H2>
<P> Joe's Hardware Online has the largest selection of <A HREF=".//hammers.html">hammers
</A> on earth.
</BODY>
</HTML>
```

در سند HTML، یک لینک حاوی آدرس `/hammers.html` وجود دارد. این URL ناقص به نظر می‌رسد، اما یک URL نسبی قانونی است. می‌توان آن را نسبت به URL سندی که در آن یافت می‌شود تفسیر کرد.

ساختار مختصر URL نسبی، به نویسنده‌گان HTML این امکان را می‌دهد که Host Scheme و سایر اجزا را از URL ها حذف کنند. این اجزا را می‌توان از طریق URL اصلی منبعی که در آن قرار دارند استنتاج کرد. URL های منابع دیگر نیز می‌توانند در این خلاصه‌نویسی مشخص شوند. `base` اصلی ما در این مثال URL زیر است که کد HTML ما را شامل می‌شود:

`http://www.joes-hardware.com/tools.html`

با استفاده از این URL به عنوان `base`، می‌توانیم اطلاعاتی که در URL نسبی وجود ندارد را استخراج کنیم. ما می‌دانیم که منبع URL `/hammers.html` یا Host Scheme را نمی‌دانیم. با استفاده از URL پایه یا `www.joes`، می‌توانیم استنباط کنیم که `http` مورد نظر Scheme بوده و `Host` برابر با `hardware.com` می‌باشد. شکل زیر این موضوع را نشان می‌دهد:





URL های نسبی فقط قطعات یا تکه هایی از URL ها هستند. برنامه هایی که URL ها را پردازش می کنند (مانند مرورگر شما) باید بتوانند تبدیل URL های نسبی و مطلق را انجام دهند.

همچنین شایان ذکر است که URL های نسبی راه مناسبی برای قابل حمل نگه داشتن مجموعه ای از منابع (مانند صفحات HTML) را فراهم می کنند. اگر از URL های نسبی استفاده می کنید، می توانید مجموعه ای از اسناد را جابه جا کنید و همچنان لینک های آنها کار کنند، زیرا آنها نسبت به `base` جدید تفسیر می شوند.

Base URLs

اولین مرحله در فرآیند تبدیل، یافتن یک URL پایه به عنوان یک نقطه مرجع برای URL نسبی عمل می کند که می تواند از چند جا آمده باشد:

- به صراحت در منبع ارائه شده است

برخی منابع به صراحت URL پایه را مشخص می کنند. به عنوان مثال، یک سند HTML ممکن است شامل یک تگ HTML باشد که URL پایه را تعریف می کند تا به وسیله آن همه URL های مرتبط در آن سند تبدیل شوند.

Encapsulating Resource URL •

اگر URL نسبی در منبعی یافت شود که به صراحت یک URL پایه را مشخص نکرده است، مانند نمونه زیر که قبلانیز به آن اشاره شده بود، می تواند از URL منبعی که در آن جاسازی شده است به عنوان پایه استفاده کند.





```
<HTML>
<HEAD><TITLE>Joe's Tools</TITLE></HEAD>
<BODY>
<H1> Tools Page </H1>
<H2> Hammers <H2>
<P> Joe's Hardware Online has the largest selection of <A HREF=".//hammers.html">hammers
</A> on earth.
</BODY>
</HTML>
```

- بدون URL پایه

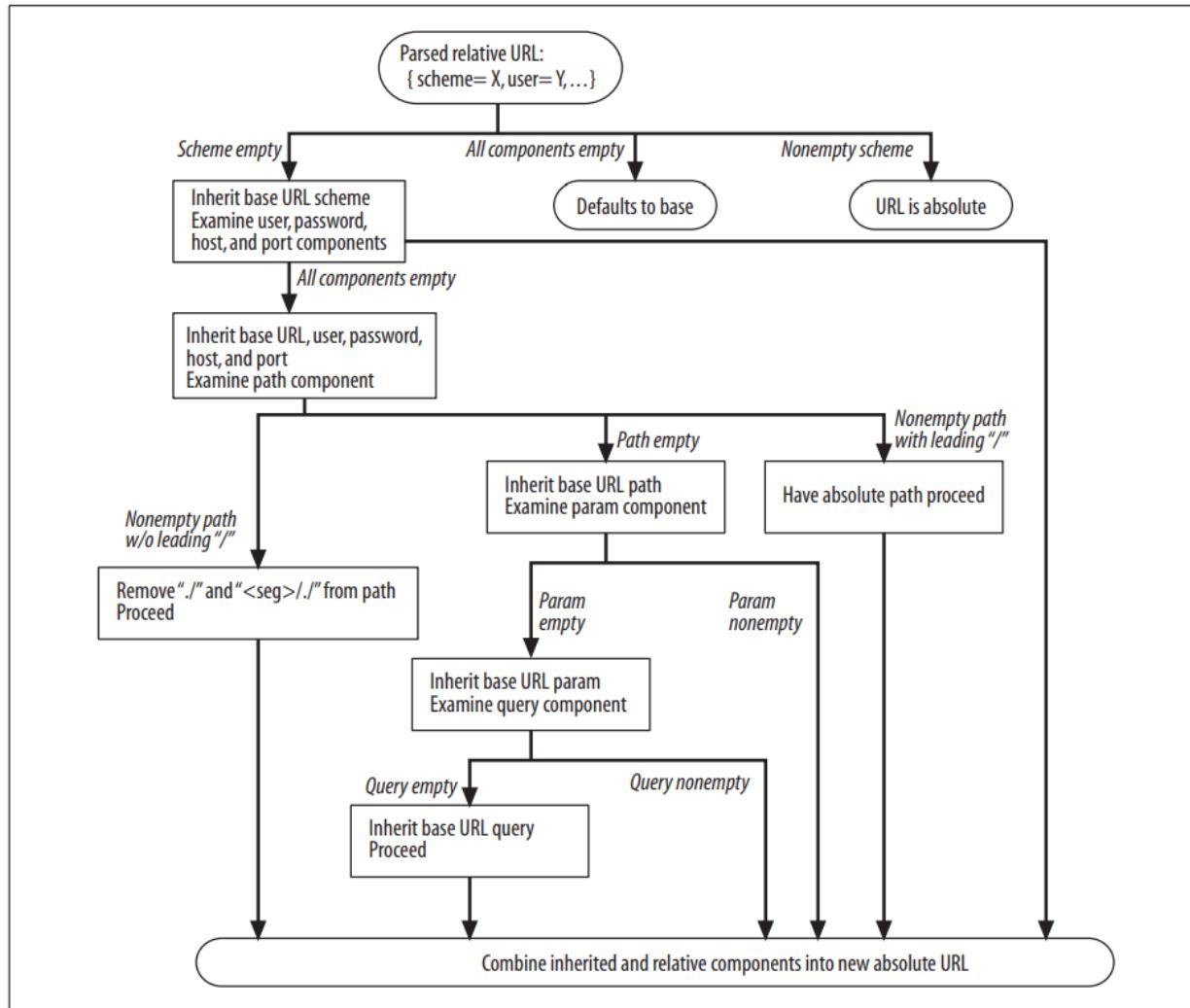
در برخی موارد، URL پایه وجود ندارد. این اغلب به این معنی است که شما یک URL مطلق دارید. با این حال، گاهی اوقات ممکن است شما فقط یک URL ناقص یا شکسته داشته باشید.

Resolving Relative References

قبل‌اً به اجزای اصلی و ساختار URL ها اشاره نموده ایم. گام بعدی در تبدیل یک URL نسبی به یک URL مطلق این است که هر دو URL نسبی و پایه را به قطعات تشکیل دهنده آن‌ها (مؤلفه‌ها) تقسیم کنید.

در واقع، شما فقط URL را تجزیه یا Parse می‌کنید، اما اغلب به آن URL Decomposing می‌گویند، زیرا شما URL را به اجزای تشکیل دهنده آن تقسیم می‌کنید.





این الگوریتم یک URL نسبی را به شکل مطلق آن تبدیل می‌کند که سپس می‌تواند برای ارجاع به منبع استفاده شود. این الگوریتم ابتدا در RFC 1808 مشخص شد و بعداً در RFC 2396 گنجانده شد.

با در نظر گرفتن `/hammers.html`. که در مثال پیشین به آن اشاره شد، می‌توانیم الگوریتم نشان داده شده در شکل بالا را اعمال کنیم:

در این مثال، Path یا مسیر `/hammers.html` است. همچنین URL پایه برابر با `hardware.com/tools.html` است.

در اینجا، URL Scheme خالی است. نیمه چپ نمودار را به سمت پایین ادامه دهید و در این صورت اصلی (HTTP) را به ارث خواهد برد.





حداقل یک جزء یا مؤلفه خالی نیست. در این صورت به پایین رفته و مؤلفه‌های Host و پورت به ارث برده خواهد شد.

با ترکیب اجزایی که از URL نسبی داریم (scheme: ./hammers.html) با آنچه که به ارث برده ایم (path: /hammers.html) مطلق URL (http, host: www.joes-hardware.com, port:80) را دریافت می‌کنیم که برابر آدرس زیر است:

<http://www.joes-hardware.com/hammers.html>

Expandomatic URLs

برخی از مرورگرها سعی می‌کنند URL ها را به طور خودکار گسترش دهند (چه پس از ارسال URL یا در حین تایپ). این قابلیت، یک میانبر برای کاربران بوده و آن‌ها مجبور نیستند URL را به صورت کامل تایپ کنند، زیرا این کار به طور خودکار انجام می‌شود.

این ویژگی که با عنوان Expandomatic شناخته می‌شود، در دو نوع ظاهر می‌شوند:

Hostname expansion •

در این نوع، مرورگر اغلب می‌تواند نام میزبانی را که تایپ می‌کنید بدون کمک شما، فقط با استفاده از چند روش اکتشافی ساده، به نام میزبان کامل گسترش دهد. به عنوان مثال، اگر در کادر آدرس "yahoo" را تایپ کنید، مرورگر شما می‌تواند به طور خودکار ".com" و "www.yahoo". را به ابتدای آن اضافه نماید. مرورگرها از این ترفندهای ساده برای صرفه جویی در زمان و ناممی‌دی شما استفاده می‌کنند.

با این حال، این ترفندهای توسعه در Hostname می‌تواند برای سایر برنامه‌های HTTP مانند پراکسی‌ها مشکل ایجاد کند.

History expansion •

تکنیک دیگری که مرورگرها برای صرفه جویی در وقت شما در تایپ URL استفاده می‌کنند، ذخیره History مربوط به URL هایی است که در گذشته بازدید کرده اید. همانطور که URL را تایپ می‌کنید، آن‌ها می‌توانند با تطبیق آنچه که تایپ می‌کنید با پیشوند URL های موجود در History خود، گزینه‌های تکمیل شده را به شما پیشنهاد دهند.





بنابراین، اگر شروع یک URL را که قبلاً از آن بازدید کرده بودید، مانند <http://www.joes.com>، را تایپ کنید، مرورگر شما می‌تواند <http://www.joes-hardware.com> را به شما پیشنهاد کند. سپس می‌توانید به جای تایپ کردن URL کامل، آن را انتخاب کنید.

توجه داشته باشید که Expansion خودکار URL ممکن است هنگام استفاده از پراکسی‌ها رفتار متفاوتی داشته باشد.

Shady Characters

URL‌ها به گونه‌ای طراحی شده اند که قابل حمل یا Portable باشند. آن‌ها همچنین برای نامگذاری یکسان همه منابع موجود در اینترنت طراحی شده اند، به این معنی که آن‌ها از طریق پروتکل‌های مختلف منتقل می‌شوند. از آنجایی که همه این پروتکل‌ها، مکانیسم‌های متفاوتی برای انتقال داده‌های خود دارند، طراحی URL‌ها به گونه‌ای مهم بود که بتوانند از طریق هر پروتکل اینترنتی به صورت ایمن منتقل شوند.

انتقال ایمن به این معنی است که URL‌ها می‌توانند بدون خطر از دست دادن اطلاعات منتقل شوند. برخی از پروتکل‌ها، مانند پروتکل SMTP برای پست الکترونیکی، از روش‌های انتقال استفاده می‌کنند که می‌توانند کarakترهای خاصی را حذف کنند. برای دور زدن این موضوع، URL‌هایی مجاز هستند فقط شامل کarakترهایی از الفبای کوچک و ایمن جهانی (universally safe alphabet) باشند.

علاوه بر اینکه طراحان می‌خواستند URL‌ها توسط همه پروتکل‌های اینترنتی قابل انتقال باشند، خوانایی آن‌ها برای افراد نیز اهمیت داشت. بنابراین، کarakترهای نامرئی (invisible) و غیرچاپی (nonprinting) نیز در URL‌ها ممنوع هستند، حتی اگر این کarakترها از طریق ایمیل عبور کنند و در غیر این صورت قابل حمل باشند.

کarakترهای غیرچاپی شامل whitespace هستند (توجه داشته باشید که RFC 2396 توصیه می‌کند که برنامه‌ها whitespace را نادیده بگیرند).

در ادامه، URL‌ها باید کامل می‌شدند. طراحان URL متوجه شدند که در موقعی، مردم می‌خواهند URL‌ها حاوی داده‌های باینری یا کarakترهایی خارج از الفبای ایمن جهانی باشند. بنابراین، یک مکانیسم فرار یا Escape اضافه گردید که به کarakترهای نامن اجازه می‌داد تا برای حمل و نقل به کarakترهای ایمن Encode شوند.

The URL Character Set

مجموعه کarakترهای پیش فرض سیستم کامپیوتری اغلب دارای یک سوگیری Anglocentric هستند. از لحاظ تاریخی، بسیاری از برنامه‌های کامپیوتری از مجموعه کarakترهای US-ASCII استفاده کرده اند.





US-ASCII از ۷ بیت برای نمایش بیشتر کلیدهای موجود در ماشین تحریر انگلیسی و چند کاراکتر کنترلی غیرچاپی برای قالب بندی متن و سیگنال دهی سخت افزاری استفاده می‌کند.

US-ASCII به دلیل میراث طولانی آن بسیار قابل حمل است. اما در حالی که برای شهروندان ایالات متحده راحت است، از کاراکترهای رایج در زبان‌های اروپایی یا صدها زبان غیر رومی که توسط میلیاردها نفر در سراسر جهان خوانده می‌شود، پشتیبانی نمی‌کند.

علاوه بر این، برخی از URL‌ها ممکن است نیاز داشته باشند که حاوی داده‌های باینری دلخواه باشند. با درک نیاز به کامل بودن، طراحان URL ویژگی Encoding Sequence را اضافه نمودند. این ویژگی امکان مقادیر دلخواه یا داده‌های کاراکتر را با استفاده از یک زیرمجموعه محدود از مجموعه کاراکترهای US-ASCII فراهم می‌کند که قابلیت حمل و کامل بودن را به همراه دارد.

Encoding Mechanisms

برای دور زدن محدودیت‌های موجود در نمایش مجموعه کاراکتر ایمن، یک طرح Encoding برای نشان دادن کاراکترهایی که ایمن نیستند در URL ابداع شد. Encoding به سادگی، کاراکتر نامن را با یک نماد "Escape" نشان می‌دهد که شامل یک علامت درصد (%) و به دنبال آن دو رقم هگزا دسیمال است که نشان دهنده کد کاراکتر ASCII است.

جدول زیر نمونه‌ای از موارد مذکور را نشان می‌دهد:

Character	ASCII code	Example URL
~	126 (0x7E)	http://www.joes-hardware.com/%7Ejoe
SPACE	32 (0x20)	http://www.joes-hardware.com/more%20tools.html
%	37 (0x25)	http://www.joes-hardware.com/100%25satisfaction.html

Character Restrictions

تعدادی از کاراکتر با توجه به داشتن معنای خاص در داخل یک URL، رزرو شده‌اند. برخی دیگر نیز در مجموعه قابل چاپ US-ASCII تعریف شده نیستند. همچنین برخی دیگر نیز با توجه به اینکه در تعدادی از Gateway‌ها و پروتکل‌های اینترنت، منجر به بروز اشتباه می‌شوند، استفاده از آن‌ها منع شده است.

جدول زیر، کاراکترهایی را نشان می‌دهد که باید قبل از استفاده از آن‌ها برای هر چیزی غیر از اهداف رزرو شده در یک URL، کدگذاری شوند.





Character	Reservation/Restriction
%	Reserved as escape token for encoded characters
/	Reserved for delimiting splitting up path segments in the path component
.	Reserved in the path component
..	Reserved in the path component
#	Reserved as the fragment delimiter
?	Reserved as the query-string delimiter
;	Reserved as the params delimiter
:	Reserved to delimit the scheme, user/password, and host/port components
\$,+@&=	Reserved
{ } \ ^ ~ [] '	Restricted because of unsafe handling by various transport agents, such as gateways
< > "	Unsafe; should be encoded because these characters often have meaning outside the scope of the URL, such as delimiting the URL itself in a document (e.g., "http://www.joes-hardware.com")
0x00–0x1F, 0x7F	Restricted; characters within these hex ranges fall within the nonprintable section of the US-ASCII character set
> 0x7F	Restricted; characters whose hex values fall within this range do not fall within the 7-bit range of the US-ASCII character set

A Bit More

ممکن است تعجب کنید که چرا وقتی از کاراکترهای نامن استفاده می‌کنید هیچ اتفاق بدی نمی‌افتد. به عنوان مثال، می‌توانید از صفحه اصلی در آدرس زیر دیدن کنید:

<http://www.joes-hardware.com/~joe>

در این مثال، کاراکتر "~" رمزگذاری نشده است. برای برخی از پروتکل‌های انتقال این موضوع مشکل خاصی را ایجاد نخواهد کرد، اما برای توسعه دهندگان برنامه، عدم Encoding کاراکترهای نامن روش مناسبی به نظر نمی‌رسد.

بهترین کار این است که برنامه‌های سمت کلاینت، قبل از ارسال هر نشانی اینترنتی به برنامه‌ای دیگر، کاراکترهای نامن یا محدود شده را تبدیل کنند. هنگامی که همه کاراکترهای نامن URL شدنده، Encode به شکل متعارفی است که می‌تواند بین برنامه‌ها به اشتراک گذاشته شود.

برنامه اصلی که URL را از کاراکتر دریافت می‌کند برای تعیین اینکه کدام کاراکترها باید Encode شوند مناسب است. از آنجایی که هر مؤلفه از URL ممکن است کاراکترهای ایمن یا نامن خود را داشته باشد





و اینکه کدام کاراکترها ایمن یا نامن هستند به Scheme بستگی دارد، تنها برنامه‌ای که URL را از کاربر دریافت می‌کند، در موقعیتی است که تعیین کند چه چیزی باید Encode شود.

البته، روش دیگر که کمی افراطی به نظر می‌رسد این است که برنامه همه کاراکترها را Encode کند. در حالی که استفاده از این روش توصیه نمی‌شود، ولی هیچ قانون خاصی در برابر Encode نمودن کاراکترهایی که قبل ایمن در نظر گرفته می‌شوند وجود ندارد. با این حال، در عمل این می‌تواند منجر به رفتارهای خاصی شود، زیرا برخی از برنامه‌ها ممکن است فرض کنند که کاراکترهای ایمن کدگذاری نمی‌شوند.

گاهی اوقات نفوذگران، کاراکترهای اضافی را Encode می‌کنند تا برنامه‌های فیلترینگ وب و ابزارهای مشابه را دور بزنند. Encoding مؤلفه‌های URL ایمن می‌تواند باعث شود برنامه‌های Pattern-Matching نتوانند الگوهایی مورد نظر را تشخیص دهند. به طور کلی، برنامه‌هایی که URL ها را تفسیر می‌کنند، باید URL ها را قبل از پردازش Decode نمایند.

برخی از مؤلفه‌های URL، مانند Scheme، باید به راحتی شناسایی شوند و لازم است که با یک کاراکتر الفبایی شروع شوند. به منظور مشاهده دستورالعمل‌های بیشتر در مورد استفاده از کاراکترهای رزرو شده و نامن در اجزای مختلف URL، به قسمت Syntax URL در همین بخش مراجعه نمایید.

A Sea of Schemes

در این بخش، نگاهی به فرمتهای رایج تر Scheme در وب خواهیم داشت. جدول زیر برخی از محبوب ترین Scheme ها را نشان می‌دهد. مرور بخش URL Syntax می‌تواند به درک بهتر این جدول به شما کمک نماید.



Scheme	Description
http	<p>The Hypertext Transfer Protocol scheme conforms to the general URL format, except that there is no username or password. The port defaults to 80 if omitted.</p> <p>Basic form:</p> <p><i>http://<host>:<port>/<path>?<query>#<frag></i></p> <p>Examples:</p> <p><i>http://www.joes-hardware.com/index.html</i> <i>http://www.joes-hardware.com:80/index.html</i></p>
https	<p>The https scheme is a twin to the http scheme. The only difference is that the https scheme uses Netscape's Secure Sockets Layer (SSL), which provides end-to-end encryption of HTTP connections. Its syntax is identical to that of HTTP, with a default port of 443.</p> <p>Basic form:</p> <p><i>https://<host>:<port>/<path>?<query>#<frag></i></p> <p>Example:</p> <p><i>https://www.joes-hardware.com/secure.html</i></p>
mailto	<p>Mailto URLs refer to email addresses. Because email behaves differently from other schemes (it does not refer to objects that can be accessed directly), the format of a mailto URL differs from that of the standard URL. The syntax for Internet email addresses is documented in Internet RFC 822.</p> <p>Basic form:</p> <p><i>mailto:<RFC-822-addr-spec></i></p> <p>Example:</p> <p><i>mailto:joe@joes-hardware.com</i></p>





Scheme	Description
ftp	<p>File Transfer Protocol URLs can be used to download and upload files on an FTP server and to obtain listings of the contents of a directory structure on an FTP server.</p> <p>FTP has been around since before the advent of the Web and URLs. Web applications have assimilated FTP as a data-access scheme. The URL syntax follows the general form.</p> <p>Basic form:</p> <p><i>ftp://<user>:<password>@<host>:<port>/<path>;<params></i></p> <p>Example:</p> <p><i>ftp://anonymous:joe%40joes-hardware.com@prep.ai.mit.edu:21/pub/gnu/</i></p>
rtsp, rtspu	<p>RTSP URLs are identifiers for audio and video media resources that can be retrieved through the Real Time Streaming Protocol.</p> <p>The "u" in the rtspu scheme denotes that the UDP protocol is used to retrieve the resource.</p> <p>Basic forms:</p> <p><i>rtsp://<user>:<password>@<host>:<port>/<path></i></p> <p><i>rtspu://<user>:<password>@<host>:<port>/<path></i></p> <p>Example:</p> <p><i>rtsp://www.joes-hardware.com:554/interview/cto_video</i></p>
file	<p>The file scheme denotes files directly accessible on a given host machine (by local disk, a network filesystem, or some other file-sharing system). The fields follow the general form. If the host is omitted, it defaults to the local host from which the URL is being used.</p> <p>Basic form:</p> <p><i>file://<host>/<path></i></p> <p>Example:</p> <p><i>file:///OFFICE-FS/policies/casual-fridays.doc</i></p>
news	<p>The news scheme is used to access specific articles or newsgroups, as defined by RFC 1036. It has the unusual property that a news URL in itself does not contain sufficient information to locate the resource.</p> <p>The news URL is missing information about where to acquire the resource—no hostname or machine name is supplied. It is the interpreting application's job to acquire this information from the user. For example, in your Netscape browser, under the Options menu, you can specify your NNTP (news) server. This tells your browser what server to use when it has a news URL.</p> <p>News resources can be accessed from multiple servers. They are said to be location-independent, as they are not dependent on any one source for access.</p> <p>The "@" character is reserved within a news URL and is used to distinguish between news URLs that refer to newsgroups and news URLs that refer to specific news articles.</p> <p>Basic forms:</p> <p><i>news:<newsgroup></i></p> <p><i>news:<news-article-id></i></p> <p>Example:</p> <p><i>news:rec.arts.startrek</i></p>





Scheme	Description
telnet	The telnet scheme is used to access interactive services. It does not represent an object per se, but an interactive application (resource) accessible via the telnet protocol. Basic form: <code>telnet://<user>:<password>@<host>:<port>/</code> Example: <code>telnet://slurp:webhound@joes-hardware.com:23/</code>

The Future

URL ها ابزار قدرتمندی هستند. طراحی URL ها به آنها اجازه می‌دهد تا تمام اشیاء موجود را نامگذاری کنند و به راحتی فرمتهای جدید را در بر گیرند. آنها مکانیسم نامگذاری یکنواختی را ارائه می‌دهند که می‌تواند بین پروتکل‌های اینترنت به اشتراک گذاشته شود.

با این حال، URL ها کامل نبوده و در واقع آدرس هستند، نه نامهای واقعی. این بدان معناست که یک URL به شما می‌گوید در حال حاضر چه چیزی در کجا قرار دارد.

چیزی که ایده آل خواهد بود این است که شما نام واقعی یک شی را داشته باشید، که می‌توانید از آن برای جستجوی آن شی صرف نظر از مکان آن استفاده کنید. مانند یک شخص، با توجه به نام منبع و چند واقعیت دیگر، می‌توانید آن منبع را بدون توجه به جایی که جابجا شده است، ردیابی کنید.

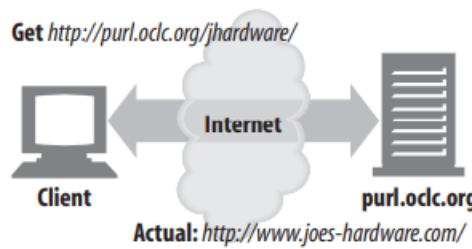
Internet Engineering Task Force (IETF) یا IETF مدتی است که روی استاندارد جدید و نامهای یکسان منابع (URN) کار می‌کند تا فقط به این موضوع رسیدگی کند. URN ها یک نام پایدار برای یک شی ارائه می‌کنند، صرف نظر از اینکه آن شی در کجا حرکت می‌کند (چه در داخل یک وب سرور یا در بین سرورهای وب).

PURL یا Persistent uniform resource locators نمونه‌ای از چگونگی دستیابی به عملکرد URN با استفاده از URL ها هستند. یک کلاینت می‌تواند یک Locator را از Persistent URL درخواست کند و سپس می‌تواند با منبعی پاسخ دهد که کلاینت را به URL واقعی و فعلی منبع هدایت می‌کند (شکل زیر را ببینید). برای اطلاعات بیشتر در مورد PURL ها، به <http://purl.oclc.org> مراجعه کنید.

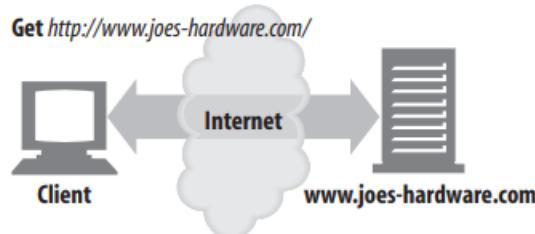




STEP 1: Ask the resource resolver what the Joe's Hardware URL is. Receive from the resolver the current location of the resource.



STEP 2: Get the actual URL for the resource



If Not Now, When

ایدههای پشت URN ها از مدت‌ها پیش وجود داشته است. در واقع، اگر به تاریخ انتشار برخی از مشخصات آن‌ها نگاه کنید، ممکن است از خود بپرسید که چرا آن‌ها هنوز به تصویب نرسیده‌اند.

تغییر از URL ها به URN ها کار بسیار بزرگی است. استانداردسازی در کل، فرآیندی کند است که اغلب دلایل خاص خود را دارد. پستیبانی از URN ها به تغییرات زیادی نیاز دارد که نمونه‌ای از آن‌ها شامل اجماع از طرف نهادهای استاندارد، تغییرات در برنامه‌های مختلف HTTP و غیره می‌باشد. بنابراین مدتی طول خواهد کشید تا همه عاملان همسو شوند تا چنین تبدیلی ممکن شود.

در طول رشد انفحاری وب، به کاربران اینترنت (از دانشمندان کامپیوتر گرفته تا کاربران عادی اینترنت) استفاده از URL ها آموزش داده شده است. URL ها محدودیت‌هایی دارند، اما این مسائل، مهم ترین مشکل جامعه توسعه وب نیستند.

در حال حاضر و برای آینده، URL ها راهی برای نامگذاری منابع در اینترنت هستند. آن‌ها همه جا هستند و ثابت کرده‌اند که بخش بسیار مهمی از موفقیت وب هستند. مدتی طول می‌کشد تا هر طرح نامگذاری دیگری URL ها را از بین ببرد. با این حال، URL ها محدودیت‌هایی خود را دارند و این احتمال وجود دارد که استانداردهای جدیدی (احتمالاً URN) ظاهر شوند و برای رفع برخی از این محدودیت‌ها به کار گرفته شوند.





For More Information

جهت کسب اطلاعات بیشتر، مراجعه به لینک‌های زیر توصیه می‌شود:

<http://www.w3.org/Addressing/>

<http://www.ietf.org/rfc/rfc1738>

<http://www.ietf.org/rfc/rfc2396.txt>

<http://www.ietf.org/rfc/rfc2141.txt>

<http://purl.oclc.org>

<http://www.ietf.org/rfc/rfc1808.txt>





فصل سوم - HTTP Messages

اگر HTTP پیک اینترنت باشد، HTTP Message ها بسته‌هایی هستند که برای جابجایی محتویات استفاده می‌شوند. در بخش اول نشان دادیم که چگونه برنامه‌های HTTP برای انجام کار به یکدیگر پیام ارسال می‌کنند. این فصل در مورد پیام‌های HTTP بوده و به چگونگی ایجاد آن‌ها می‌پردازد. در این فصل، به آنچه برای نوشتن برنامه‌های HTTP نیاز است، خواهیم پرداخت. مواردی موجود در این بخش عبارتند از:

- پیام‌ها چگونه جریان می‌یابند.
- مباحث مربوط به سه بخش پیام‌های HTTP (خط شروع، هدرها و بخش‌های Body).
- تفاوت بین پیام‌های درخواست و پاسخ.
- توابع مختلف (Methods) که پیام‌های درخواست پشتیبانی می‌کنند.
- کدهای وضعیت مختلف که با پیام‌های پاسخ بازگردانده می‌شوند.
- کاری که هدرهای مختلف HTTP انجام می‌دهند.

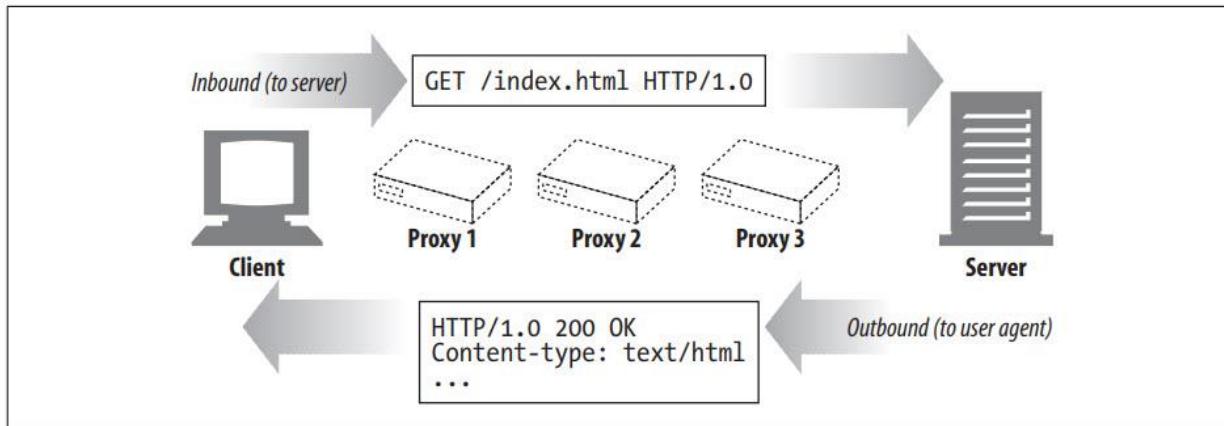
The Flow of Messages

پیام‌های HTTP بلوک‌های داده‌ای هستند که بین برنامه‌های HTTP ارسال می‌شوند. این بلوک‌های داده با برخی meta-information متنی که محتویات و معنی پیام را توصیف می‌کنند و سپس داده‌های اختیاری آغاز می‌شوند. این پیام‌ها بین کلاینت‌ها، سرورها و پروکسی‌ها جریان دارند. اصطلاحات "inbound"، "downstream" و "upstream" و "outbound" جهت پیام را توصیف می‌کنند.

Messages Commute Inbound to the Origin Server

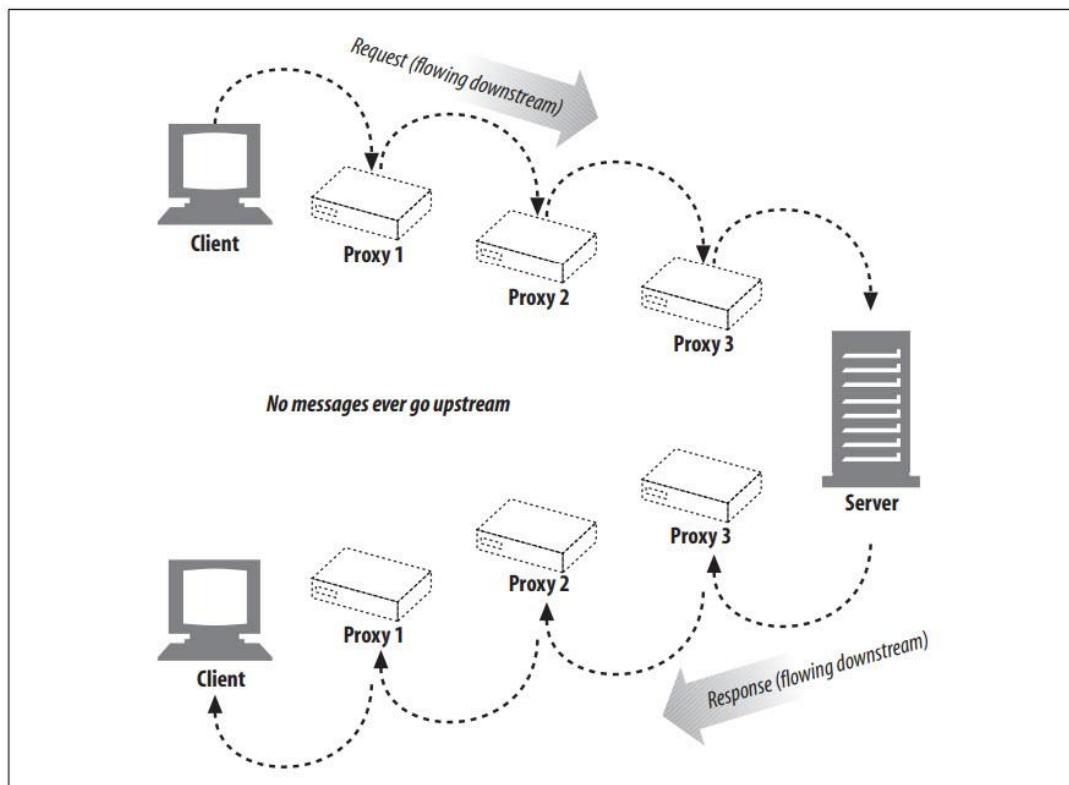
همانطور که اشاره شد، HTTP از اصطلاحات inbound و outbound برای توصیف جهت تراکنش استفاده می‌کند. پیام‌ها به سمت سرور مبدا به سمت inbound می‌روند، و هنگامی که کارشان تمام شد، به سمت عامل کاربر بر می‌گردند (شکل زیر را ببینید).





Messages Flow Downstream

پیام‌های HTTP مانند رودخانه‌ها جریان دارند. همه پیام‌ها بدون توجه به اینکه پیام‌های درخواست هستند یا پیام‌های پاسخ، به سمت پایین جریان می‌یابند (شکل زیر را ببینید). فرستنده هر پیامی در بالادست گیرنده است. در شکل زیر، پروکسی ۱ در **upstream** پروکسی ۳ برای درخواست است اما **downstream** پراکسی ۳ برای پاسخ است.





The Parts of a Message

پیام‌های HTTP بلوک‌های ساده و قالب بندی شده‌ای از داده‌ها هستند. هر پیام شامل یک درخواست از یک کلاینت یا یک پاسخ از یک سرور است. آن‌ها از سه بخش تشکیل شده‌اند:

- یک start line که پیام را توصیف می‌کند
- یک بلوک از هدرها حاوی attribute ها
- یک body اختیاری حاوی داده ها

start line و هدرها فقط متن ASCII هستند که با خطوط از هم جدا می‌شوند. هر خط با یک دنباله پایان خط دو کاراکتری، مشتمل از یک Carriage Return (ASCII 13) و یک کاراکتر Line-Feed (ASCII 10) به پایان می‌رسد. این دنباله انتهای خط در واقع همان "CRLF" است. شایان ذکر است در حالی که مشخصات HTTP برای خطوط پایان دهنده، CRLF می‌باشد، برنامه‌های کاربردی باید فقط یک کاراکتر Line-Feed را نیز بپذیرند. برخی از برنامه‌های HTTP قدیمی یا خراب همیشه هم Carriage Return و هم Line-Feed را ارسال نمی‌کنند.

Start یا بدن پیام (یا فقط همان body) به سادگی یک تکه اختیاری از داده است. برخلاف Line و هدرها، بدن می‌تواند حاوی متن یا داده‌های باینری بوده و یا خالی باشد.

در مثال شکل زیر، هدرها اطلاعات کمی در مورد بدن به شما می‌دهند. خط Content-Type به شما می‌گوید که نوع محتوای موجود در بدن به چه صورت است که در این مثال، یک سند متن ساده می‌باشد. خط Content-Length به شما می‌گوید که سایز بدن چقدر است که در اینجا ۱۹ بایت می‌باشد.



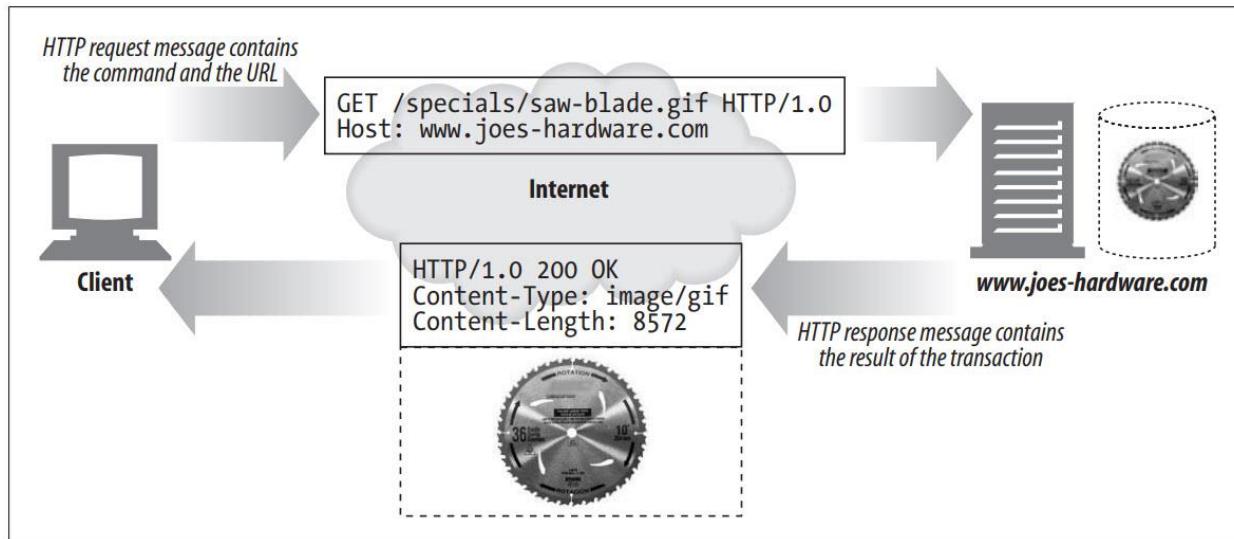
Message Syntax

همه پیام‌های HTTP به دو نوع تقسیم می‌شوند: پیام‌های درخواستی (Request Message) و پیام‌های پاسخ (Response Message).



پیام‌های درخواستی، اقدامی را از سرور وب درخواست می‌کنند. پیام‌های پاسخ، نتایج یک درخواست را به مشتری برمی‌گرداند. هر دو پیام درخواست و پاسخ ساختار پیام اصلی یکسانی دارند.

شکل زیر پیام‌های درخواست و پاسخ را برای دریافت یک تصویر GIF نشان می‌دهد.



فرمت پیام درخواست به شکل زیر است:

```
<method> <request-URL> <version>
<headers>

<entity-body>
```

فرمت پیام پاسخ نیز به شکل زیر است (توجه داشته باشید که Syntax، فقط در خط شروع متفاوت است):

```
<version> <status> <reason-phrase>
<headers>

<entity-body>
```

در اینجا توضیح مختصری از بخش‌های مختلف آورده شده است:

Method •

عملی است که مشتری می‌خواهد سرور روی منبع انجام دهد. این بخش شامل یک کلمه واحد است، مانند GET، POST یا HEAD.

request-URL •





یک URL کامل که منبع درخواستی یا مؤلفه path مربوط به URL را نامگذاری می‌کند.

Version •

نسخه HTTP که پیام از آن استفاده می‌کند. فرمت آن مشابه نمونه زیر است:

HTTP/<major>.<minor>

هر دو اعداد صحیح هستند. minor و major

status-code •

یک عدد سه رقمی که توصیف کننده اتفاقاتی است که در طول درخواست رخ داده است. رقم اول هر کد، کلاس عمومی وضعیت (success ، error و غیره) را توصیف می‌کند. فهرست کاملی از کدهای وضعیت تعریف شده در مشخصات HTTP و معانی آن‌ها در ادامه این فصل ارائه می‌شود.

reason-phrase •

یک نسخه قابل خواندن برای انسان از کد وضعیت عددی است. reason-phrase صرفاً برای انسان در نظر گرفته شده است، بنابراین، برای مثال، خطوط پاسخ حاوی «HTTP/1.0 200 NOT OK» و «HTTP/1.0 200 OK» باید به عنوان نشانه‌های موفقیت معادل تلقی شوند، علیرغم اینکه reason-phrase خلاف آن را نشان می‌دهند.

Headers •

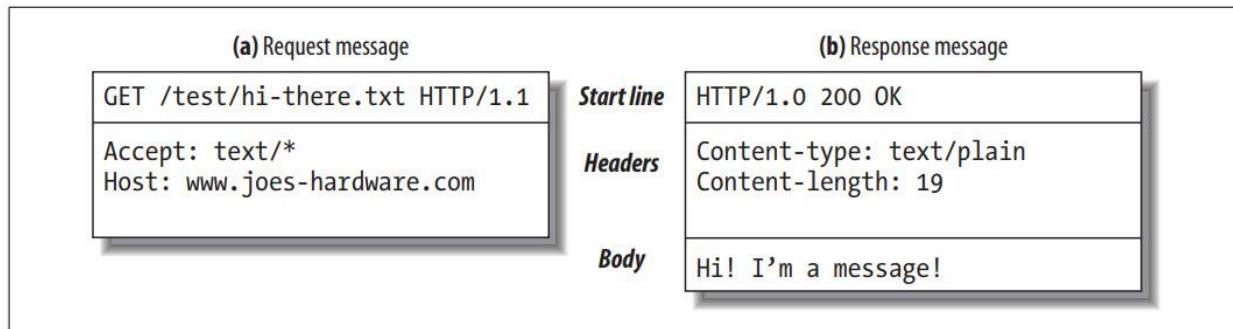
بخش‌های هدر، شامل یک نام به همراه دو نقطه (:) و مقدار مربوط به هر بخش هستند. همچنین در انتهای هر خط نیز یک CRLF قرار دارد. هدرها با یک خط خالی (CRLF) خاتمه می‌یابند که انتهای لیست هدرها و ابتدای Body را مشخص می‌کند. برخی از نسخه‌های HTTP، مانند HTTP/1.1، برای معتبر بودن پیام درخواست یا پاسخ، به هدرهای خاصی نیاز دارند.

entity-body •

entity body شامل بلوکی از داده‌های دلخواه است. همه پیام‌ها حاوی entity body نیستند، بنابراین گاهی اوقات یک پیام با یک CRLF خالی خاتمه می‌یابد.

شکل زیر پیام‌های درخواست و پاسخ فرضی را نشان می‌دهد.





توجه داشته باشید که مجموعه‌ای از هدرهای HTTP همیشه باید به یک خط خالی (CRLF) ختم شود، حتی اگر هیچ هدر و entity body‌ای وجود نداشته باشد. با این حال، از نظر تاریخی، بسیاری از کلاینت‌ها و سرورها (به اشتباہ) CRLF نهایی را در صورتی که عدم وجود entity body، حذف کردند. برای تعامل با این پیاده سازی‌های محبوب اما ناسازگار، کلاینت‌ها و سرورها باید پیام‌هایی را بپذیرند که بدون CRLF نهایی پایان می‌یابند.

Start Lines

همه پیام‌های HTTP با یک start line شروع می‌شوند. پیام درخواست، می‌گوید که چه کاری باید انجام شود. start line پیام پاسخ می‌گوید چه اتفاقی افتاده است.

Request line

پیام‌های درخواستی از سرورها می‌خواهند که کاری برای یک منبع انجام دهند. start line برای یک پیام درخواست یا خط درخواست، حاوی متده است که عملیاتی را که باید روی سرور انجام شود مشخص نموده و همچنین شامل یک URL است. start line همچنین شامل نسخه HTTP نیز می‌باشد که به سرور می‌گوید کلاینت با چه ساختار HTTP صحبت می‌کند.

همه این فیلدها با space از هم جدا می‌شوند. در بخش a از تصویر پیشین، متده درخواست درخواست /test/hi-there.txt HTTP/1.1 و نسخه HTTP/1.0 است. لازم به ذکر است که قبل از خطوط درخواست شامل نسخه HTTP/1.0 نبود.

Response line

پیام‌های پاسخ، اطلاعات وضعیت و هرگونه داده‌های حاصل از عملیات را به کلاینت منتقل می‌کنند. start line پیام پاسخ یا خط پاسخ شامل نسخه HTTP است که پیام پاسخ از آن استفاده می‌کند، یک کد وضعیت عددی و یک عبارت دلیل (reason) متنی که وضعیت عملیات را توصیف می‌کند.





همه این فیلد ها با فضای خالی از هم جدا می شوند. در بخش b شکل پیشین، نسخه HTTP برابر 0.1 است. کد وضعیت 200 (نشان دهنده موفقیت) و عبارت دلیل، OK است، به این معنی که سند با موفقیت بازگردانده شده است. قبل از HTTP/1.0، پاسخ ها نیازی به یک خط پاسخ نداشتند.

Methods

متد مشخص شده در start line درخواستها به سرور می گوید که چه کاری انجام دهد. به عنوان مثال، در خط "GET /specials/saw-blade.gif HTTP/1.0" متد GET است.

مشخصات HTTP مجموعه ای از متدهای درخواست رایج را تعریف کرده است. به عنوان مثال، متد GET یک سند را از یک سرور دریافت می کند، متد POST داده ها را برای پردازش به سرور ارسال می کند، و متد OPTIONS قابلیت های کلی یک وب سرور را برای یک منبع خاص تعیین می کند.

جدول زیر هفت مورد از این متدها را شرح می دهد. توجه داشته باشید که برخی از متدها دارای یک بدن هستند. پیام درخواست هستند و متدهای دیگر دارای درخواست های بدون بدن هستند.

Method	Description	Message body?
GET	Get a document from the server.	No
HEAD	Get just the headers for a document from the server.	No
POST	Send data to the server for processing.	Yes
PUT	Store the body of the request on the server.	Yes
TRACE	Trace the message through proxy servers to the server.	No
OPTIONS	Determine what methods can operate on a server.	No
DELETE	Remove a document from the server.	No

همه سرورها تمام هفت متد مشخص شده در جدول بالا را پیاده سازی نمی کنند. علاوه بر این، از آنجایی که HTTP به گونه ای طراحی شده است که به راحتی قابل توسعه باشد، سرورهای دیگر ممکن است متدهای درخواست خود را علاوه بر اینها پیاده سازی کنند. به این متدهای اضافی، extension methods یا متدهای توسعه می گویند، زیرا مشخصات HTTP را گسترش می دهند.

Status codes

همانطور که متدها به سرور می گویند چه کاری انجام دهد، کدهای وضعیت به مشتری می گویند چه اتفاقی افتاده است. کدهای وضعیت در خطوط ابتدایی پاسخ ها قرار دارند. به عنوان مثال، در خط "HTTP/1.0 200 OK" کد وضعیت 200 است.





هنگامی که کلاینت‌ها پیام‌های درخواستی را به سرور HTTP ارسال می‌کنند، ممکن است اتفاقات زیادی رخ دهد. اگر خوش شانس باشید، درخواست با موفقیت تکمیل خواهد شد. ممکن است همیشه آنقدر خوش شانس نباشد. سرور ممکن است به شما بگوید که منبعی که درخواست کرده‌اید یافت نشد، اجازه دسترسی به منبع را ندارید یا شاید منبع به جای دیگری منتقل شده است.

کدهای وضعیت در خط شروع هر پیام پاسخ، بازگردانده می‌شوند. هر دو وضعیت عددی و قابل خواندن توسط انسان برگردانده می‌شوند. کد عددی پردازش خطای برای برنامه‌ها آسان می‌کند، در حالی که عبارت دلیل (reason phrase) به راحتی توسط انسان قابل درک است.

کدهای وضعیت بر اساس کدهای عددی سه رقمی، در کلاس‌های مختلف، گروه بندی می‌شوند. کدهای وضعیت بین ۲۰۰ تا ۲۹۹ نشان دهنده موفقیت هستند. کدهای بین ۳۰۰ و ۳۹۹ نشان می‌دهد که منبع منتقل شده است. کدهای بین ۴۰۰ و ۴۹۹ به این معنی است که درخواست کلاینت اشتباه بوده است. کدهای بین ۵۰۰ و ۵۹۹ به این معنی است که مشکلی در سرور رخ داده است.

کلاس‌های کد وضعیت در جدول زیر نشان داده شده است.

Overall range	Defined range	Category
100-199	100-101	Informational
200-299	200-206	Successful
300-399	300-305	Redirection
400-499	400-415	Client error
500-599	500-505	Server error

نسخه‌های فعلی HTTP تنها چند کد برای هر دسته وضعیت تعریف می‌کنند. با تکامل پروتکل، کدهای وضعیت بیشتری به طور رسمی در مشخصات HTTP تعریف خواهند شد. اگر کد وضعیتی را دریافت کردید که نمی‌شناسید، به احتمال زیاد شخصی آن را به عنوان پسوند پروتکل فعلی تعریف کرده است. شما باید آن را به عنوان یک عضو کلی از کلاسی که در محدوده آن قرار می‌گیرد، در نظر بگیرید.

به عنوان مثال، اگر کد وضعیت 515 را دریافت کردید (که خارج از محدوده تعریف شده برای کدهای 5xx فهرست شده در جدول بالا است)، باید پاسخ را به عنوان نشان دهنده خطای سرور تلقی کنید که در کلاس کلی پیام‌های 5xx است.





جدول زیر برخی از رایج ترین کدهای وضعیت را که مشاهده خواهید کرد نمایش می‌دهد.

Status code	Reason phrase	Meaning
200	OK	Success! Any requested data is in the response body.
401	Unauthorized	You need to enter a username and password.
404	Not Found	The server cannot find a resource for the requested URL.

Reason Phrases

عبارت Reason Phrases یا عبارت دلیل آخرین جزء start line مربوط به پاسخ است. این بخش یک توضیح متنی از کد وضعیت ارائه می‌دهد. به عنوان مثال، در خط "HTTP/1.0 200 OK"، عبارت دلیل برابر OK است.

عبارات دلیل یک به یک با کدهای وضعیت جفت می‌شوند. عبارت دلیل یک نسخه قابل خواندن برای انسان از کد وضعیت ارائه می‌دهد که توسعه دهنده‌گان برنامه می‌توانند به کاربران خود منتقل کنند تا نشان دهند که در طول درخواست چه اتفاقی افتاده است.

Version Numbers

یا شماره نسخه در هر دو خط شروع پیام درخواست و پاسخ با فرمت `HTTP/x.y` ظاهر Version Numbers می‌شود. آن‌ها ابزاری را برای برنامه‌های HTTP فراهم می‌کنند تا به یکدیگر بگویند با چه نسخه‌ای از پروتکل مطابقت دارند.

با توجه به اینکه نسخه‌های مختلف HTTP دارای قابلیت‌ها و ویژگی‌های مختلفی هستند، یک برنامه HTTP نسخه ۱.۲ که با یک برنامه HTTP نسخه ۱.۱ ارتباط برقرار می‌کند باید بداند که Nebayid از هیچ ویژگی جدید ۱.۲ استفاده کند، زیرا احتمالاً توسط برنامه‌ای که نسخه قدیمی پروتکل را مورد استفاده قرار می‌دهد، اجرا نمی‌شود.

توجه داشته باشید که Version Numbers یا شماره نسخه، بالاترین نسخه HTTP را نشان می‌دهد که یک برنامه از آن پشتیبانی می‌کند.

همچنین Version Numbers به عنوان اعداد اعشاری در نظر گرفته نمی‌شوند. هر عدد در نسخه (به عنوان مثال، "۱" و ".۰" در `HTTP/1.0`) به عنوان یک عدد جداگانه در نظر گرفته می‌شود.

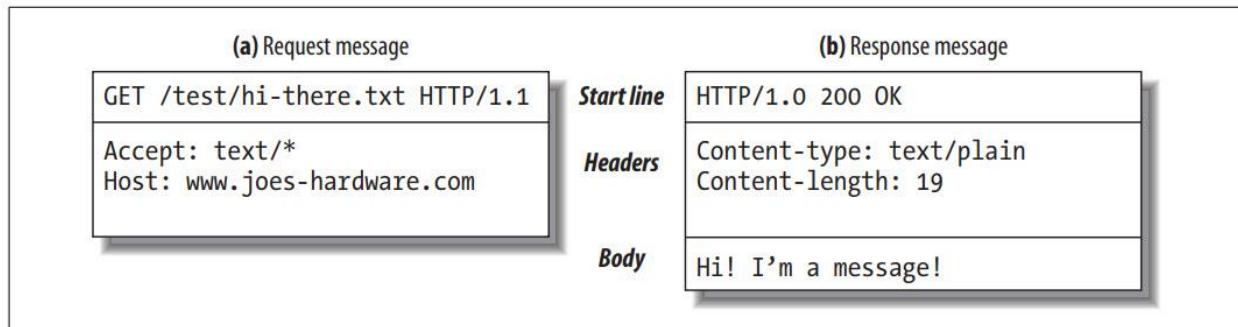
بنابراین، هنگام مقایسه نسخه‌های HTTP، هر عدد باید جداگانه مقایسه شود تا مشخص شود که کدام نسخه بالاتر است. به عنوان مثال، `HTTP/2.22` یک نسخه بالاتر از `HTTP/2.3` است، زیرا ۲۲ عددی بزرگتر از ۳ است.





Headers

بخش قبلی روی خط اول پیام‌های درخواست و پاسخ (متدها، کدهای وضعیت، عبارات دلیل و شماره نسخه) متمرکز بود. به دنبال خط شروع، لیستی از فیلدهای هدر HTTP آمده است که ممکن است درخواست بدون آن‌ها بوده و یا شامل برخی از آن‌ها شود (شکل زیر که قبلاً هم به آن اشاره شده بود را ببینید).



فیلدهای هدر HTTP اطلاعات بیشتری را به پیام‌های درخواست و پاسخ اضافه می‌کنند. آن‌ها اساساً لیستی از جفت name/value هستند. به عنوان مثال، خط هدر زیر مقدار ۱۹ را به فیلد سرصفحه اختصاص می‌دهد:

Content-length: 19

Header Classifications •

HTTP چندین فیلد هدر را تعریف می‌کند. برنامه‌ها همچنین امکان اضافه نمودن هدرهای مختص به خود را دارند. هدرهای HTTP به دسته‌های زیر تقسیم بندی می‌شوند:

: می‌تواند در هر دو پیام درخواست و پاسخ ظاهر شود. **General headers**

: اطلاعات بیشتری در مورد درخواست ارائه می‌دهد. **Request headers**

: اطلاعات بیشتری در مورد پاسخ ارائه می‌دهد. **Response headers**

: اندازه بدن، محتويات یا خود منبع را شرح می‌دهد. **Entity headers**

: این بخش شامل هدرهای جدیدی هستند که در ساختار، تعریف نشده‌اند. **Extension headers**

هر هدر HTTP یک ساختار ساده دارد: یک نام، به دنبال آن یک کولون (:)، به دنبال آن Space اختیاری، به دنبال آن مقدار فیلد، به دنبال آن یک CRLF. جدول زیر چند نمونه هدر را رایج را فهرست می‌کند:





Header example	Description
Date: Tue, 3 Oct 1997 02:16:03 GMT	The date the server generated the response
Content-length: 15040	The entity body contains 15,040 bytes of data
Content-type: image/gif	The entity body is a GIF image
Accept: image/gif, image/jpeg, text/html	The client accepts GIF and JPEG images and HTML

Header continuation lines •

خطوط هدر طولانی را می‌توان با شکستن آن‌ها به چند خط، قبل از هر خط اضافی با حداقل یک فاصله یا کاراکتر tab، خواناتر کرد. مثال زیر را در نظر بگیرید:

HTTP/1.0 200 OK

Content-Type: image/gif

Content-Length: 8572

Server: Test Server

Version 1.0

در این مثال، پیام پاسخ حاوی یک هدر سرور است که مقدار آن به دو خط تقسیم شده است. مقدار کامل هدر "Test Server Version 1.0" است.

Entity Bodies

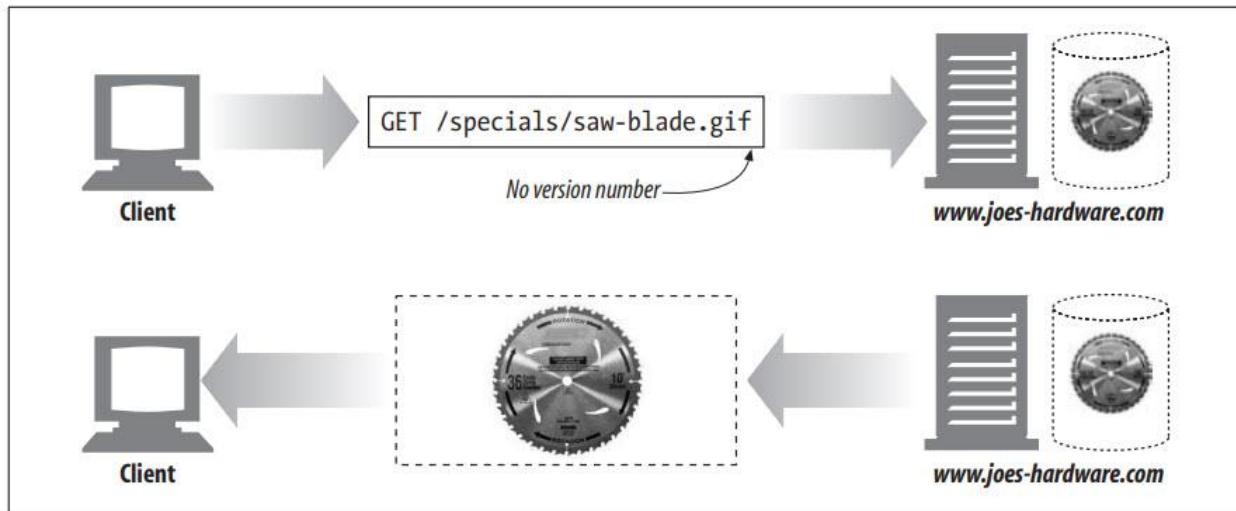
بخش سوم از پیام HTTP، Entity Bodies است که اختیاری می‌باشد. این بخش، پیلود پیام‌های HTTP بوده و مواردی هستند که HTTP برای انتقال آن‌ها طراحی شده است.

پیام‌های HTTP می‌توانند انواع مختلفی از داده‌های دیجیتال را حمل کنند: این موارد شامل تصاویر، ویدئو، اسناد HTML، اطلاعات برنامه‌های کاربردی، تراکنش‌های کارت اعتباری، پست الکترونیکی و غیره می‌باشند.

Version 0.9 Messages

HTTP نسخه 0.9 اولیه پروتکل HTTP بود. این نسخه، نقطه شروع برای پیام‌های درخواست و پاسخی است که HTTP امروزه از آن استفاده می‌کند، اما بسیار ساده‌تر از پروتکل HTTP امروزی است (شکل زیر را ببینید).





پیامهای HTTP/0.9 نیز شامل درخواست‌ها و پاسخ‌ها بودند، اما درخواست صرفاً شامل متدهای URL درخواست بود و پاسخ فقط شامل Entity بود. هیچ اطلاعات نسخه (اولین و تنها نسخه در آن زمان بود)، هیچ کد وضعیت یا عبارت دلیل و هیچ هدری در این نسخه از پروتکل گنجانده نشده بود.

با این حال، این سادگی، امکان انعطاف پذیری یا پیاده‌سازی اکثر ویژگی‌ها و برنامه‌های کاربردی HTTP که در این کتاب توضیح داده شده است را نمی‌دهد. ما در اینجا به اختصار آن را توضیح می‌دهیم زیرا هنوز کلاینت‌ها، سرورها و برنامه‌های کاربردی دیگری هستند که از آن استفاده می‌کنند و برنامه نویسان باید از محدودیت‌های آن آگاه باشند.

Methods

بیایید با جزئیات بیشتری در مورد برخی از متدهای اصلی HTTP که قبلًا به آن‌ها اشاره شده بود، صحبت کنیم. توجه داشته باشید که همه متدها توسط هر سروری پیاده‌سازی نمی‌شوند. برای سازگاری با HTTP نسخه 1.1 سرور فقط باید متدهای GET و HEAD را برای منابع خود پیاده‌سازی کند.

حتی زمانی که سرورها همه این متدها را پیاده‌سازی می‌کنند، متدها به احتمال زیاد کاربرد محدودی دارند. برای مثال، سرورهایی که از PUT یا DELETE (که در ادامه در این بخش توضیح داده می‌شود) پشتیبانی می‌کنند، نمی‌خواهند هر کسی بتواند منابع را حذف یا ذخیره کند. این محدودیت‌ها معمولاً در پیکربندی سرور تنظیم می‌شوند، بنابراین از سایت دیگر و از سروری به سرور دیگر متفاوت هستند.





Safe Methods

HTTP مجموعه ای از متدها را تعریف می کند که به آنها متدهای ایمن می گویند. گفته می شود که متدهای GET و HEAD ایمن هستند، به این معنی که هیچ اقدامی نباید در نتیجه درخواست HTTP که از روش HEAD یا استفاده می کند رخ دهد.

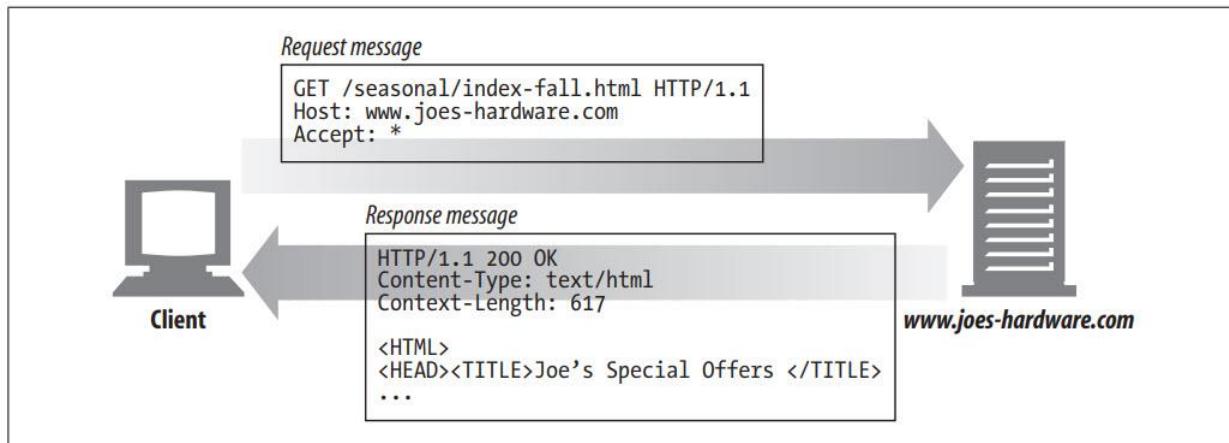
منظور ما از هیچ اقدامی این است که در نتیجه درخواست HTTP هیچ اتفاقی روی سرور رخ نخواهد داد. به عنوان مثال، زمانی را در نظر بگیرید که در یک سایت فروشگاه آنلاین اقدام به خرید می کنید و روی دکمه «خرید» کلیک می کنید. با کلیک بر روی این دکمه، یک درخواست POST (که بعداً در مورد آن بحث خواهد شد) همراه با اطلاعات کارت اعتباری شما ارسال می شود و یک Action از طرف شما در سرور انجام می شود. در این حالت، این است که کارت اعتباری شما برای خرید شما شارژ می شود.

هیچ تضمینی وجود ندارد که یک متدهای ایمن باعث انجام یک Action نشود (در این به توسعه دهندهان وب بستگی دارد). منظور از متدهای ایمن این است که به توسعه دهندهان برنامه های HTTP اجازه می دهند تا در صورت استفاده از یک متدهای ایمن که ممکن است باعث انجام برخی اقدامات شود، به کاربران اطلاع دهند. در مثال قبلی ما، مرورگر وب شما ممکن است یک پیام هشدار ظاهر شود که به شما اطلاع می دهد که درخواستی را با متدهای ایمن ارسال می کنید و در نتیجه ممکن است اتفاقی روی سرور بیفتد (مثالاً شارژ شدن کارت اعتباری شما).

GET

GET رایج ترین متده است. معمولاً برای درخواست از سرور برای ارسال یک منبع استفاده می شود. HTTP/1.1 به سرورها برای پیاده سازی این متده نیاز دارد. شکل زیر نمونه ای را نشان می دهد که کلاینت، درخواست HTTP را با متده GET انجام می دهد.

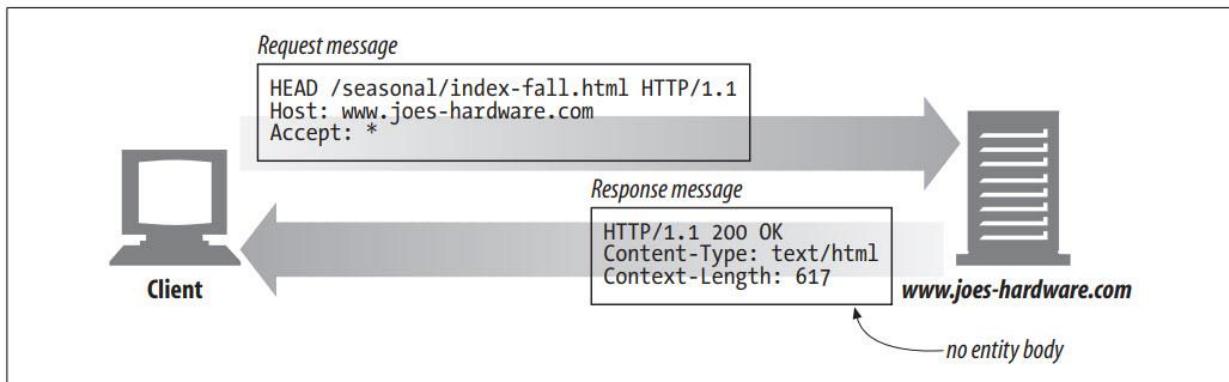


**HEAD**

متدهای HEAD و GET دقیقاً مانند هم کارند، اما سرور فقط هدرهای پاسخ را بر می‌گرداند. هیچ Entity هرگز بازگردانده نمی‌شود. این متدهای کلاینت اجازه می‌دهند بدون نیاز به دریافت منبع، هدرها را بازرسی کنند. با استفاده از HEAD می‌توانید:

- بدون دریافت منبع، اطلاعاتی در مورد یک منبع پیدا کنید (به عنوان مثال، نوع آن را تعیین کنید).
- با مشاهده کد وضعیت پاسخ، ببینید آیا یک شی وجود دارد یا خیر.
- با مشاهده هدرها، بررسی کنید که آیا منبع اصلاح شده است یا خیر.

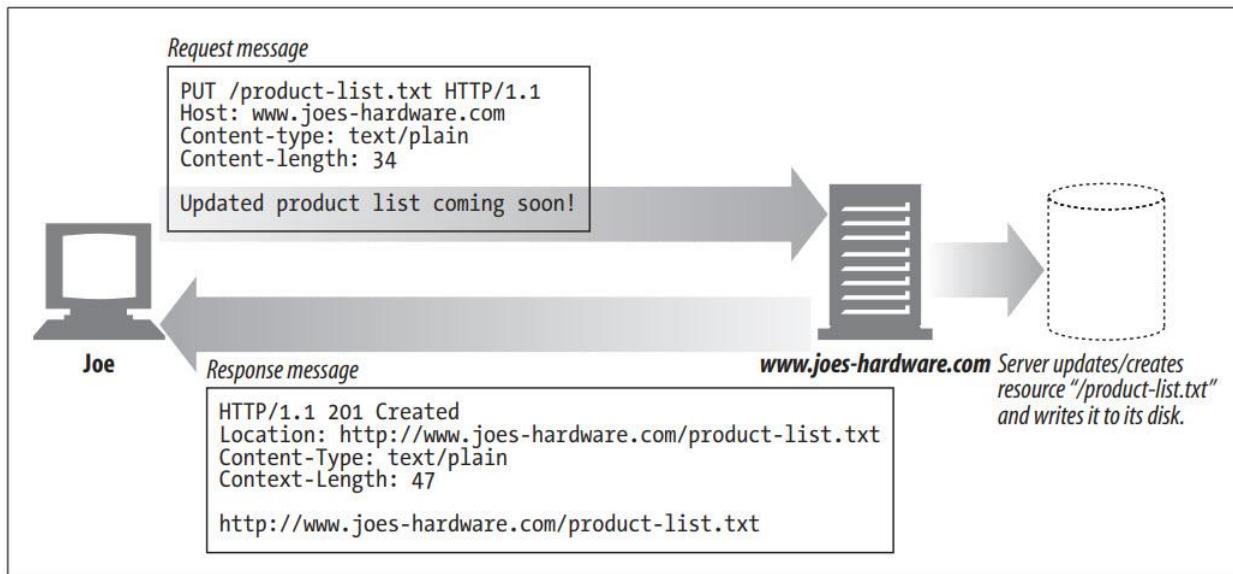
توسعه دهنده‌گان سرور باید اطمینان حاصل کنند که هدرهای بازگردانده شده دقیقاً همان هدرهایی هستند که درخواست HEAD بر می‌گرداند. متدهای HEAD به پیاده سازی پروتکل HTTP/1.1 نیاز دارند. شکل زیر متدهای HEAD را در عمل نشان می‌دهد.





PUT

متد PUT بر عکس متد GET که اسناد را از سرور می خواند، اسناد را روی سرور می نویسد. برخی از سیستم های Publishing به شما اجازه می دهند صفحات وب ایجاد کنید و آنها را مستقیماً با استفاده از PUT روی سرور وب نصب کنید (شکل زیر را ببینید).

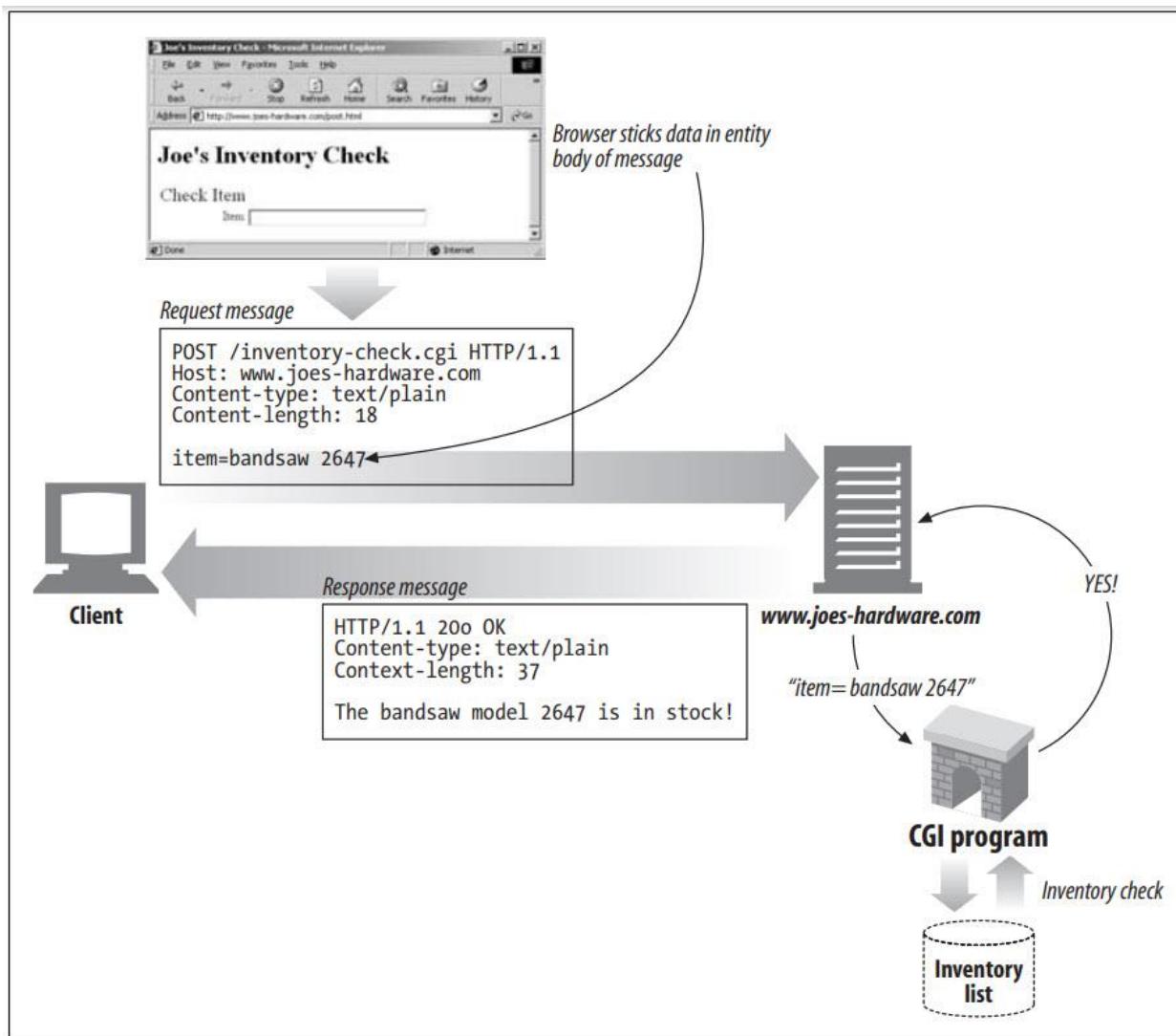


معنای متد PUT برای سرور این است که بدن درخواست را بگیرد و یا از آن برای ایجاد یک سند جدید با نام URL درخواستی استفاده کند یا اگر آن URL از قبل وجود داشته باشد، از بدن برای جایگزینی آن استفاده کند. از آنجایی که می تواند محتوا را تغییر دهد، بسیاری از سرورهای وب قبل از انجام PUT از شما می خواهند که با رمز عبور وارد شوید.

POST

متد POST برای ارسال داده های ورودی به سرور طراحی شده است. در عمل، اغلب از این متد برای پشتیبانی از فرم های HTML استفاده می شود. داده های یک فرم پر شده معمولاً به سرور ارسال می شود و سپس آنها را به جایی که باید بروд هدایت می کند. شکل زیر یک کلاینت را نشان می دهد که درخواست HTTP را بوسیله متد POST ارسال می کند.





TRACE

هنگامی که یک کلاینت درخواستی را ارائه می‌کند، ممکن است آن درخواست از فایروال‌ها، پروکسی‌ها، Gateway‌ها یا سایر برنامه‌ها عبور کند. هر یک از این‌ها این فرصت را دارد که درخواست اصلی HTTP را تغییر دهد. متدهای TRACE به کلاینت‌ها این امکان را می‌دهد که ببینند درخواست‌ها در نهایت که به سرور می‌رسند، چگونه به نظر می‌آیند.

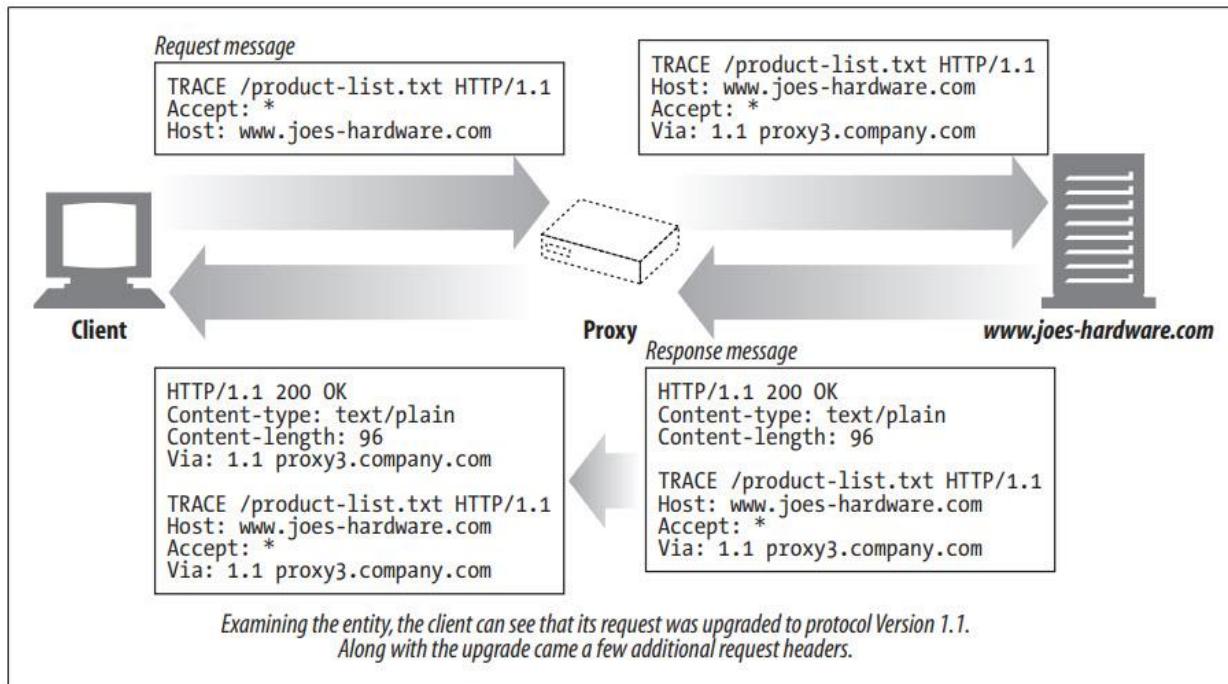
یک درخواست TRACE یک تشخیص "Loopback" را در سرور مقصد آغاز می‌کند.

سرور در آخرین مرحله سفر، یک پاسخ TRACE را با پیام درخواست دست نخورده‌ای که در متن پاسخ دریافت کرده است، باز می‌گرداند. سپس یک کلاینت می‌تواند ببیند که آیا پیام اصلی آن در امتداد





زنجیره درخواست/پاسخ، توسط برنامه‌های واسط HTTP تغییر یا اصلاح شده است یا خیر (شکل زیر را ببینید).



متد TRACE در درجه اول جهت تشخیص مورد استفاده قرار می‌گیرد. به عنوان مثال، تأیید اینکه درخواست‌ها طبق خواسته‌ما، از طریق زنجیره درخواست/پاسخ عبور می‌کنند. همچنین ابزار خوبی برای مشاهده تأثیرات پروکسی‌ها و سایر برنامه‌ها بر روی درخواست‌های شما است.

همانطور که TRACE برای تشخیص خوب است ولی مشکلاتی هم دارد.

بسیاری از برنامه‌های HTTP بسته به متد، کارهای متفاوتی انجام می‌دهند - برای مثال، یک پروکسی ممکن است یک درخواست POST را مستقیماً به سرور ارسال نموده، اما سعی کند یک درخواست GET را به یک برنامه HTTP دیگر (مانند یک کش وب) ارسال کند. TRACE مکانیسمی برای تشخیص متدها ارائه نمی‌دهد.

هیچ TRACE را نمی‌توان با درخواست Entity Body مربوط به پاسخ ارسال کرد. شامل درخواستی است که سرور پاسخ دهنده دریافت کرده است.

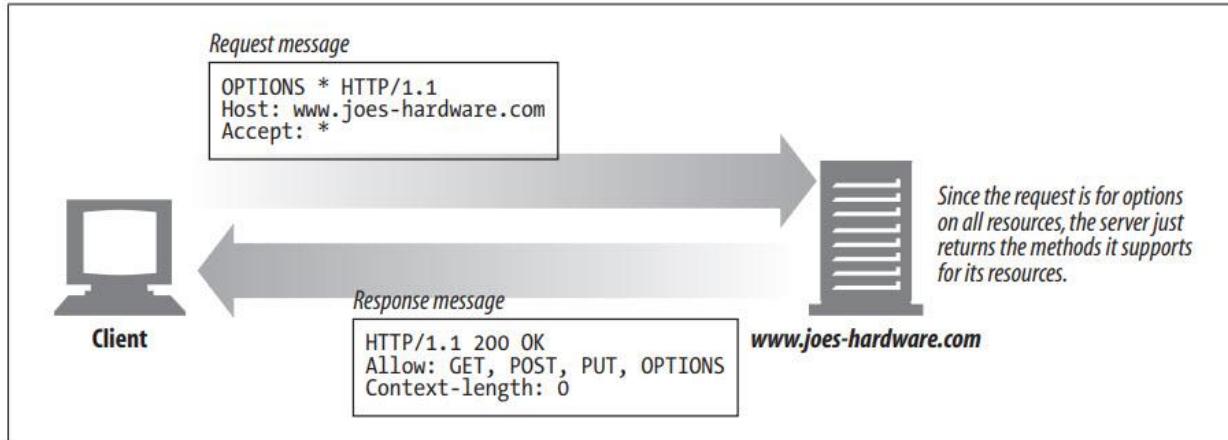
OPTIONS

متد OPTIONS از سرور می‌خواهد که قابلیت‌های مختلف پشتیبانی شده وب سرور را به ما نشان دهد. (برخی سرورها ممکن است عملیات خاصی را فقط بر روی انواع خاصی از اشیاء پشتیبانی کنند).



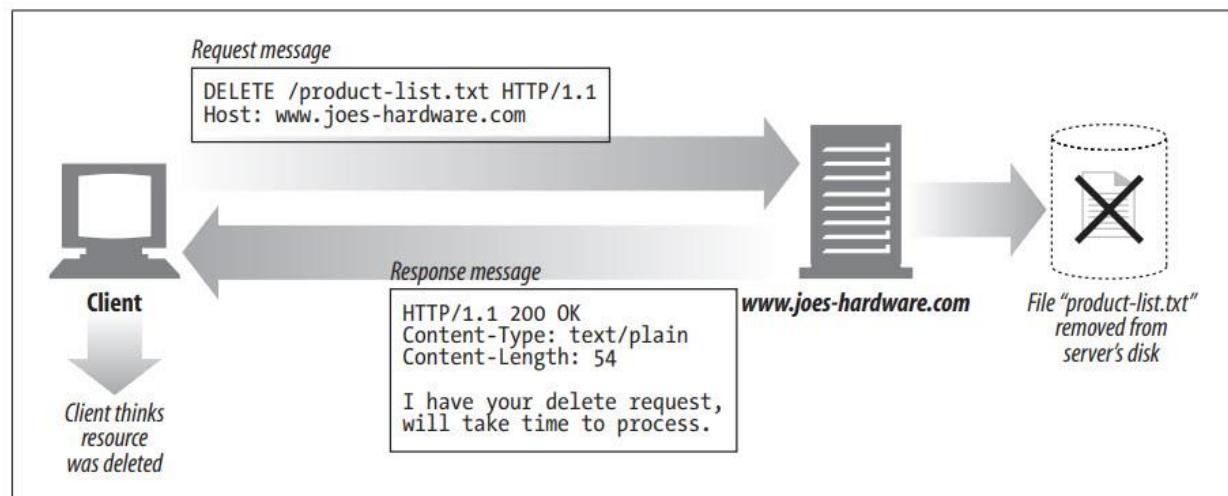


این متد وسیله‌ای را برای برنامه‌های کاربردی کلاینت فراهم می‌کند تا تعیین کنند که چگونه می‌توانند به منابع مختلف دسترسی داشته باشند بدون اینکه واقعاً به آن‌ها دسترسی داشته باشند. شکل زیر یک سناریوی درخواست را با استفاده از روش OPTIONS نشان می‌دهد.



DELETE

متد DELETE دقیقاً همان کاری را انجام می‌دهد که شما فکر می‌کنید. این متد، از سرور می‌خواهد منابع مشخص شده توسط URL را حذف کند. با این حال، استفاده از این متد، تضمینی برای حذف منابع، به برنامه‌های کلاینت نمی‌دهد. این موضوع به این دلیل است که قابلیت‌های پروتکل HTTP به سرور اجازه می‌دهد تا درخواست را بدون اطلاع به مشتری لغو کند. شکل زیر نمونه‌ای از متد DELETE را نشان می‌دهد.



Extension Methods

به گونه‌ای طراحی شده است که قابلیت توسعه میدانی داشته باشد، بنابراین ویژگی‌های جدید باعث از کار افتادن نرم‌افزارهای قدیمی نمی‌شوند. Extension Methods متدایی هستند که در





قابلیت‌های HTTP/1.1 تعریف نشده‌اند. آن‌ها ابزاری را برای توسعه‌دهندگان فراهم می‌کنند تا قابلیت‌های سرویس‌های HTTP را که سرورهای ایشان بر روی منابعی که سرورها مدیریت می‌کنند پیاده‌سازی می‌کنند، گسترش دهند. برخی از نمونه‌های رایج Extension Methods در جدول زیر فهرست شده‌اند. این متدها بخشی از افزونه WebDAV HTTP هستند که به پشتیبانی از انتشار محتوای وب به سرورهای وب از طریق HTTP کمک می‌کند.

Method	Description
LOCK	Allows a user to “lock” a resource—for example, you could lock a resource while you are editing it to prevent others from editing it at the same time
MKCOL	Allows a user to create a resource
COPY	Facilitates copying resources on a server
MOVE	Moves a resource on a server

توجه به این نکته مهم است که همه Extension Method‌ها در یک مشخصات رسمی تعریف نشده‌اند. اگر یک Extension Method را تعریف کنید، احتمالاً برای اکثر برنامه‌های HTTP قابل درک نیست.

Status Codes

همانطور که قبلاً به نیز آن اشاره شد، کدهای وضعیت HTTP به پنج دسته کلی طبقه بندی می‌شوند. کدهای وضعیت راهی آسان برای کلاینت به منظور درک نتایج تراکنش‌های خود را فراهم می‌کند.

100–199: Informational Status Codes

نسخه 1.1 پروتکل HTTP، کدهای وضعیت Informational را معرفی کرد. آن‌ها نسبتاً جدید هستند و در مورد پیچیدگی و ارزش درک شده آن‌ها کمی بحث و جدل وجود دارد. جدول زیر کدهای وضعیت Informational تعریف شده را نشان می‌دهد.

Status code	Reason phrase	Meaning
100	Continue	Indicates that an initial part of the request was received and the client should continue. After sending this, the server must respond after receiving the request. See the Expect header in Appendix C for more information.
101	Switching Protocols	Indicates that the server is changing protocols, as specified by the client, to one listed in the Upgrade header.

کد وضعیت 100 Continue کمی گیج کننده است. هدف آن بهینه سازی مواردی است که در آن یک برنامه کلاینت HTTP دارای یک Entity Body برای ارسال به سرور است اما می‌خواهد بررسی کند که سرور قبل از ارسال آن موجودیت را می‌پذیرد.





Clients and 100 Continue

اگر کلاینت در حال ارسال یک Entity به یک سرور است و مایل است قبل از ارسال آن، منتظر پاسخ 100 باشد، کلاینت باید یک هدر درخواست Expect با مقدار 100-continue ارسال کند. اگر کلاینت قصد ارسال Entity را ندارد، نباید یک هدر Expect با مقدار 100-continue ارسال کند، زیرا این فقط سرور را گیج می‌کند و سرور گمان می‌کند که کلاینت ممکن است یک موجودیت را ارسال کند.

مقدار 100-continue از بسیاری جهات یک بهینه سازی است. یک برنامه کلاینت باید واقعاً از مقدار 100-continue استفاده کند تا از ارسال یک موجودیت بزرگ به سرور قادر به مدیریت یا استفاده از آن نیست، جلوگیری کند.

به دلیل سردرگمی اولیه در مورد وضعیت 100-continue (و با توجه به برخی از پیاده سازی‌های قدیمی‌تر)، کلاینت‌هایی که هدر Expect را برای 100-continue می‌فرستند، نباید برای همیشه منتظر بمانند تا سرور یک پاسخ 100-continue را ارسال کند. پس از مدتی وقفه، کلاینت فقط باید Entity را ارسال کند.

در عمل، پیاده‌کننده‌های کلاینت نیز باید برای مقابله با پاسخ‌های غیرمنتظره 100-continue (آزاردهنده، اما واقعی) آماده باشند. برخی از برنامه‌های کاربردی HTTP به اشتباه این کد را به صورت نامناسب ارسال می‌کنند.

Servers and 100 Continue

اگر سرور درخواستی با هدر Expect و مقدار 100-continue دریافت کند، باید با 100 Continue یا کد خطا به آن پاسخ دهد. سرورها هرگز نباید کد وضعیت 100 را برای کلاینت‌هایی ارسال کنند که 100-continue را ارسال نمی‌کنند. با این حال، همانطور که در بالا اشاره کردیم، برخی از سرورهای خطاكار این کار را انجام می‌دهند.

اگر به دلایلی سرور، قبل از اینکه فرصتی برای ارسال یک پاسخ 100 Continue 100 داشته باشد، مقداری (یا همه) Entity را دریافت کند، نیازی به ارسال این کد وضعیت ندارد، زیرا کلاینت قبلًا تصمیم به ادامه کار گرفته است. با این حال، وقتی سرور خواندن درخواست را تمام کرد، همچنان باید یک کد وضعیت نهایی درخواست ارسال کند.

در نهایت، اگر سروری درخواستی با 100-continue دریافت کند و تصمیم بگیرد قبل از خواندن متن Entity درخواست را پایان دهد (مثلاً به دلیل رخدادن خطایی)، نباید فقط یک پاسخ ارسال کند و اتصال را بینند. زیرا این امر می‌تواند از دریافت پاسخ توسط کلاینت جلوگیری کند.

Proxies and 100 Continue





یک پروکسی که از کلاینت درخواستی دریافت می‌کند که شامل 100-continue است، باید چند کار را انجام دهد. اگر پروکسی بداند که سرور بعدی مطابق با HTTP/1.1 است یا نداند سرور بعدی با چه نسخه‌ای مطابقت دارد، باید درخواست را با هدر Expect در آن ارسال کند. اگر می‌داند که سرور بعدی با نسخه‌ای از HTTP پیش از 1.1 سازگار است، باید با خطای Expectation Failed 417 پاسخ دهد.

اگر پروکسی تصمیم بگیرد از طرف کلاینت سازگار با HTTP/1.0 یا قبل از آن، یک هدر Expect و مقدار 100-continue را در درخواست خود لحاظ کند، نباید پاسخ 100 Continue را (اگر از سرور دریافت کند) به کلاینت ارسال کند. زیرا کلاینت نمی‌داند که از آن چه استفاده‌ای بکند.

200–299: Success Status Codes

هنگامی که کلاینت‌ها درخواستی را ارسال می‌کنند، درخواست‌ها معمولاً موفقیت آمیز هستند. سرورها دارای مجموعه‌ای از کدهای وضعیت برای نشان دادن موفقیت هستند که با انواع مختلف درخواست‌ها مطابقت دارند. جدول زیر کدهای وضعیت موفقیت تعریف شده را فهرست می‌کند.





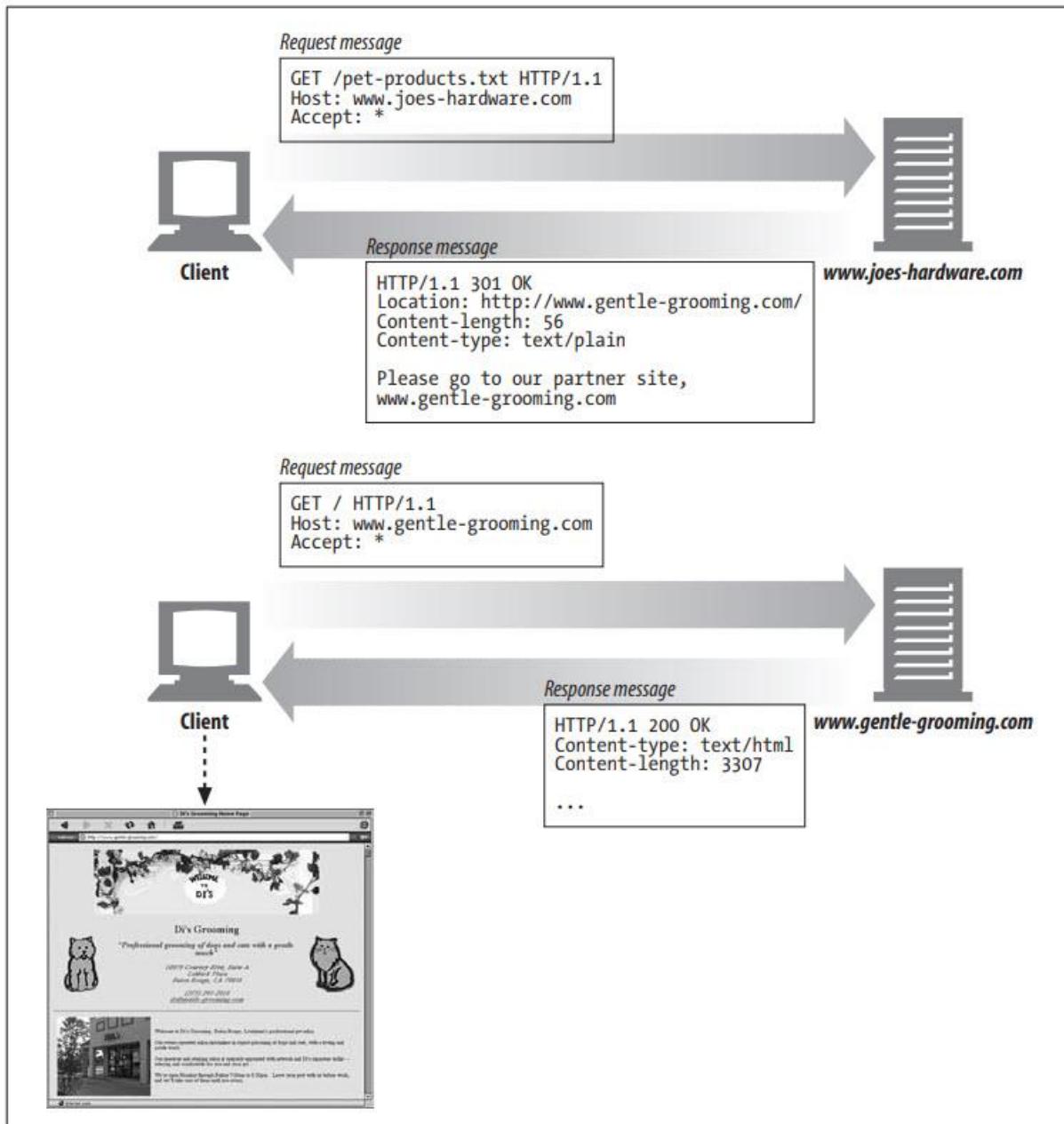
Status code	Reason phrase	Meaning
200	OK	Request is okay, entity body contains requested resource.
201	Created	For requests that create server objects (e.g., PUT). The entity body of the response should contain the various URLs for referencing the created resource, with the Location header containing the most specific reference. See Table 3-21 for more on the Location header.
202	Accepted	The server must have created the object prior to sending this status code. The request was accepted, but the server has not yet performed any action with it. There are no guarantees that the server will complete the request; this just means that the request looked valid when accepted. The server should include an entity body with a description indicating the status of the request and possibly an estimate for when it will be completed (or a pointer to where this information can be obtained).
203	Non-Authoritative Information	The information contained in the entity headers (see “Entity Headers” for more information on entity headers) came not from the origin server but from a copy of the resource. This could happen if an intermediary had a copy of a resource but could not or did not validate the meta-information (headers) it sent about the resource. This response code is not required to be used; it is an option for applications that have a response that would be a 200 status if the entity headers had come from the origin server.
204	No Content	The response message contains headers and a status line, but no entity body. Primarily used to update browsers without having them move to a new document (e.g., refreshing a form page).
205	Reset Content	Another code primarily for browsers. Tells the browser to clear any HTML form elements on the current page.
206	Partial Content	A partial or <i>range</i> request was successful. Later, we will see that clients can request part or a range of a document by using special headers—this status code indicates that the range request was successful. See “Range Requests” in Chapter 15 for more on the Range header.

300–399: Redirection Status Codes

کدهای وضعیت تغییر مسیر یا به کلاینت می‌گویند که از مکان‌های جایگزین برای منابع مورد علاقه خود استفاده کنند یا به جای محتوا، یک پاسخ جایگزین ارائه دهند. اگر منبعی جایه‌جا شده باشد، یک کد وضعیت تغییر مسیر و یک هدر Location اختیاری می‌تواند ارسال شود تا به کلاینت بگوید که منبع منتقل شده است و اکنون می‌توان آن را پیدا کرد در مسیر جدید پیدا کند (شکل زیر را ببینید).

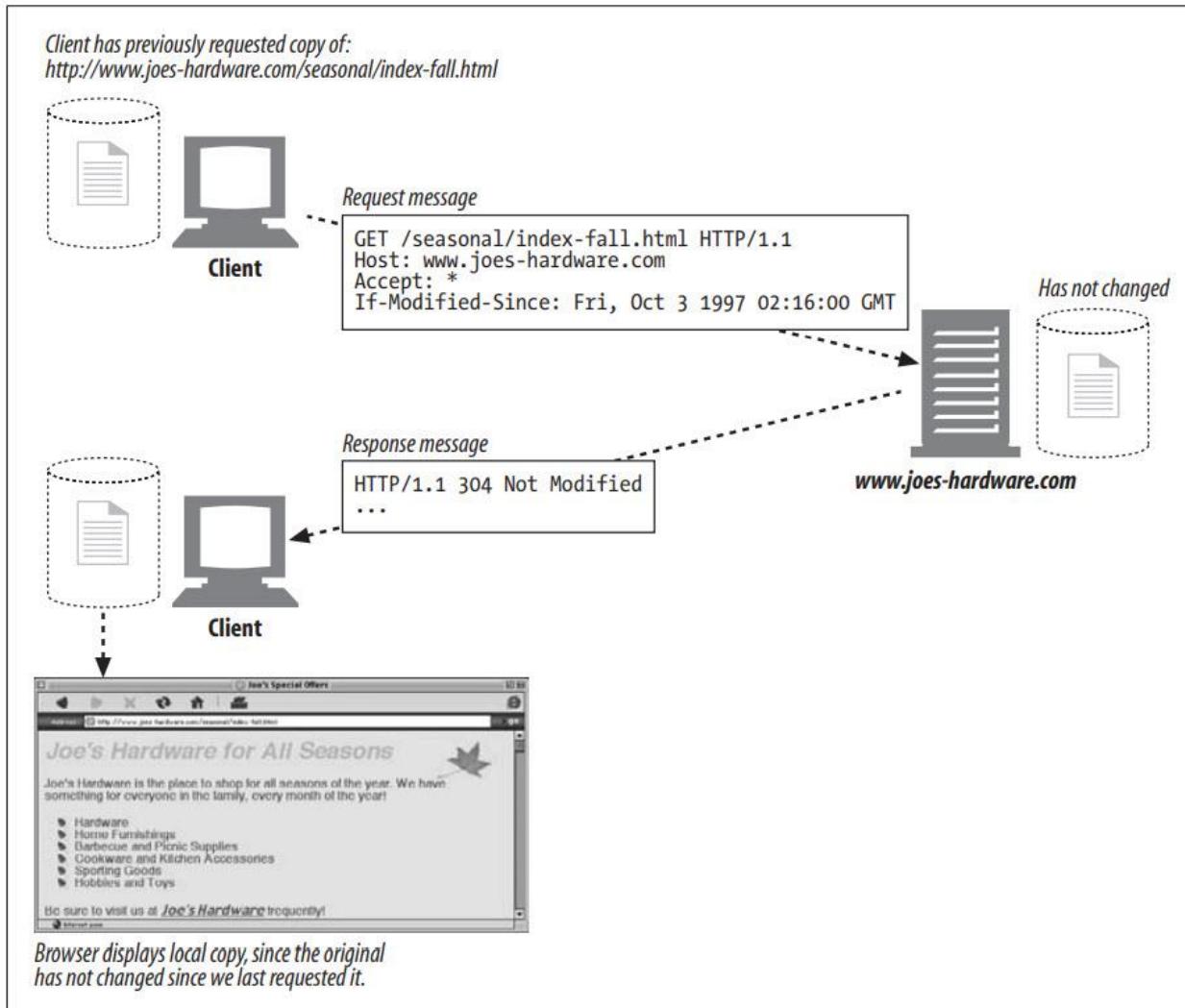
این به مرورگرهای اجازه می‌دهد تا به صورت شفاف و بدون آزار کاربران خود به مکان جدید هدایت کنند.





برخی از کدهای وضعیت تغییر مسیر را می‌توان برای تأیید اعتبار یک Local Copy برای برنامه کاربردی از یک منبع با سرور مبدا استفاده کرد. به عنوان مثال، یک برنامه HTTP می‌تواند بررسی کند که آیا Local Copy منبع آن هنوز به روز است یا اینکه منبع در سرور اصلی اصلاح شده است. شکل زیر نمونه‌ای از این موضوع را نشان می‌دهد. کلاینت یک هدر ویژه If-Modified-Since را ارسال می‌کند که می‌گوید فقط در صورتی سند را دریافت کنید که از اکتبر ۱۹۹۷ اصلاح شده باشد. سند از این تاریخ تغییر نکرده است، بنابراین سرور به جای محتویات، با یک کد وضعیت 304 پاسخ می‌دهد.





به طور کلی، برای پاسخ به درخواست‌های غیر HEAD که شامل یک کد وضعیت تغییر مسیر هستند، روش خوبی است که شامل یک Entity با توضیحات و لینک‌هایی به URL(های) هدایت شده باشد. جدول زیر کدهای وضعیت تغییر مسیر تعریف شده را فهرست می‌کند.





Status code	Reason phrase	Meaning
300	Multiple Choices	Returned when a client has requested a URL that actually refers to multiple resources, such as a server hosting an English and French version of an HTML document. This code is returned along with a list of options; the user can then select which one he wants. See Chapter 17 for more on clients negotiating when there are multiple versions. The server can include the preferred URL in the Location header.
301	Moved Permanently	Used when the requested URL has been moved. The response should contain in the Location header the URL where the resource now resides.
302	Found	Like the 301 status code; however, the client should use the URL given in the Location header to locate the resource temporarily. Future requests should use the old URL.
303	See Other	Used to tell the client that the resource should be fetched using a different URL. This new URL is in the Location header of the response message. Its main purpose is to allow responses to POST requests to direct a client to a resource.
304	Not Modified	Clients can make their requests conditional by the request headers they include. See Table 3-15 for more on conditional headers. If a client makes a conditional request, such as a GET if the resource has not been changed recently, this code is used to indicate that the resource has not changed. Responses with this status code should not contain an entity body.
305	Use Proxy	Used to indicate that the resource must be accessed through a proxy; the location of the proxy is given in the Location header. It's important that clients interpret this response relative to a specific resource and do not assume that this proxy should be used for all requests or even all requests to the server holding the requested resource. This could lead to broken behavior if the proxy mistakenly interfered with a request, and it poses a security hole.
306	(Unused)	Not currently used.
307	Temporary Redirect	Like the 301 status code; however, the client should use the URL given in the Location header to locate the resource temporarily. Future requests should use the old URL.

از جدول بالا، ممکن است متوجه کمی همپوشانی بین کدهای وضعیت ۳۰۲، ۳۰۳ و ۳۰۷ شده باشید. تفاوت‌های ظریفی در نحوه استفاده از این کدهای وضعیت وجود دارد، که بیشتر آن‌ها ناشی از تفاوت در متدهایی است که برنامه‌های HTTP/1.0 و HTTP/1.1 با این کدهای وضعیت برخورد می‌کنند.

هنگامی که یک کلاینت HTTP/1.0 یک درخواست POST ارسال می‌کند و در پاسخ یک کد وضعیت تغییر مسیر 302 دریافت می‌کند، URL تغییر مسیر را در هدر Location با یک درخواست GET به آن URL دنبال می‌کند.

سرورهای HTTP/1.0 انتظار دارند کلاینت‌های HTTP/1.0 این کار را انجام دهند. زمانی که سرور HTTP/1.0 پس از دریافت یک درخواست POST از یک کلاینت HTTP/1.0 یک کد وضعیت 302 ارسال می‌کند، سرور انتظار دارد که کلاینت با درخواست GET، تغییر مسیر را دنبال کند.





سردرگمی با 1.1 HTTP ایجاد می‌شود. پروتکل 1.1 HTTP از کد وضعیت 303 برای دریافت همین رفتار استفاده می‌کند (سرورها کد وضعیت 303 را برای تغییر مسیر درخواست POST مشتری ارسال می‌کنند تا با یک درخواست GET دنبال شود).

به منظور دور زدن این سردرگمی، پروتکل 1.1 HTTP می‌گوید که از کد وضعیت 307 به جای کد وضعیت 302 برای تغییر مسیرهای موقت به کلاینت‌های 1.1 HTTP استفاده کنید. سپس سرورها می‌توانند کد وضعیت 302 را برای استفاده با کلاینت‌های 1.0 HTTP ذخیره کنند.

همه موارد مذکور ما را به این نتیجه می‌رساند که سرورها باید نسخه HTTP یک کلاینت را بررسی کنند تا به درستی کد وضعیت تغییر مسیر را برای ارسال در پاسخ ریدایرکت انتخاب کنند.

400–499: Client Error Status Codes

گاهی اوقات یک کلاینت چیزی را درخواست می‌کند که سرور نمی‌تواند از پس آن براآید، مانند یک پیام درخواست بد شکل یا، اغلب، درخواستی برای URL که وجود ندارد.

همه ما کد خطای 404 Not Found را هنگام مرور دیده‌ایم – در این حالت سرور به ما می‌گوید که ما منبعی را درخواست کرده‌ایم که درباره آن چیزی نمی‌داند.

بسیاری از خطاهای مبتنی بر کلاینت توسط مرورگر شما برطرف می‌شود، بدون اینکه هرگز شما را آزار دهد. جدول زیر کدهای مختلف وضعیت خطای کلاینت را نشان می‌دهد.





Status code	Reason phrase	Meaning
400	Bad Request	Used to tell the client that it has sent a malformed request.
401	Unauthorized	Returned along with appropriate headers that ask the client to authenticate itself before it can gain access to the resource. See Chapter 12 for more on authentication.
402	Payment Required	Currently this status code is not used, but it has been set aside for future use.
403	Forbidden	Used to indicate that the request was refused by the server. If the server wants to indicate why the request was denied, it can include an entity body describing the reason. However, this code usually is used when the server does not want to reveal the reason for the refusal.
404	Not Found	Used to indicate that the server cannot find the requested URL. Often, an entity is included for the client application to display to the user.
405	Method Not Allowed	Used when a request is made with a method that is not supported for the requested URL. The Allow header should be included in the response to tell the client what methods are allowed on the requested resource. See "Entity Headers" for more on the Allow header.
406	Not Acceptable	Clients can specify parameters about what types of entities they are willing to accept. This code is used when the server has no resource matching the URL that is acceptable for the client. Often, servers include headers that allow the client to figure out why the request could not be satisfied. See "Content Negotiation and Transcoding" in Chapter 17 for more information.
407	Proxy Authentication Required	Like the 401 status code, but used for proxy servers that require authentication for a resource.
408	Request Timeout	If a client takes too long to complete its request, a server can send back this status code and close down the connection. The length of this timeout varies from server to server but generally is long enough to accommodate any legitimate request.
409	Conflict	Used to indicate some conflict that the request may be causing on a resource. Servers might send this code when they fear that a request could cause a conflict. The response should contain a body describing the conflict.
410	Gone	Similar to 404, except that the server once held the resource. Used mostly for web site maintenance, so a server's administrator can notify clients when a resource has been removed.
411	Length Required	Used when the server requires a Content-Length header in the request message. See "Content headers" for more on the Content-Length header.
412	Precondition Failed	Used if a client makes a conditional request and one of the conditions fails. Conditional requests occur when a client includes an Expect header. See Appendix C for more on the Expect header.
413	Request Entity Too Large	Used when a client sends an entity body that is larger than the server can or wants to process.
414	Request URI Too Long	Used when a client sends a request with a request URL that is larger than the server can or wants to process.
415	Unsupported Media Type	Used when a client sends an entity of a content type that the server does not understand or support.
416	Requested Range Not Satisfiable	Used when the request message requested a range of a given resource and that range either was invalid or could not be met.
417	Expectation Failed	Used when the request contained an expectation in the Expect request header that the server could not satisfy. See Appendix C for more on the Expect header. A proxy or other intermediary application can send this response code if it has unambiguous evidence that the origin server will generate a failed expectation for the request.



500–599: Server Error Status Codes

گاهی اوقات یک کلاینت یک درخواست معتبر ارسال می‌کند، اما خود سرور دارای خطای است. در این حالت ممکن است کلاینت با محدودیت سرور یا یک خطای در یکی از اجزای فرعی سرور، مانند یک منبع *Gateway*، مواجه شده باشد.

پروکسی‌ها اغلب هنگام تلاش برای صحبت با سرورها از طرف کلاینت با مشکل مواجه می‌شوند. پروکسی‌ها کدهای وضعیت خطای سرور 5xx را برای توصیف مشکل صادر می‌کنند. جدول زیر کدهای وضعیت خطای سرور تعریف شده را فهرست می‌کند.

Status code	Reason phrase	Meaning
500	Internal Server Error	Used when the server encounters an error that prevents it from servicing the request.
501	Not Implemented	Used when a client makes a request that is beyond the server's capabilities (e.g., using a request method that the server does not support).
502	Bad Gateway	Used when a server acting as a proxy or gateway encounters a bogus response from the next link in the request response chain (e.g., if it is unable to connect to its parent gateway).
503	Service Unavailable	Used to indicate that the server currently cannot service the request but will be able to in the future. If the server knows when the resource will become available, it can include a Retry-After header in the response. See "Response Headers" for more on the Retry-After header.
504	Gateway Timeout	Similar to status code 408, except that the response is coming from a gateway or proxy that has timed out waiting for a response to its request from another server.
505	HTTP Version Not Supported	Used when a server receives a request in a version of the protocol that it can't or won't support. Some server applications elect not to support older versions of the protocol.

Headers

هدرها و متدها با هم کار می‌کنند تا مشخص کنند که کلاینت‌ها و سرورها چه کاری انجام می‌دهند.

هدرهایی وجود دارند که برای پیام‌های خاص استفاده شده و برخی نیز از نظر هدف کلی تر هستند و اطلاعاتی را هم در پیام‌های درخواست و هم در پاسخ ارائه می‌دهند. هدرها به پنج کلاس اصلی تقسیم می‌شوند:





General headers

این‌ها هدرهای عمومی هستند که هم توسط کلاینت و هم سرور استفاده می‌شوند. آن‌ها اهداف کلی را ارائه می‌کنند که برای کلاینت، سرور و سایر برنامه‌ها مفید است. به عنوان مثال، هدر Date یک هدر همه منظوره است که به هر دو طرف اجازه می‌دهد زمان و تاریخی که در آن پیام ساخته شده است را نشان دهنده:

Date: Tue, 3 Oct 1974 02:16:00 GMT

Request headers

همانطور که از نام آن پیداست، هدرهای درخواست مخصوص پیام‌های درخواست هستند. آن‌ها اطلاعات اضافی را در اختیار سرورها قرار می‌دهند، مانند نوع داده‌ایی که کلاینت مایل به دریافت آن است. به عنوان مثال، هدر Accept زیر به سرور می‌گوید که کلاینت هر نوع رسانه‌ای را که با درخواستش مطابقت داشته باشد می‌پذیرد:

Accept: */*

Response headers

پیام‌های پاسخ مجموعه‌ای از هدرهای خاص هستند که اطلاعاتی را در اختیار کلاینت قرار می‌دهند (به عنوان مثال، کلاینت با چه نوع سروری صحبت می‌کند). به عنوان مثال، هدر Server زیر به کلاینت می‌گوید که با سرور Tiki-Hut نسخه ۱.۰ صحبت می‌کند:

Server: Tiki-Hut/1.0

Entity headers

هدر موجودیت به هدرهایی اشاره دارد که با Entity Body سروکار دارند. به عنوان مثال، هدرهای Entity می‌توانند نوع داده‌ها را در بدن موجودیت مشخص کنند. برای مثال، هدر Content-Type زیر به برنامه اجازه می‌دهد بداند که داده‌های یک سند HTML در مجموعه کاراکترهای iso-latin-1 هستند:

Content-Type: text/html; charset=iso-latin-1

Extension headers

هدرهای Extension، هدرهای غیر استانداردی هستند که توسط توسعه دهندگان برنامه ایجاد شده اند اما هنوز به مشخصات HTTP اضافه نشده اند. برنامه‌های HTTP باید هدرهای Extension را تحمل کرده و ارسال کنند، حتی اگر معنی هدرها را ندانند.





General Headers

برخی از هدرها، اطلاعات بسیار ابتدایی را در مورد یک پیام ارائه می‌دهند. به این هدرها، هدرهای عمومی می‌گویند. این هدرها، اطلاعات مفیدی را در مورد یک پیام صرف نظر از نوع آن ارائه می‌دهند.

به عنوان مثال، چه در حال ساخت یک پیام درخواست یا یک پیام پاسخ باشید، تاریخ و زمان ایجاد پیام به یک معنی است، بنابراین هدری که این نوع اطلاعات را ارائه می‌دهد برای هر دو نوع پیام، General است. جدول زیر هدرهای اطلاعاتی کلی را فهرست می‌کند.

Header	Description
Connection	Allows clients and servers to specify options about the request/response connection
Date ^a	Provides a date and time stamp telling when the message was created
MIME-Version	Gives the version of MIME that the sender is using
Trailer	Lists the set of headers that are in the trailer of a message encoded with the chunked transfer encoding ^b
Transfer-Encoding	Tells the receiver what encoding was performed on the message in order for it to be transported safely
Upgrade	Gives a new version or protocol that the sender would like to "upgrade" to using
Via	Shows what intermediaries (proxies, gateways) the message has gone through

General caching headers

پروتکل HTTP/1.0 هدرهایی را معرفی کرد که به برنامه‌های HTTP اجازه می‌داد تا Local Copy از اشیاء را به جای اینکه همیشه مستقیماً از سرور اصلی واکسی کنند، در Cache ذخیره کنند. آخرین نسخه HTTP دارای مجموعه بسیار غنی از پارامترهای Cache است. جدول زیر هدرهای اصلی Caching را فهرست می‌کند.

Header	Description
Cache-Control	Used to pass caching directions along with the message
Pragma ^a	Another way to pass directions along with the message, though not specific to caching

^a Pragma technically is a request header. It was never specified for use in responses. Because of its common misuse as a response header, many clients and proxies will interpret Pragma as a response header, but the precise semantics are not well defined. In any case, Pragma is deprecated in favor of Cache-Control.

Request Headers

هدرهای درخواست، هدرهایی هستند که فقط در پیام درخواست معنا می‌یابند. آن‌ها اطلاعاتی در مورد اینکه چه کسی یا چه چیزی درخواست را ارسال می‌کند، درخواست از کجا شروع شده است، یا اولویت‌ها و توانایی‌های کلاینت چیست، ارائه می‌دهند. سرورها می‌توانند از اطلاعاتی که هدر درخواست در مورد





کلاینت به آن‌ها می‌دهد استفاده کنند تا سعی کنند پاسخ بهتری به کلاینت بدهند. جدول زیر هدرها اطلاعاتی درخواست را فهرست می‌کند.

Header	Description
Client-IP ^a	Provides the IP address of the machine on which the client is running
From	Provides the email address of the client's user ^b
Host	Gives the hostname and port of the server to which the request is being sent
Referer	Provides the URL of the document that contains the current request URL
UA-Color	Provides information about the color capabilities of the client machine's display
UA-CPU ^c	Gives the type or manufacturer of the client's CPU
UA-Disp	Provides information about the client's display (screen) capabilities
UA-OS	Gives the name and version of operating system running on the client machine
UA-Pixels	Provides pixel information about the client machine's display
User-Agent	Tells the server the name of the application making the request

^a Client-IP and the UA-* headers are not defined in RFC 2616 but are implemented by many HTTP client applications.

^b An RFC 822 email address format.

^c While implemented by some clients, the UA-* headers can be considered harmful. Content, specifically HTML, should not be targeted at specific client configurations.

Accept headers

هدرهای Accept راهی را به کلاینت ارائه می‌دهد تا ترجیحات و قابلیت‌های خود را به سرورها بگوید:

چه چیزی را می‌خواهند، چه چیزی را می‌توانند استفاده کنند و مهمتر از همه، چه چیزی را نمی‌خواهند. سپس سرورها می‌توانند از این اطلاعات اضافی برای تصمیم‌گیری هوشمندانه تر در مورد ارسال اطلاعات استفاده کنند.

هدرهای Accept به نفع هر دو طرف اتصال هستند. کلاینت‌ها آنچه را که می‌خواهند دریافت می‌کنند و سرورها زمان و پهنانی باند خود را برای ارسال چیزی که کلاینت نمی‌تواند از آن استفاده کند هدر نمی‌دهند. جدول زیر هدرهای مختلف Accept را فهرست می‌کند.



Header	Description
Accept	Tells the server what media types are okay to send
Accept-Charset	Tells the server what charsets are okay to send
Accept-Encoding	Tells the server what encodings are okay to send
Accept-Language	Tells the server what languages are okay to send
TE ^a	Tells the server what extension transfer codings are okay to use

^a See "Transfer-Encoding Headers" in Chapter 15 for more on the TE header.

Conditional request headers

گاهی اوقات، کلاینت‌ها می‌خواهند محدودیت‌هایی را روی یک درخواست اعمال کنند. به عنوان مثال، اگر کلاینت قبل‌اً یک سند داشته باشد، ممکن است بخواهد از یک سرور بخواهد که سند را فقط در صورتی ارسال کند که با نسخه‌ای که وی قبل‌اً دارد متفاوت باشد. با استفاده از هدرهای درخواست Conditional، کلاینت‌ها می‌توانند چنین محدودیت‌هایی را روی درخواست‌ها اعمال کنند و از سرور بخواهند قبل از برآورده کردن درخواست، از صحت Conditional اطمینان حاصل کند. جدول زیر هدرهای مختلف درخواست Conditional را فهرست می‌کند.

Header	Description
Expect	Allows a client to list server behaviors that it requires for a request
If-Match	Gets the document if the entity tag matches the current entity tag for the document ^a
If-Modified-Since	Restricts the request unless the resource has been modified since the specified date
If-None-Match	Gets the document if the entity tags supplied do not match those of the current document
If-Range	Allows a conditional request for a range of a document
If-Unmodified-Since	Restricts the request unless the resource has <i>not</i> been modified since the specified date
Range	Requests a specific range of a resource, if the server supports range requests ^b

^a See Chapter 7 for more on entity tags. The tag is basically an identifier for a version of the resource.

^b See "Range Requests" in Chapter 15 for more on the Range header.

Request security headers

HTTP به صورت بومی از یک طرح ساده احراز هویت challenge/response برای درخواست‌ها پشتیبانی می‌کند. این مدل تلاش می‌کند تا ارتباط را اندکی امن‌تر کند و از کلاینت‌ها می‌خواهد قبل از دسترسی به منابع خاص، خود را احراز هویت کنند. جدول زیر هدرهای امنیتی درخواست را فهرست می‌کند.





Header	Description
Authorization	Contains the data the client is supplying to the server to authenticate itself
Cookie	Used by clients to pass a token to the server—not a true security header, but it does have security implications ^a
Cookie2	Used to note the version of cookies a requestor supports; see “Version 1 (RFC 2965) Cookies” in Chapter 11

^a The Cookie header is not defined in RFC 2616; it is discussed in detail in Chapter 11.

Proxy request headers

از آنجایی که پراکسی‌ها در اینترنت به طور فزاینده‌ای رایج شده‌اند، چند هدر برای کمک به عملکرد بهتر آن‌ها تعریف شده است. جدول زیر هدرهای درخواست پروکسی را فهرست می‌کند.

Header	Description
Max-Forwards	The maximum number of times a request should be forwarded to another proxy or gateway on its way to the origin server—used with the TRACE method ^a
Proxy-Authorization	Same as Authorization, but used when authenticating with a proxy
Proxy-Connection	Same as Connection, but used when establishing connections with a proxy

^a See “Max-Forwards” in Chapter 6.

Response Headers

هدرهای پاسخ اطلاعات اضافی را به مشتریان ارائه می‌دهند، مانند اینکه چه کسی پاسخ را ارسال می‌کند، توانایی‌های پاسخ‌دهنده یا حتی دستورالعمل‌های ویژه در مورد پاسخ چیست. این هدرها به کلاینت کمک می‌کند تا پاسخ‌ها مشاهده نموده و در آینده درخواست‌های بهتری ارائه دهد. جدول زیر هدرهای اطلاعاتی پاسخ را فهرست می‌کند.





Header	Description
Age	How old the response is ^a
Public ^b	A list of request methods the server supports for its resources
Retry-After	A date or time to try back, if a resource is unavailable
Server	The name and version of the server's application software
Title ^c	For HTML documents, the title as given by the HTML document source
Warning	A more detailed warning message than what is in the reason phrase

^a Implies that the response has traveled through an intermediary, possibly from a proxy cache.

^b The Public header is defined in RFC 2068 but does not appear in the latest HTTP definition (RFC 2616).

^c The Title header is not defined in RFC 2616; see the original HTTP/1.0 draft definition (<http://www.w3.org/Protocols/HTTP/HTTP2.html>).

Negotiation headers

پروتکل HTTP/1.1 به سرورها و کلاینت‌ها این امکان را می‌دهد که در صورت وجود چندین نمایش، برای یک منبع Negotiation کنند. (به عنوان مثال، وقتی ترجمه‌های فرانسوی و آلمانی یک سند HTML روی یک سرور وجود دارد) در اینجا چند هدر وجود دارد که سرورها برای انتقال اطلاعات در مورد منابع قابل مذاکره از آن استفاده می‌کنند. جدول زیر هدرهای مذکور را فهرست می‌کند.

Header	Description
Accept-Ranges	The type of ranges that a server will accept for this resource
Vary	A list of other headers that the server looks at and that may cause the response to vary; i.e., a list of headers the server looks at to pick which is the best version of a resource to send the client

Response security headers

شما قبلاً هدرهای امنیتی درخواست را دیده‌اید، که اساساً سمت پاسخ طرح احراز هویت مربوط به پروتکل HTTP challenge/response است. در حال حاضر، در اینجا هدرهای اصلی چالش آورده شده است. جدول زیر هدرهای امنیتی پاسخ را فهرست می‌کند.





Header	Description
Proxy-Authenticate	A list of challenges for the client from the proxy
Set-Cookie	Not a true security header, but it has security implications; used to set a token on the client side that the server can use to identify the client ^a
Set-Cookie2	Similar to Set-Cookie, RFC 2965 Cookie definition; see "Version 1 (RFC 2965) Cookies" in Chapter 11
WWW-Authenticate	A list of challenges for the client from the server

^a Set-Cookie and Set-Cookie2 are extension headers that are also covered in Chapter 11.

Entity Headers

هدرهای زیادی برای توصیف پیلود پیام‌های HTTP وجود دارد. از آنجایی که هر دو پیام درخواست و پاسخ می‌توانند دارای موجودیت باشند، این هدرها می‌توانند در هر نوع پیام ظاهر شوند.

هدرهای Entity یا موجودیت طیف وسیعی از اطلاعات را در مورد موجودیت و محتوای آن، از اطلاعات مربوط به نوع شیء گرفته تا متدهای درخواست معتبری که می‌توان روی منبع ایجاد کرد، ارائه می‌کند. به طور کلی، هدرهای موجودیت به گیرنده پیام می‌گویند که با چه چیزی سروکار دارد. جدول زیر هدرهای اطلاعاتی موجودیت را فهرست می‌کند.

Header	Description
Allow	Lists the request methods that can be performed on this entity
Location	Tells the client where the entity really is located; used in directing the receiver to a (possibly new) location (URL) for the resource





Content headers

هدرهای Content، اطلاعات خاصی در مورد محتوای موجودیت ارائه می‌دهند، که نوع، اندازه و سایر اطلاعات مفید برای پردازش، نمونه‌ای از آن‌ها می‌باشد. به عنوان مثال، یک مرورگر وب می‌تواند به نوع محتوای بازگشتی نگاه کند و بداند چگونه شیء را نمایش دهد. جدول زیر هدرهای مختلف محتوا را فهرست می‌کند.

Header	Description
Content-Base ^a	The base URL for resolving relative URLs within the body
Content-Encoding	Any encoding that was performed on the body
Content-Language	The natural language that is best used to understand the body
Content-Length	The length or size of the body
Content-Location	Where the resource actually is located
Content-MD5	An MD5 checksum of the body
Content-Range	The range of bytes that this entity represents from the entire resource
Content-Type	The type of object that this body is

Entity caching headers

هدرهای Cache عمومی دستورالعمل‌هایی را در مورد چگونگی یا زمان Cache ارائه می‌دهند. هدرهای موجود در Cache اطلاعاتی را در مورد موجودیتی که در حافظه Cache می‌شود ارائه می‌دهد. به عنوان مثال، اطلاعات مورد نیاز برای تأیید اعتبار یک کپی ذخیره شده در حافظه Cache منبع هنوز معتبر است یا خیر.

جدول زیر هدرهای Entity caching را فهرست می‌کند.

Header	Description
ETag	The entity tag associated with this entity ^a
Expires	The date and time at which this entity will no longer be valid and will need to be fetched from the original source
Last-Modified	The last date and time when this entity changed

^a Entity tags are basically identifiers for a particular version of a resource.





For More Information

<http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

HTTP Pocket Reference Clintin Wong, O'Reilly & Associates, Inc.

<http://www.w3.org/Protocols/>





فصل چهارم – Connection Management

پروتکل HTTP پیام‌های HTTP را به خوبی توضیح می‌دهد، اما در مورد اتصالات HTTP، ساختار حیاتی که پیام‌های HTTP از آن عبور می‌کنند، صحبتی نمی‌شود. اگر برنامه نویسی هستید که برنامه‌های HTTP را توسعه می‌دهید، باید نکات مربوط به اتصالات HTTP و نحوه استفاده از آن‌ها را بدانید.

در این فصل که به مدیریت اتصال HTTP پرداخته می‌شود با موارد زیر آشنا خواهید شد:

- نحوه استفاده HTTP از اتصالات TCP
- تاخیر، bottlenecks و گرفتگی در اتصالات TCP
- بهینه سازی‌های HTTP، از جمله اتصالات Parallel و Keep-Alive
- بایدها و نبایدها برای مدیریت ارتباطات

TCP Connections

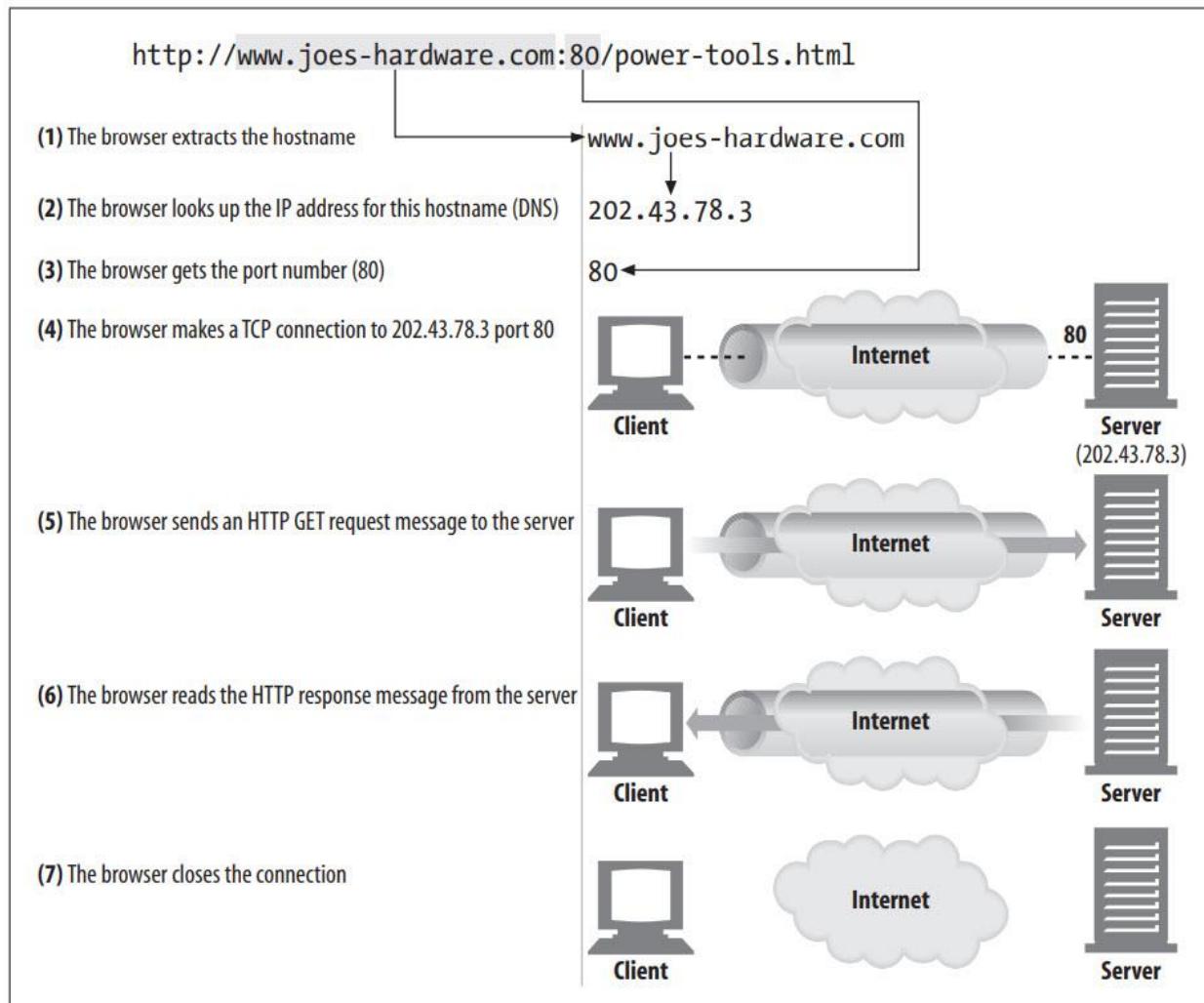
تقریباً تمام ارتباطات HTTP از طریق TCP/IP انجام می‌شود، یک مجموعه لایه‌ای محبوب از پروتکل‌های شبکه packet-switched که توسط رایانه‌ها و دستگاه‌های شبکه در سراسر جهان از آن استفاده می‌شود. یک برنامه سمت کلاینت می‌تواند یک اتصال TCP/IP را با یک برنامه سرور باز کند، که تقریباً در هر نقطه از جهان اجرا می‌شود. پس از برقراری ارتباط، پیام‌های رد و بدل شده بین رایانه‌های کلاینت و سرور هرگز از بین نرفته و آسیب نمی‌بینند.

آدرس زیر را در نظر بگیرید:

<http://www.joes-hardware.com:80/power-tools.html>

وقتی این URL به شما داده می‌شود، مرورگر شما مراحل نشان داده شده در شکل زیر را انجام می‌دهد.





در مراحل ۱ تا ۳، آدرس IP و شماره پورت سرور از URL خارج می‌شود.

در مرحله ۴ یک اتصال TCP به وب سرور ایجاد شده و در مرحله ۵ یک پیام درخواست در سرسر اتصال ارسال می‌شود.

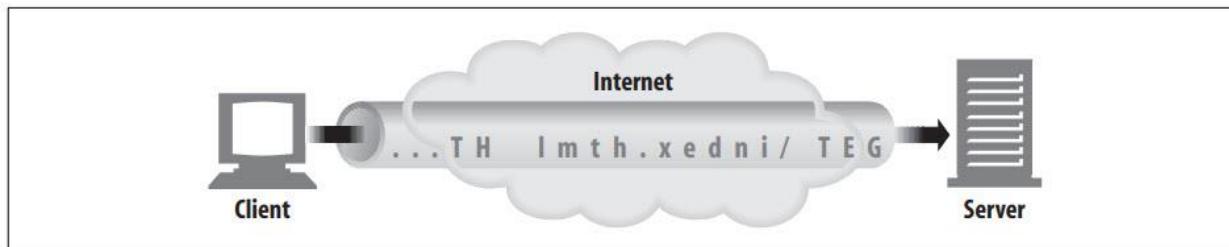
پاسخ در مرحله ۶ خوانده شده و اتصال در مرحله ۷ بسته می‌شود.

TCP Reliable Data Pipes

اتصالات HTTP واقعاً چیزی بیش از اتصالات TCP به علاوه چند قانون در مورد نحوه استفاده از آن‌ها نیستند. اتصالات TCP اتصالات قابل اعتماد اینترنت هستند.

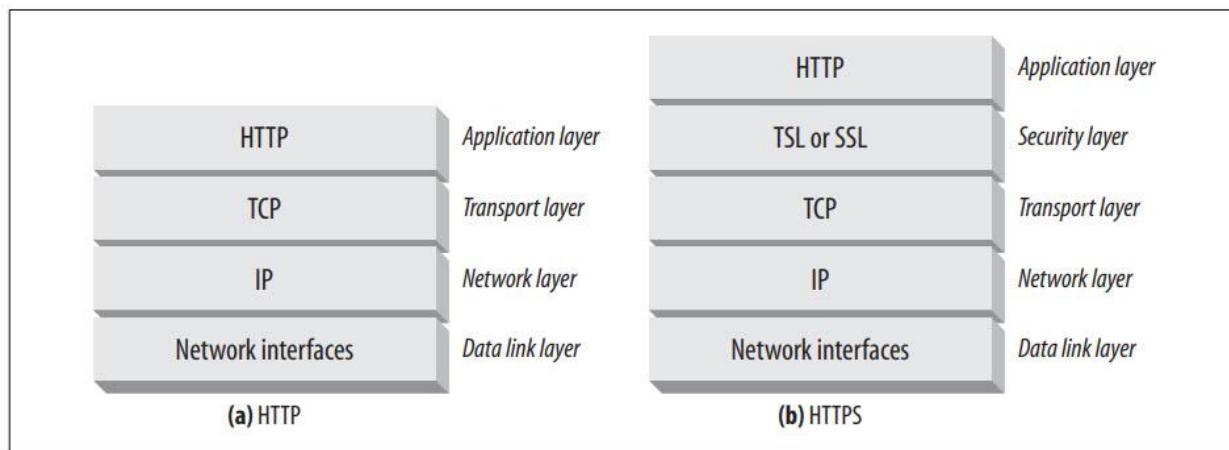
TCP به HTTP یک لوله از بیت‌های قابل اعتماد می‌دهد. بایت‌های ارسال شده در یک طرف اتصال TCP از طرف دیگر به درستی و به ترتیب درست بیرون می‌آیند.





TCP Streams Are Segmented and Shipped by IP Packets

TCP داده‌های خود را در تکه‌های کوچکی به نام بسته‌های IP (یا دیتاگرام IP) ارسال می‌کند. به این ترتیب، HTTP لایه بالایی در "پشته پروتکل" است، همانطور که در بخش a از شکل زیر نشان داده شده است. یک نوع آمن، HTTPS، یک لایه رمزگذاری رمزنگاری (به نام TLS یا SSL) را بین TCP و HTTP قرار می‌دهد (بخش b شکل زیر).



هنگامی که HTTP می‌خواهد پیامی را ارسال کند، محتوای داده پیام را به ترتیب از طریق یک اتصال TCP باز، پخش می‌کند. TCP جریان داده را می‌گیرد، جریان داده را به قطعاتی به نام سگمنت تقسیم می‌کند و بخش‌ها را در سراسر اینترنت در داخل پاکت‌هایی به نام بسته‌های IP منتقل می‌کند. همه این‌ها توسط ساختار TCP/IP انجام می‌شود. برنامه نویس HTTP هیچ کدام را نمی‌بیند.

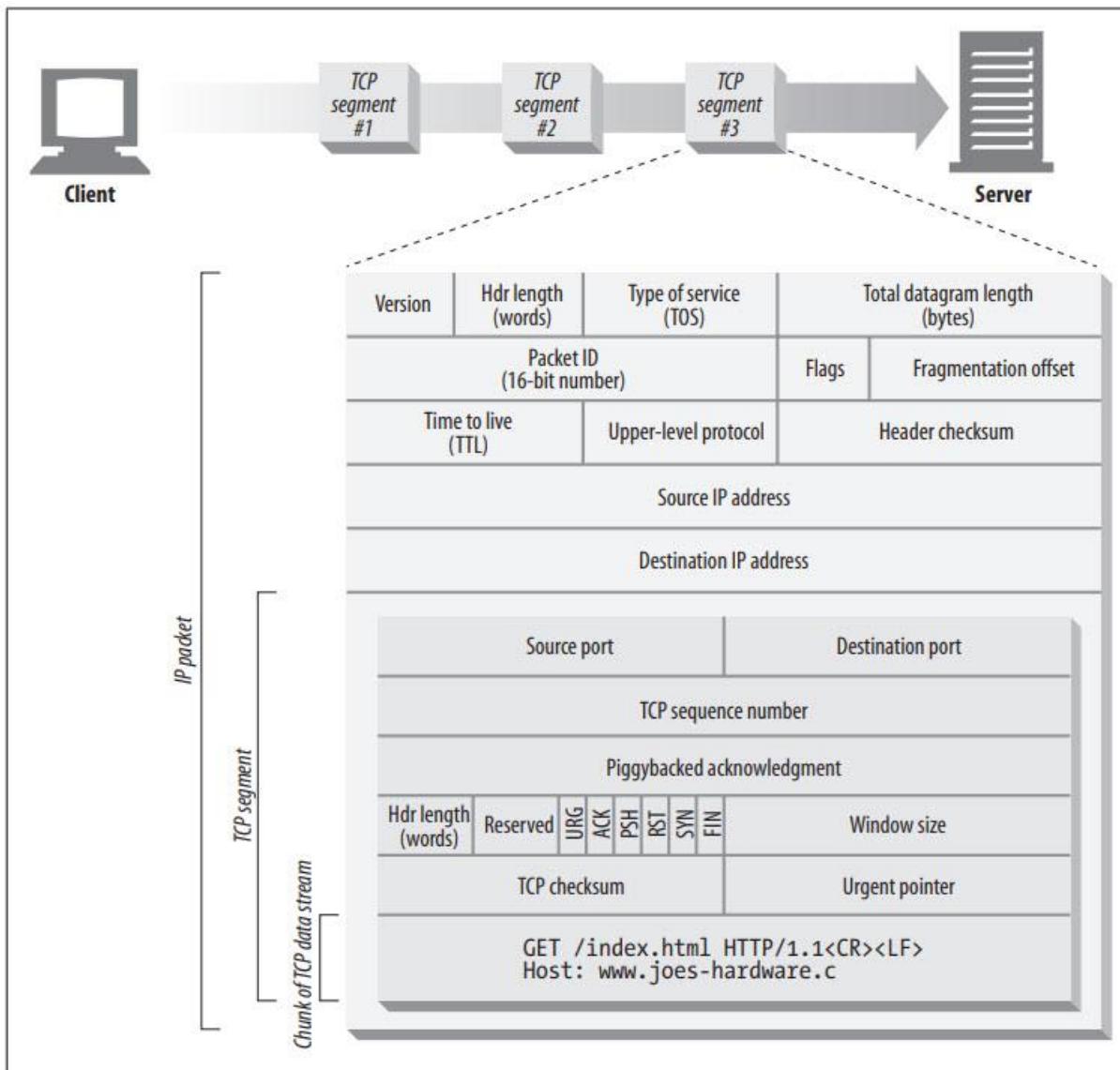
هر بخش TCP توسط یک بسته IP از یک آدرس IP به آدرس IP دیگر منتقل می‌شود. هر یک از این بسته‌های IP شامل موارد زیر هستند:

- هدر بسته IP (معمولًاً ۲۰ بایت)
- هدر TCP (معمولًا ۲۰ بایت)
- تکه ای از داده‌های TCP (۰ بایت یا بیشتر)





هدر IP حاوی آدرس‌های IP مبدأ و مقصد، اندازه و سایر بخش‌ها است. هدر قطعه TCP شامل شماره پورت TCP و مقادیر عددی است که برای ترتیب داده‌ها و بررسی یکپارچگی استفاده می‌شود.



Keeping TCP Connections Straight

یک کامپیوتر ممکن است در هر زمان چندین اتصال TCP باز داشته باشد. تمام این اتصالات را مستقیماً از طریق شماره پورت حفظ می‌کند.

شماره پورت‌ها مانند شماره داخلی تلفن کارمندان هستند. همانطور که شماره تلفن اصلی یک شرکت شما را به میز پذیرش رسانده و شماره داخلی شما را به کارمند مناسب می‌رساند، آدرس IP شما را به رایانه مناسب و شماره پورت شما را به برنامه مناسب می‌رساند. اتصال TCP با چهار مقدار متمایز می‌شود:



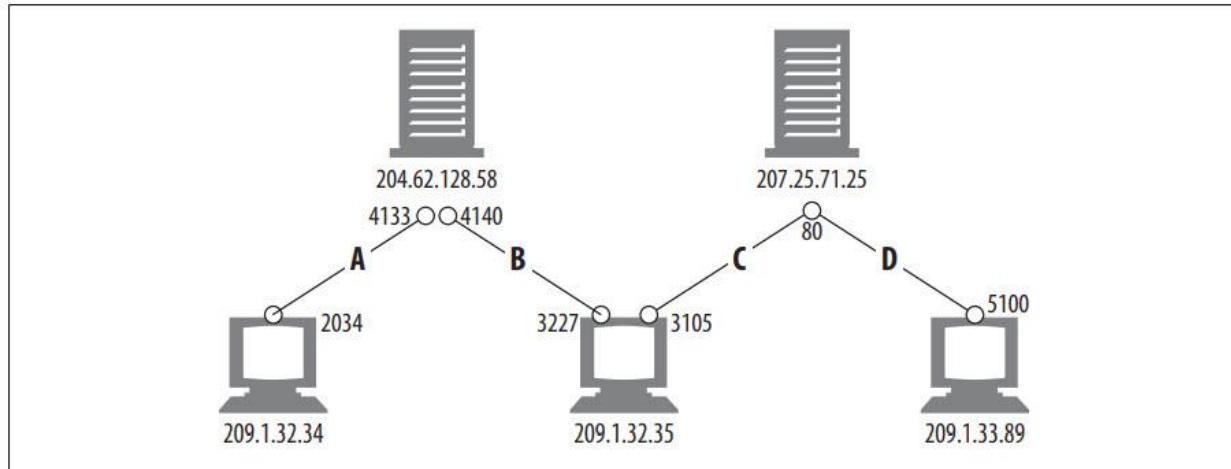


<source-IP-address, source-port, destination-IP-address, destination-port>

این چهار مقدار با هم به طور منحصر به فردی یک اتصال را تعریف می‌کنند. دو اتصال TCP مختلف، مجاز به داشتن مقادیر یکسان برای هر چهار مؤلفه آدرس نیستند (اما اتصالات مختلف می‌توانند مقادیر یکسانی برای برخی از مؤلفه‌ها داشته باشند).

در شکل زیر، چهار اتصال وجود دارد: A، B، C و D.

Connection	Source IP address	Source port	Destination IP address	Destination port
A	209.1.32.34	2034	204.62.128.58	4133
B	209.1.32.35	3227	204.62.128.58	4140
C	209.1.32.35	3105	207.25.71.25	80
D	209.1.33.89	5100	207.25.71.25	80



توجه داشته باشید که برخی از اتصالات دارای شماره پورت مقصد یکسانی هستند (C و D هر دو دارای پورت مقصد ۸۰ هستند). برخی از اتصالات آدرس IP منبع یکسانی دارند (B و C). برخی از آن‌ها آدرس IP مقصد یکسانی دارند (A و B و C و D). اما هیچ دو اتصال مختلف هر چهار مقدار یکسان را به اشتراک نمی‌گذارند.

Programming with TCP Sockets

سیستم عامل‌ها امکانات مختلفی را برای دستکاری اتصالات TCP خود فراهم می‌کنند. بیایید نگاهی گذران به یک رابط برنامه نویسی TCP بیندازیم تا همه چیز را ملموس کنیم. جدول زیر برخی از رابطه‌های اولیه ارائه شده توسط سوکت‌های API را نشان می‌دهد. این سوکت API تمام جزئیات TCP و IP را از برنامه نویس HTTP پنهان می‌کند. سوکت‌های API اولین بار برای سیستم عامل یونیکس توسعه داده شد، اما انواع آن در حال حاضر تقریباً برای هر سیستم عامل و زبانی موجود است.



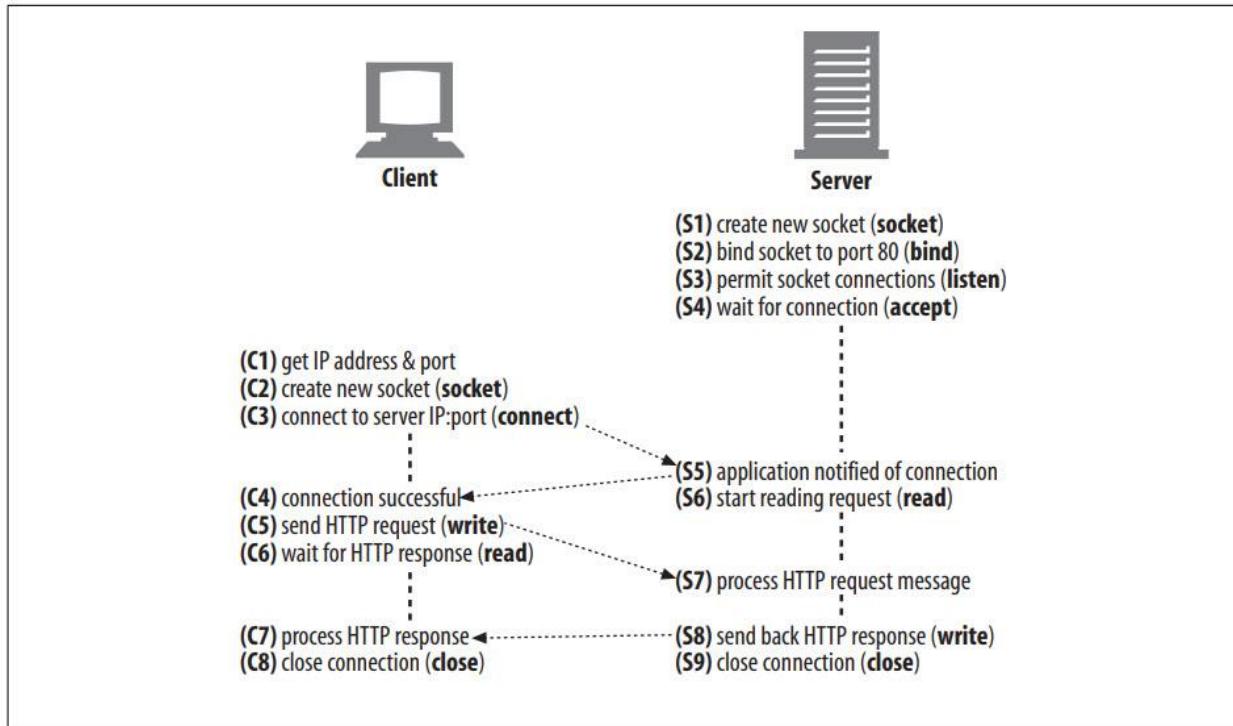


Sockets API call	Description
s = socket(<parameters>)	Creates a new, unnamed, unattached socket.
bind(s, <local IP:port>)	Assigns a local port number and interface to the socket.
connect(s, <remote IP:port>)	Establishes a TCP connection to a local socket and a remote host and port.
listen(s,...)	Marks a local socket as legal to accept connections.
s2 = accept(s)	Waits for someone to establish a connection to a local port.
n = read(s,buffer,n)	Tries to read n bytes from the socket into the buffer.
n = write(s,buffer,n)	Tries to write n bytes from the buffer into the socket.
close(s)	Completely closes the TCP connection.
shutdown(s,<side>)	Closes just the input or the output of the TCP connection.
getsockopt(s, ...)	Reads the value of an internal socket configuration option.
setsockopt(s, ...)	Changes the value of an internal socket configuration option.

Sockets API به شما امکان می‌دهد ساختارهای داده نقطه پایانی TCP ایجاد کنید، این Endpoint ها یا نقاط پایانی را به نقاط پایانی TCP سرور از راه دور متصل کنید و جریان‌های داده را بخوانید و بنویسید. API TCP تمام جزئیات مربوط به دست دادن پروتکل شبکه (Handshaking) و تقسیم بندی و مونتاز مجدد جریان داده TCP به و از بسته‌های IP را پنهان می‌کند.

در بخش‌های پیشین، نشان دادیم که چگونه یک مرورگر وب می‌تواند صفحه وب power-tools.html را از فروشگاه سخت افزار Joe با استفاده از HTTP دانلود کند. شبه کد موجود در شکل زیر، نحوه استفاده از سوکت‌های API را برای بررسی کردن مراحلی که کلاینت و سرور می‌توانند برای اجرای این تراکنش HTTP انجام دهنده ترسیم می‌کند.





ما با وب سرور که منتظر اتصال است شروع می‌کنیم (S4). کلاینت آدرس IP و شماره پورت را از URL تعیین می‌کند و به ایجاد یک اتصال TCP به سرور ادامه می‌دهد (C3). بسته به فاصله سرور، بار روی سرور و ازدحام اینترنت، ایجاد یک اتصال ممکن است کمی طول بکشد.

هنگامی که اتصال برقرار شد، کلاینت درخواست HTTP را ارسال می‌کند (C5) و سرور آن را می‌خواند (S6). هنگامی که سرور کل پیام درخواست را دریافت می‌کند، درخواست را پردازش می‌کند، عمل درخواستی را انجام می‌دهد (S7) و داده‌ها را به کلاینت ارسال می‌کند. کلاینت آن را می‌خواند (C6) و داده‌های پاسخ را پردازش می‌کند (C7).

TCP Performance Considerations

از آنجایی که HTTP مستقیماً بر روی TCP لایه بندی می‌شود، عملکرد تراکنش‌های HTTP به شدت به عملکرد TCP plumbing زیربنایی بستگی دارد. این بخش برخی از ملاحظات عملکرد قابل توجه این اتصالات TCP را بر جسته می‌کند. با درک برخی از ویژگی‌های اصلی عملکرد TCP، بهتر از ویژگی‌های بهینه‌سازی اتصال HTTP قادرانی می‌کنید و می‌توانید برنامه‌های HTTP با کارایی بالاتر را طراحی و پیاده‌سازی کنید.

این بخش نیاز به درک کمی از جزئیات داخلی پروتکل TCP دارد. اگر به جزئیات ملاحظات عملکرد «HTTP Connection Handling» (یا با آن راحت هستید، به راحتی به «TCP علاقه مند نیستید

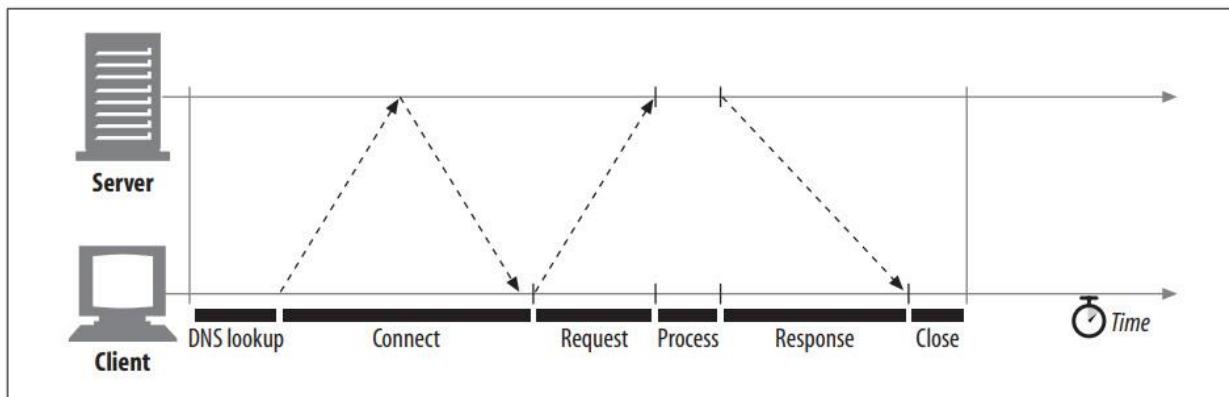




بروید. از آنجایی که TCP یک موضوع پیچیده است، ما می‌توانیم در اینجا فقط یک نمای کلی از عملکرد TCP را بررسی کنیم. شکل زیر تأخیرهای اصلی اتصال، انتقال و پردازش را برای یک تراکنش HTTP نشان می‌دهد.

HTTP Transaction Delays

بیایید تور عملکرد TCP خود را با بررسی تأخیرهای شبکه در جریان درخواست HTTP شروع کنیم. شکل زیر تأخیرهای اصلی اتصال، انتقال و پردازش را برای یک تراکنش HTTP نشان می‌دهد.



توجه داشته باشید که زمان پردازش تراکنش در مقایسه با زمان مورد نیاز برای راه اندازی اتصالات TCP و انتقال پیام‌های درخواست و پاسخ می‌تواند بسیار کم باشد. تا زمانی که کلاینت یا سرور بیش از حد بارگذاری شود (یا منابع پویا پیچیده ای را اجرا کند، بیشتر تأخیرهای HTTP ناشی از تأخیر شبکه TCP است) (overloaded).

چندین دلیل احتمالی برای تأخیر در تراکنش HTTP وجود دارد:

۱. یک کلاینت ابتدا باید آدرس IP و شماره پورت سرور وب را از URI تعیین کند. اگر نام میزبان در اختیار بازدید نشده است، ممکن است دهها ثانیه طول بکشد تا نام میزبان از یک URI به آدرس IP با استفاده از زیرساخت DNS تبدیل شود.

۲. در مرحله بعد، کلاینت یک درخواست اتصال TCP به سرور ارسال می‌کند و منتظر می‌ماند تا سرور پاسخ پذیرش اتصال را ارسال کند. تأخیر راه اندازی اتصال برای هر اتصال TCP جدید رخ می‌دهد. این معمولاً حداقل یک یا دو ثانیه طول می‌کشد، اما زمانی که صدها تراکنش HTTP انجام می‌شود، می‌تواند به سرعت جمع شود.

۳. هنگامی که اتصال برقرار شد، کلاینت درخواست HTTP را از طریق TCP Pipe تازه تاسیس ارسال می‌کند. وب سرور پیام درخواست را از اتصال TCP به هنگام رسیدن داده‌ها می‌خواند و درخواست را پردازش می‌کند. مدت زمانی طول می‌کشد تا پیام درخواست از طریق اینترنت عبور کند و توسط سرور پردازش شود.





۴. سپس وب سرور پاسخ HTTP را باز می‌گرداند که این نیز کمی زمان می‌برد.

بزرگی این تاخیرهای شبکه TCP به سرعت سخت افزار، بار شبکه و سرور، اندازه پیام‌های درخواست و پاسخ و فاصله بین کلاینت و سرور بستگی دارد. تاخیرها نیز به طور قابل توجهی تحت تأثیر پیچیدگی‌های فنی پروتکل TCP قرار دارند.

Performance Focus Areas

در ادامه این بخش برخی از متدائل‌ترین تاخیرهای مرتبط با TCP که بر برنامه‌نویسان HTTP تأثیر می‌گذارند، از جمله علل و تأثیرات عملکردی را بیان می‌کند:

TCP Handshake مربوط به راه اندازی اتصال

TCP slow-start کنترل ازدحام با

Nagle الگوریتم برای تجمیع داده‌ها

piggyback الگوریتم تایید تاخیر TCP برای تایید

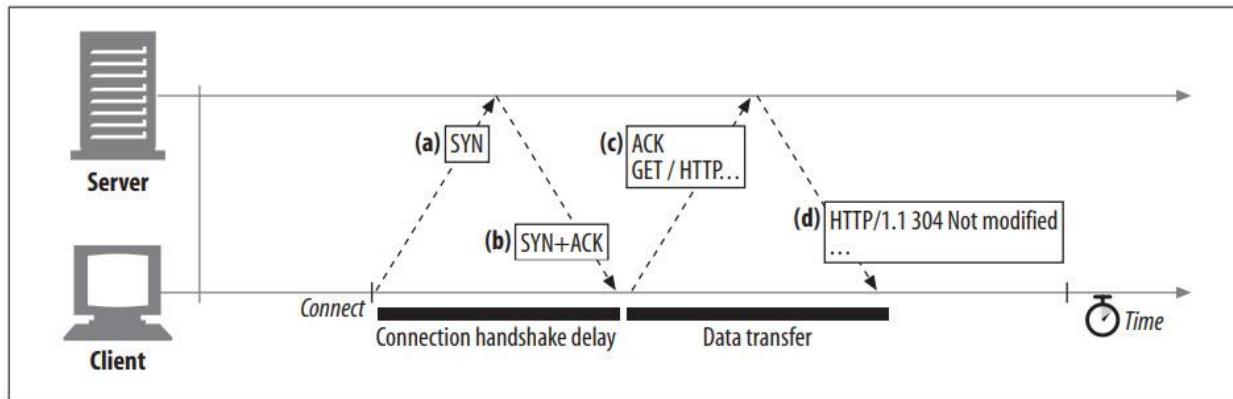
TIME_WAIT و Port Exhaustion تأخیر

اگر در حال نوشتن نرم افزار HTTP با کارایی بالا هستید، باید هر یک از این عوامل را درک کنید. اگر به این سطح از بهینه‌سازی عملکرد نیاز ندارید، به راحتی از این موارد رد شوید.

TCP Connection Handshake Delays

هنگامی که یک اتصال TCP جدید راه اندازی می‌کنید، حتی قبل از ارسال هرگونه داده، نرم افزار TCP یک سری بسته IP را برای مذاکره در مورد شرایط اتصال مبادله می‌کند (شکل زیر را ببینید). اگر از اتصالات برای انتقال داده‌های کوچک استفاده شود، این مبادلات می‌توانند به طور قابل توجهی عملکرد HTTP را کاهش دهند.





در اینجا مراحل TCP Connection Handshake آمده است:

۱. برای درخواست اتصال TCP جدید، کلاینت یک بسته TCP کوچک (معمولاً ۴۰ تا ۶۰ بایت) به سرور ارسال می‌کند. بسته دارای یک مجموعه Flag خاص "SYN" است، به این معنی که یک درخواست اتصال است. این موضوع در بخش a از شکل بالا نشان داده شده است.

۲. اگر سرور اتصال را پذیرد، برخی از پارامترهای اتصال را محاسبه می‌کند و یک بسته TCP را با دو Flag "SYN" و "ACK" تنظیم می‌کند که نشان می‌دهد درخواست اتصال پذیرفته شده است (بخش b از شکل بالا را ببینید).

۳. در نهایت، مشتری یک تأییدیه را به سرور ارسال می‌کند و به او اطلاع می‌دهد که اتصال با موفقیت برقرار شده است (بخش c از شکل بالا را ببینید). TCP Stack های مدرن به کلاینت این امکان را می‌دهند که داده‌ها را در این بسته Acknowledgment ارسال کند.

برنامه نویس HTTP هرگز این بسته‌ها را نمی‌بیند – آن‌ها به صورت نامرئی توسط نرم افزار TCP/IP مدیریت می‌شوند. تنها چیزی که برنامه نویس HTTP می‌بیند تاخیر در ایجاد یک اتصال TCP جدید است.

Handshake مریبوط به SYN/SYN+ACK (بخش a و b شکل بالا) زمانی که تراکنش‌های HTTP داده‌های زیادی را مبادله نمی‌کنند، تاخیر قابل اندازه گیری ایجاد می‌کند. بسته TCP ACK اتصال (بخش c از شکل بالا) اغلب به اندازه‌ای بزرگ است که بتواند کل پیام درخواست HTTP را حمل کند و بسیاری از پیام‌های پاسخ سرور HTTP در یک بسته IP قرار می‌گیرند (مثلاً زمانی که پاسخ یک فایل HTML کوچک از یک بخش گرافیکی یا یک پاسخ 304 Not Modified به درخواست کش مرورگر است).

نتیجه نهایی این است که تراکنش‌های کوچک HTTP ممکن است ۵۰ درصد یا بیشتر از زمان خود را صرف راه اندازی TCP کنند.





Delayed Acknowledgments

از آنجایی که اینترنت خود تحویل بسته اعتماد قابل تضمین نمی‌کند (روترهای اینترنت می‌توانند بسته‌ها را به میل خود در صورت بارگذاری بیش از حد از بین ببرند)، TCP طرح تأیید خود (Acknowledgment) را برای تضمین تحویل موفقیت آمیز داده‌ها پیاده سازی می‌کند.

هر بخش TCP یک Data-Integrity Checksum و یک Sequence Number دریافت می‌کند. گیرنده هر سگمنت بسته‌های Acknowledgment کوچک را زمانی به فرستنده بر می‌گرداند که بخش‌ها دست نخورده دریافت شوند. اگر یک فرستنده در یک بازه زمانی مشخص، Acknowledgment ای دریافت نکند، فرستنده نتیجه می‌گیرد که بسته از بین رفته یا خراب شده است و داده‌ها را دوباره ارسال می‌کند.

از آنجایی که Acknowledgment‌ها کوچک هستند، TCP به آن‌ها اجازه می‌دهد تا بسته‌های داده خروجی را که در همان جهت حرکت می‌کنند، piggyback کنند. با ترکیب Acknowledgment‌های برگشتی با بسته‌های داده خروجی، TCP می‌تواند استفاده موثرتری از شبکه داشته باشد.

بسیاری از TCP Stack‌ها برای افزایش شанс یافتن یک بسته داده در یک جهت توسط یک Acknowledgment ، الگوریتم "Delayed Acknowledgment" را پیاده سازی می‌کنند. تأییدهای تأخیری یا Delayed Acknowledgment، خروجی را برای یک پنجره زمانی معین (معمولًاً ۱۰۰ تا ۲۰۰ میلی ثانیه) در یک بافر نگه می‌دارند و به دنبال یک بسته داده خروجی می‌گردند که روی آن Piggyback باشد. اگر هیچ بسته داده خروجی در آن زمان وارد نشود، Acknowledgment در بسته خود ارسال می‌شود.

متأسفانه، رفتار دووجهی درخواست-پاسخ HTTP، احتمال وقوع Piggybacking را کاهش می‌دهد. غالباً غیرفعال، تاخیرهای قابل توجهی را معرفی می‌کنند. بسته به سیستم عامل خود، ممکن است بتوانید الگوریتم Delayed Acknowledgment را تنظیم یا غیرفعال کنید.

قبل از اینکه پارامترهای TCP Stack خود را تغییر دهید، مطمئن شوید که می‌دانید چه کاری انجام می‌دهید. الگوریتم‌های داخل TCP برای محافظت از اینترنت در برابر برنامه‌های کاربردی با طراحی ضعیف معرفی شدند. اگر پیکربندی‌های TCP را تغییر می‌دهید، کاملاً مطمئن باشید که برنامه شما مشکلاتی را ایجاد نمی‌کند.

TCP Slow Start

عملکرد انتقال داده TCP به سن (age) اتصال TCP نیز بستگی دارد. اتصالات TCP در طول زمان، خود را تنظیم می‌کنند، در ابتدا حداکثر سرعت اتصال را محدود نموده و در طول زمان با انتقال موفقیت آمیز





داده‌ها، سرعت را افزایش می‌دهند. این تنظیم TCP با نام slow start شناخته می‌شود و برای جلوگیری از overloading ناگهانی و ازدحام اینترنت استفاده می‌شود.

TCP slow start تعداد بسته‌هایی را که یک نقطه پایانی TCP می‌تواند در هر لحظه در حال پرواز داشته باشد کاهش می‌دهد. به زبان ساده، هر بار که یک بسته با موفقیت دریافت می‌شود، فرستنده اجازه ارسال دو بسته دیگر را می‌گیرد. اگر یک تراکنش HTTP مقدار زیادی داده برای ارسال داشته باشد، نمی‌تواند همه بسته‌ها را یکجا ارسال کند و باید یک بسته ارسال شده و منتظر تایید باشد. سپس می‌تواند دو بسته ارسال کند که هر کدام باید تایید شوند، که اجازه می‌دهد چهار بسته دیگر ارسال شود و این موضوع به همین شکل ادامه پیدا می‌کند. به این قابلیت در اصطلاح opening the congestion window گفته می‌شود.

به دلیل این ویژگی کنترل تراکم، اتصالات جدید‌کنترل‌تر از اتصالات «تنظیم شده» هستند که قبلًا مقدار متوسطی از داده را رد و بدل کرده‌اند. از آنجایی که اتصالات تنظیم شده سریع‌تر هستند، HTTP دارای امکاناتی است که به شما امکان می‌دهد از اتصالات موجود استفاده مجدد کنید. ما در مورد این "اتصالات پایدار" HTTP در ادامه همین فصل صحبت خواهیم کرد.

Nagle's Algorithm and TCP_NODELAY

TCP دارای یک رابط جریان داده (data stream interface) است که به برنامه‌ها اجازه می‌دهد تا داده‌ها را با هر اندازه‌ای به پشته TCP - حتی یک بایت در یک زمان، پخش کنند! اما از آنجایی که هر بخش TCP حداقل ۴۰ بایت Flag و هدرها را حمل می‌کند، اگر TCP تعداد زیادی بسته حاوی مقادیر کمی داده ارسال کند، عملکرد شبکه می‌تواند به شدت کاهش یابد.

الگوریتم Nagle (که به نام سازنده آن، John Nagle نامگذاری شده است) سعی می‌کند قبل از ارسال بسته، مقدار زیادی از داده‌های TCP را جمع آوری کند و به کارایی شبکه کمک کند. این الگوریتم در RFC 896 "Congestion Control in IP/TCP Internetworks" توضیح داده شده است.

الگوریتم Nagle از ارسال بخش‌هایی که اندازه کامل ندارند (یک بسته با حداکثر اندازه حدود ۱۵۰۰ بایت در یک LAN یا چند صد بایت در سراسر اینترنت است) جلوگیری می‌کند. الگوریتم Nagle به شما این امکان را می‌دهد که یک بسته غیر کامل را فقط در صورتی ارسال کنید که همه بسته‌های دیگر تایید شده باشند. اگر بسته‌های دیگر هنوز in flight باشند، داده‌های جزئی بافر می‌شوند. این داده‌های بافر تنها زمانی ارسال می‌شوند که بسته‌های معلق تأیید شوند یا زمانی که بافر اطلاعات کافی برای ارسال یک بسته کامل را جمع‌آوری کرده باشد.





الگوریتم Nagle منجر به ایجاد چندین مشکل عملکردی HTTP می‌شود. اول، پیام‌های کوچک HTTP ممکن است یک بسته را پر نکنند، بنابراین ممکن است در انتظار داده‌های اضافی که هرگز نخواهند رسید، به تعویق بیفتد. دوم، الگوریتم Nagle با acknowledgment های غیرفعال تعامل ضعیفی دارد - الگوریتم Nagle ارسال داده‌ها را تا رسیدن یک تأیید متوقف می‌کند، اما خود تأیید ۱۰۰ تا ۲۰۰ میلی‌ثانیه توسط الگوریتم Delayed Acknowledgment.

برنامه‌های HTTP اغلب الگوریتم Nagle را برای بهبود عملکرد با تنظیم پارامتر TCP_NODELAY در پسته های خود غیرفعال می‌کنند. اگر این کار را انجام می‌دهید، باید مطمئن شوید که تکه‌های بزرگ داده را در TCP می‌نویسید تا انبوهی از بسته‌های کوچک ایجاد نکنید.

TIME_WAIT Accumulation and Port Exhaustion

TIME_WAIT Port Exhaustion یک مشکل عملکرد جدی است که بر معیار عملکرد تأثیر می‌گذارد، اما در استقرار واقعی نسبتاً غیر معمول است.

هنگامی که یک نقطه پایانی TCP یک اتصال را می‌بندد، یک بلوك کنترل کوچک را در حافظه نگه می‌دارد که آدرس‌های IP و شماره پورت اتصال اخیراً بسته شده را ضبط می‌کند. این اطلاعات برای مدت کوتاهی، معمولاً حدود دو برابر حداقل طول عمر بخش تخمینی (به نام "2MSL"، اغلب دو دقیقه*) نگهداری می‌شود تا اطمینان حاصل شود که یک اتصال TCP جدید با همان آدرس‌ها و شماره پورت‌ها در این مدت ایجاد نمی‌شود. این کار از تزریق تصادفی بسته‌های تکراری از اتصال قبلی به یک اتصال جدید که آدرس‌ها و شماره پورت‌های یکسانی دارد جلوگیری می‌کند. در عمل، این الگوریتم از ایجاد، بسته شدن و ایجاد مجدد دو اتصال با آدرس‌های IP و شماره پورت عیناً یکسان در عرض دو دقیقه جلوگیری می‌کند.

نکته: مقدار 2MSL دو دقیقه تاریخی است. مدت‌ها پیش، زمانی که روتراها بسیار کندر بودند، تخمین زده می‌شد که یک کپی تکراری از یک بسته می‌تواند تا یک دقیقه قبل از نابودی در اینترنت در صف باقی بماند. امروزه حداقل طول عمر سگمنت بسیار کمتر است.

در روتراهای با سرعت بالاتر امروزی، نمایش یک بسته تکراری در آستانه سرور، چند دقیقه پس از بسته شدن اتصال، بسیار بعید است. برخی از سیستم عامل‌ها 2MSL را روی یک مقدار کوچکتر تنظیم می‌کنند، اما مراقب نادیده گرفته شدن این مقدار باشید. بسته‌ها تکراری می‌شوند و اگر بسته تکراری از اتصال گذشته در جریان جدیدی با مقادیر اتصال یکسان قرار گیرد، داده‌های TCP خراب می‌شوند.





تأخیر بسته شدن اتصال 2MSL به طور معمول مشکلی ایجاد نمی‌کند، اما در موقعیت‌های بنچمارک امکان بروز مشکل وجود دارد. معمول است که فقط یک یا چند کامپیوتر load-generation آزمایشی تحت آزمایش بنچمارک، به یک سیستم متصل می‌شوند که تعداد آدرس‌های IP کلاینت را که به سرور متصل می‌شوند محدود می‌کند. علاوه بر این، سرور معمولاً به پورت TCP پیشفرض HTTP، ۸۰ گوش می‌دهد. این شرایط، در زمانی که شماره‌های پورت برای استفاده مجدد تا TIME_WAIT مسدود می‌شوند، ترکیب‌های موجود مقادیر اتصال را محدود می‌کند.

در یک موقعیت پاتولوژیک با یک کلاینت و یک سرور، از چهار مقداری که یک اتصال TCP را تشکیل می‌دهند (IP مبدا، پورت مبدا، IP مقصد و پورت مقصد)، سه مورد از آن‌ها ثابت هستند و فقط پورت مبدا برای تغییر آزاد است.

هر بار که کلاینت به سرور متصل می‌شود، یک پورت مبدا جدید دریافت می‌کند تا یک اتصال منحصر به فرد داشته باشد. اما از آنجایی که تعداد محدودی از پورت‌های منبع در دسترس هستند (مثلاً ۶۰۰۰۰) و هیچ اتصالی را نمی‌توان برای 2MSL ثانیه (مثلاً ۱۲۰ ثانیه) دوباره استفاده کرد، این امر نرخ اتصال را به $120 / 60000 = 500$ تراکنش در ثانیه محدود می‌کند.

اگر به بهینه‌سازی ادامه می‌دهید و سرعت سرور شما از حدود ۵۰۰ تراکنش در ثانیه بیشتر نمی‌شود، مطمئن شوید که TIME_WAIT Port Exhaustion را تجربه نمی‌کنید. می‌توانید این مشکل را با استفاده از ماشین‌های load-generator کلاینت بیشتر یا اطمینان از چرخش کلاینت و سرور از طریق چندین آدرس IP مجازی برای افزودن ترکیب‌های اتصال بیشتر، برطرف کنید.

حتی اگر از مشکلات Port Exhaustion رنج نمی‌برید، مراقب تعداد زیادی اتصال باز یا تعداد زیادی بلوک کنترلی برای اتصال در حالت انتظار اختصاص داده شده باشید. برخی از سیستم‌عامل‌ها زمانی که اتصالات باز یا بلوک‌های کنترلی متعددی وجود دارند، سرعت‌شان به شدت کاهش می‌یابد.

HTTP Connection Handling

ماقی این فصل، فناوری HTTP را برای دستکاری و بهینه سازی اتصالات توضیح می‌دهد. ما با هدر اتصال HTTP شروع می‌کنیم، بخش مهمی از مدیریت اتصال HTTP که اغلب اشتباہ فهمیده می‌شود. سپس در مورد تکنیک‌های بهینه سازی اتصال HTTP صحبت خواهیم کرد.





The Oft-Misunderstood Connection Header

HTTP به زنجیره‌ای از واسطه‌های HTTP بین کلاینت و سرور اصلی (پراکسی‌ها، کش‌ها و غیره) اجازه می‌دهد. پیام‌های HTTP از طریق دستگاه‌های واسطه به سرور مبدأ (یا بر عکس) به صورت hop by hop از کلاینت ارسال می‌شوند.

در برخی موارد، دو برنامه HTTP مجاور ممکن است بخواهند مجموعه‌ای از گزینه‌ها را برای اتصال مشترک خود اعمال کنند. فیلد هدر اتصال HTTP دارای فهرستی از Connection Tokens جدا شده با کاما است که گزینه‌هایی را برای اتصال مشخص می‌کند که به سایر اتصالات منتشر نمی‌شوند. به عنوان مثال، اتصالی که باید پس از ارسال پیام بعدی بسته شود را می‌توان با Connection: close نشان داد.

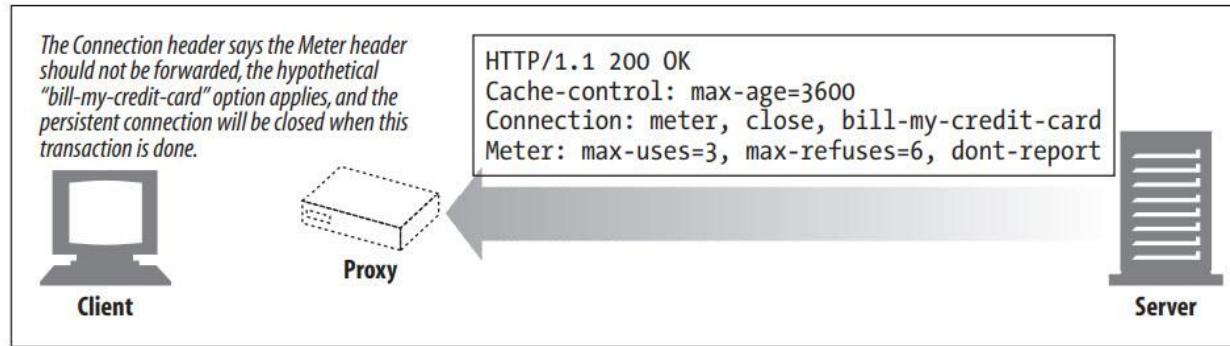
هدر Connection گاهی گیج کننده است، زیرا می‌تواند سه نوع مختلف توکن را حمل کند:

- نام‌های فیلد هدر HTTP، فهرست‌بندی هدرهای مربوط به این اتصال
- مقادیر توکن دلخواه، توصیف گزینه‌های غیر استاندارد برای این اتصال
- مقدار Close، نشان می‌دهد که اتصال دائمی پس از اتمام بسته خواهد شد.

اگر یک Connection Token حاوی نام یک فیلد هدر HTTP باشد، آن فیلد هدر حاوی اطلاعات مربوط به اتصال است و نباید ارسال شود. هر فیلد هدر فهرست شده در هدر Connection باید قبل از ارسال پیام حذف شود.

قرار دادن نام هدر hop-by-hop در هدر Connection به عنوان "protecting the header" شناخته می‌شود، زیرا هدر Connection از ارسال تصادفی هدر محلی محافظت می‌کند. یک مثال در شکل زیر نشان داده شده است.

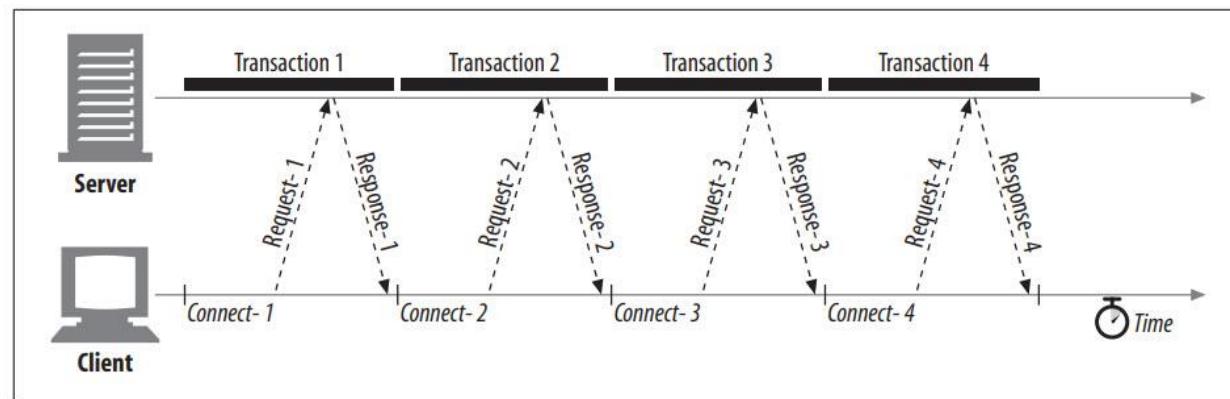




هنگامی که یک برنامه HTTP پیامی با هدر Connection دریافت می‌کند، گیرنده تمام گزینه‌های درخواست شده توسط فرستنده را تجزیه و اعمال می‌کند. سپس هدر Connection و تمام هدرهای فهرست شده در هدر hop-by-hop را قبل از ارسال پیام به hop بعدی حذف می‌کند. علاوه بر این، تعدادی هدر Connection وجود دارد که ممکن است به عنوان مقادیر یک هدر Connection فهرست نشده باشند، اما نباید پروکسی شوند. این‌ها عبارتند از Upgrade، Transfer-Encoding، Proxy-Connection، Proxy-Authenticate و Serial Transaction Delays.

Serial Transaction Delays

اگر اتصالات، ساده مدیریت شوند، تأخیرهای عملکرد TCP می‌توانند افزایش پیدا کنند. به عنوان مثال، فرض کنید یک صفحه وب با سه تصویر embedded شده دارید. مرورگر شما برای نمایش این صفحه باید چهار تراکنش HTTP صادر کند: یکی برای HTML سطح بالا و سه مورد برای تصاویر embedded شده. اگر هر تراکنش نیاز به یک اتصال جدید داشته باشد، تأخیرهای اتصال و slow-start می‌توانند اضافه شوند.



علاوه بر تاخیر واقعی که بارگذاری سریال تحمیل می‌کند، زمانی که یک تصویر در حال بارگذاری است و هیچ اتفاقی در بقیه صفحه نمی‌افتد، در ک روان‌شناختی کندی نیز وجود دارد. کاربران ترجیح می‌دهند چندین تصویر همزمان بارگذاری شوند. (این درست است که حتی اگر بارگذاری چند تصویر به طور همزمان کندر از بارگذاری تصاویر در یک زمان باشد! کاربران اغلب بارگذاری چند تصویر را سریعتر می‌دانند.)





یکی دیگر از معایب بارگذاری سریال این است که برخی از مرورگرها نمی‌توانند چیزی را روی صفحه نمایش دهند تا زمانی که اشیاء به اندازه کافی بارگذاری شوند، زیرا آن‌ها اندازه اشیاء را تا زمانی که بارگذاری نشده اند نمی‌دانند و ممکن است برای تصمیم گیری در مورد مکان قرار دادن آن‌ها به اطلاعات سایز نیاز داشته باشند. در این شرایط، مرورگر ممکن است پیشرفت خوبی در بارگذاری سریال اشیا داشته باشد، اما کاربر ممکن است با یک صفحه سفید خالی مواجه شود، غافل از اینکه اصلاً پیشرفتی در حال انجام است.

چندین تکنیک فعلی و نوظهور برای بهبود عملکرد اتصال HTTP در دسترس هستند. بخش‌های بعدی چهار تکنیک از این قبیل را مورد بحث قرار می‌دهد:

TCP: درخواست‌های HTTP همزمان در چندین اتصال Parallel connections

connect/close: استفاده مجدد از اتصالات TCP برای حذف تاخیرهای Persistent connections

Pipelined connections: درخواست‌های HTTP همزمان در یک اتصال TCP مشترک

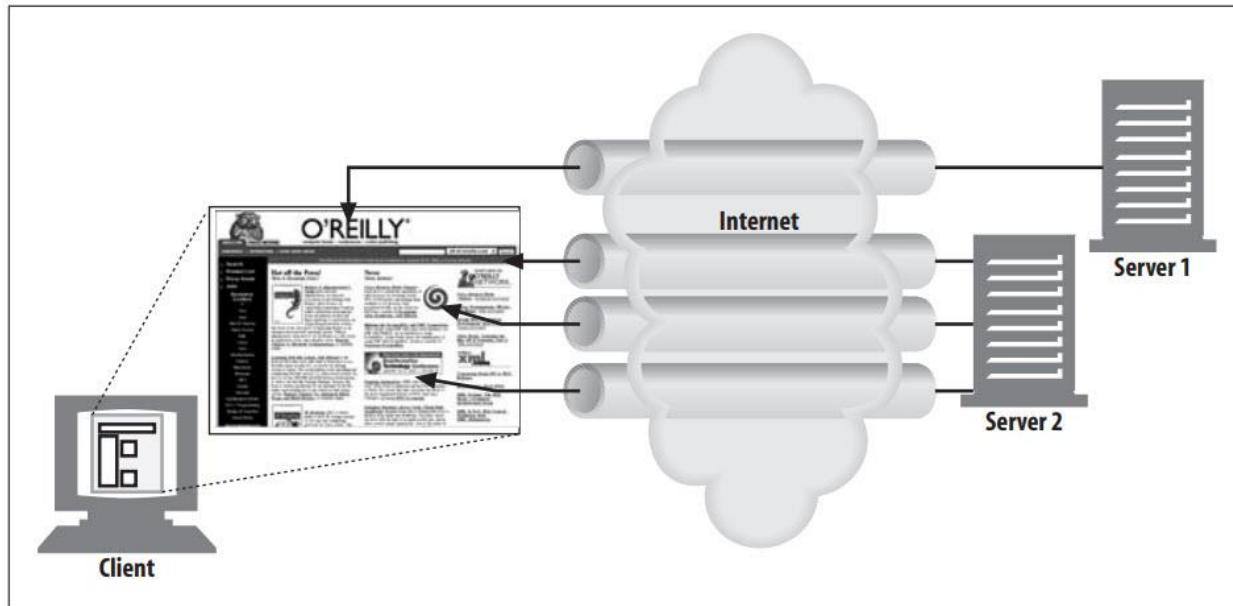
Multiplexed connections (experimental): در هم آمیختن تکه‌های درخواست‌ها و پاسخ‌ها

Parallel Connections

همانطور که قبلاً اشاره کردیم، یک مرورگر می‌تواند ساده لوحانه هر شی جاسازی شده را با درخواست کامل صفحه HTML اصلی، سپس اولین شی جاسازی شده، سپس دومین شی جاسازی شده و غیره پردازش کند. اما این خیلی کند است!

HTTP به مشتریان اجازه می‌دهد تا چندین اتصال را باز کنند و چندین تراکنش HTTP را به صورت موازی انجام دهند، همانطور که در شکل زیر نشان داده شده است. در این مثال، چهار تصویر تعبیه شده به صورت موازی بارگذاری می‌شوند و هر تراکنش اتصال TCP خود را دریافت می‌کند.



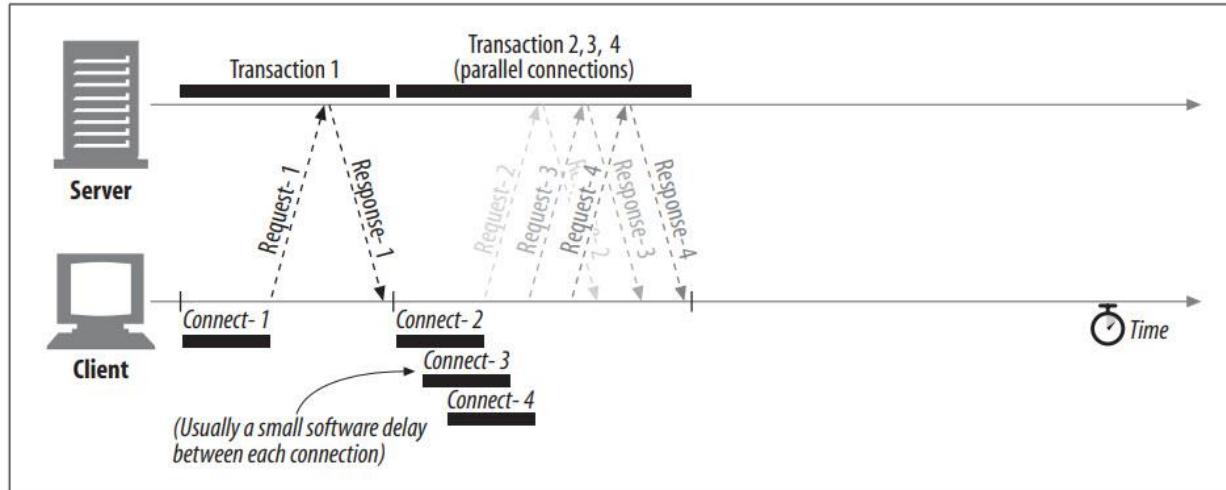


Parallel Connections May Make Pages Load Faster

صفحات ترکیبی متشكل از اشیاء جاسازی شده ممکن است سریعتر بارگذاری شوند اگر از محدودیت زمان و پهنهای باند مرده یک اتصال استفاده کنند. تأخیرها می‌توانند همپوشانی داشته باشند و اگر یک اتصال واحد پهنهای باند اینترنت کلاینت را اشباع نکند، پهنهای باند استفاده نشده را می‌توان به بارگیری اشیاء اضافی اختصاص داد.

شکل زیر یک جدول زمانی برای اتصالات موازی را نشان می‌دهد که به طور قابل توجهی سریعتر از حالت مربوط به سریال است. صفحه HTML محصور ابتدا بارگیری می‌شود و سپس سه تراکنش باقیمانده به طور همزمان پردازش می‌شوند، که هر کدام اتصال خاص خود را دارند.





Parallel Connections Are Not Always Faster

اگرچه اتصالات موازی ممکن است سریعتر باشند، اما همیشه سریعتر نیستند. هنگامی که پهنای باند شبکه کلاینت اندک است (به عنوان مثال، مرورگری که از طریق یک مودم ۲۸.۸ کیلوبیت بر ثانیه به اینترنت متصل است)، ممکن است بیشتر زمان صرف انتقال داده شود. در این شرایط، یک تراکنش HTTP به یک سرور سریع، می‌تواند به راحتی تمام پهنای باند مودم موجود را مصرف کند. اگر چندین شی به صورت موازی بارگذاری شوند، هر شی فقط برای این پهنای باند محدود رقابت می‌کند، بنابراین هر شی به نسبت کندرت بارگذاری می‌شود و مزیت عملکردی کمی به همراه خواهد داشت.

همچنین تعداد زیادی از اتصالات باز می‌توانند حافظه زیادی را مصرف نموده و مشکلات عملکردی خود را ایجاد نمایند. صفحات وب پیچیده ممکن است دهها یا صدها شی جاسازی شده داشته باشند. کلاینت‌ها ممکن است بتوانند صدھا اتصال را باز کنند، اما تعداد کمی از سرورهای وب مایل به انجام این کار هستند، زیرا اغلب درخواست‌های بسیاری از کاربران دیگر را همزمان پردازش می‌کنند. صد کاربر همزمان که هر کدام ۱۰۰ اتصال را باز می‌کنند، بار ۱۰۰۰۰ اتصال را بر دوش سرور خواهند گذاشت. این می‌تواند باعث کندی قابل توجه سرور شود. همین وضعیت برای پراکسی‌های **high-load** نیز صادق است.

در عمل، مرورگرها از اتصالات موازی استفاده می‌کنند، اما تعداد کل اتصالات موازی را به تعداد کمی (اغلب چهار) محدود می‌کنند. سرورها برای بستن اتصالات بیش از حد از یک کلاینت خاص آزادند.

Parallel Connections May “Feel” Faster

بسیار خوب، بنابراین اتصالات موازی همیشه باعث نمی‌شود که صفحات سریعتر بارگذاری شوند. اما حتی اگر در واقع سرعت انتقال صفحه را افزایش ندهند، همانطور که قبلًا گفتیم، اتصالات موازی اغلب باعث





می‌شوند کاربران احساس کنند که صفحه سریع‌تر لود می‌شود، زیرا می‌توانند پیشرفت ایجاد شده را با ظاهر شدن چندین شیء مؤلفه به صورت موازی روی صفحه مشاهده کنند.

Persistent Connections

وب کلاینت‌ها اغلب اتصالات خود را به همان سایت باز می‌کنند. به عنوان مثال، اکثر تصاویر جاسازی شده در یک صفحه وب اغلب از یک وب سایت می‌آیند و تعداد قابل توجهی از لینک‌ها به اشیاء دیگر اغلب به همان سایت اشاره می‌کنند. بنابراین، برنامه‌ای که درخواست HTTP را به یک سرور آغاز می‌کند، احتمالاً در آینده نزدیک درخواست‌های بیشتری را به آن سرور ارسال می‌کند. به این ویژگی **site locality** می‌گویند.

به همین دلیل، HTTP/1.1 (و نسخه‌های پیشرفته 1.0) به دستگاه‌های HTTP اجازه می‌دهد تا اتصالات TCP را پس از تکمیل تراکنش‌ها باز نگه دارند و از اتصالات قبلی برای درخواست‌های HTTP آینده استفاده مجدد کنند. اتصالات TCP که پس از اتمام تراکنش‌ها باز نگه داشته می‌شوند، اتصالات پایدار یا **Persistent Connections** نامیده می‌شوند. اتصالات غیر مداوم یا **NonPersistent Connections** پس از هر تراکنش بسته می‌شوند. اتصالات دائمی در سراسر تراکنش‌ها تا زمانی که کلاینت یا سرور تصمیم به بستن آن‌ها بگیرند، باز می‌مانند.

با استفاده مجدد از یک اتصال بیکار و دائمی که از قبل برای سرور مورد نظر باز است، می‌توانید از تنظیم کند اتصال جلوگیری کنید. علاوه بر این، اتصالی که از قبل باز مانده، می‌تواند از مرحله تطبیق تراکم با slow-start جلوگیری کند و امکان انتقال سریع‌تر داده‌ها را فراهم کند.

Persistent Versus Parallel Connections

همانطور که دیدیم، اتصالات موازی می‌توانند سرعت انتقال صفحات ترکیبی را افزایش دهند. اما اتصالات موازی دارای معایبی هستند:

- هر تراکنش یک اتصال جدید را باز و بسته می‌کند که منجر به ایجاد هزینه برای زمان و پهنهای می‌گردد.
- هر اتصال جدید به دلیل شروع کند TCP عملکرد را کاهش می‌دهد.
- محدودیت عملی در تعداد اتصالات موازی باز وجود دارد.

اتصالات دائمی مزایایی نسبت به اتصالات موازی دارند. آن‌ها تأخیر و سربار برقراری اتصال را کاهش می‌دهند، اتصالات را در حالت تنظیم شده نگه می‌دارند و منجر به کاهش تعداد بالقوه اتصالات باز می‌شوند. با این حال، اتصالات دائمی باید با احتیاط مدیریت شوند.



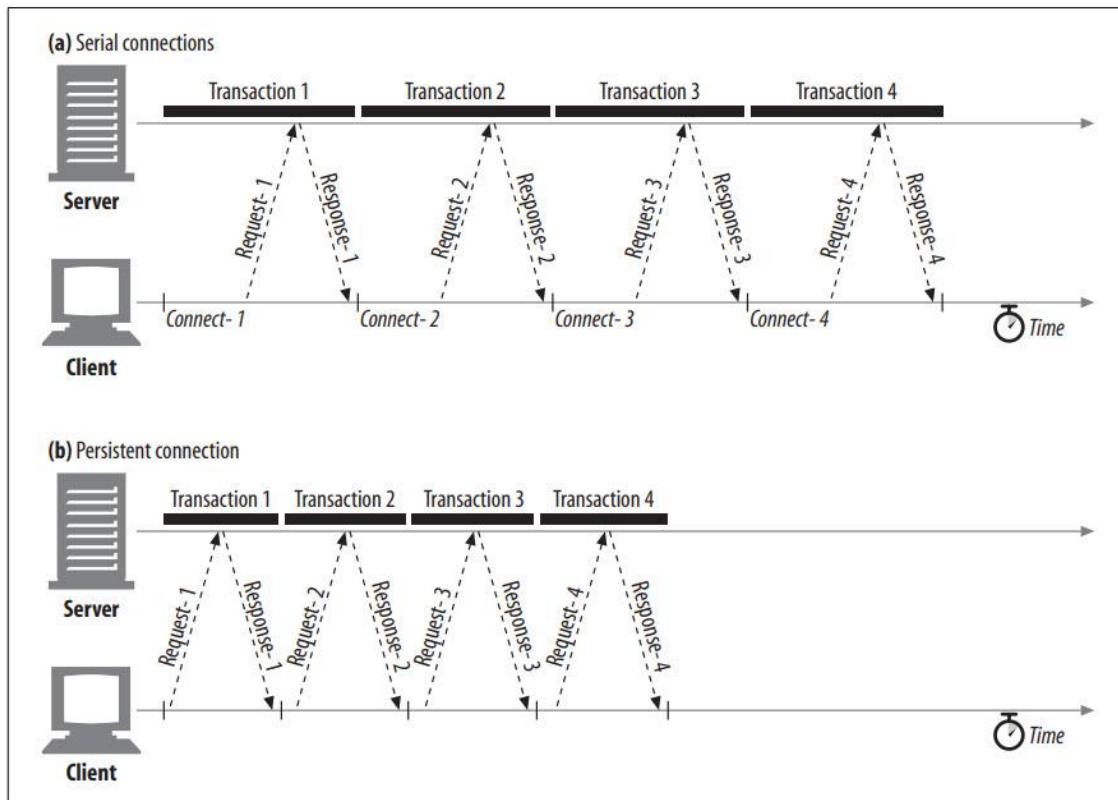
اتصالات پایدار زمانی می‌توانند موثرتر باشند که در ارتباط با اتصالات موازی استفاده شوند. امروزه، بسیاری از برنامه‌های کاربردی وب تعداد کمی از اتصالات موازی را باز می‌کنند که هر کدام پایدار هستند. دو نوع اتصال دائمی وجود دارد:

- اتصالات قدیمی‌تر "HTTP/1.0+ Keep-Alive"
- اتصالات مدرن پایدار HTTP/1.1

HTTP/1.0+ Keep-Alive Connections

بسیاری از مرورگرها و سرورهای HTTP/1.0 (که از حدود سال ۱۹۹۶ شروع شد) برای پشتیبانی از نوع اولیه و آزمایشی اتصالات پایدار به نام اتصالات نگهدارنده یا Keep-Alive Connections توسعه یافته‌اند. این اتصالات پایدار اولیه از برخی مشکلات طراحی مربوط به قابلیت همکاری رنج می‌بردند که در ویرایش‌های بعدی HTTP/1.1 اصلاح شدند، اما بسیاری از کلاینت‌ها و سرورها هنوز از این اتصالات نگهدارنده قبلی استفاده می‌کنند.

برخی از مزایای عملکرد اتصالات نگهدارنده در شکل زیر قابل مشاهده است، که جدول زمانی چهار تراکنش HTTP را از طریق اتصالات سریال در مقابل همان تراکنش‌ها در یک اتصال دائمی مقایسه می‌کند. تایم لاین فشرده شده است زیرا سربار اتصال و بسته حذف شده‌اند.



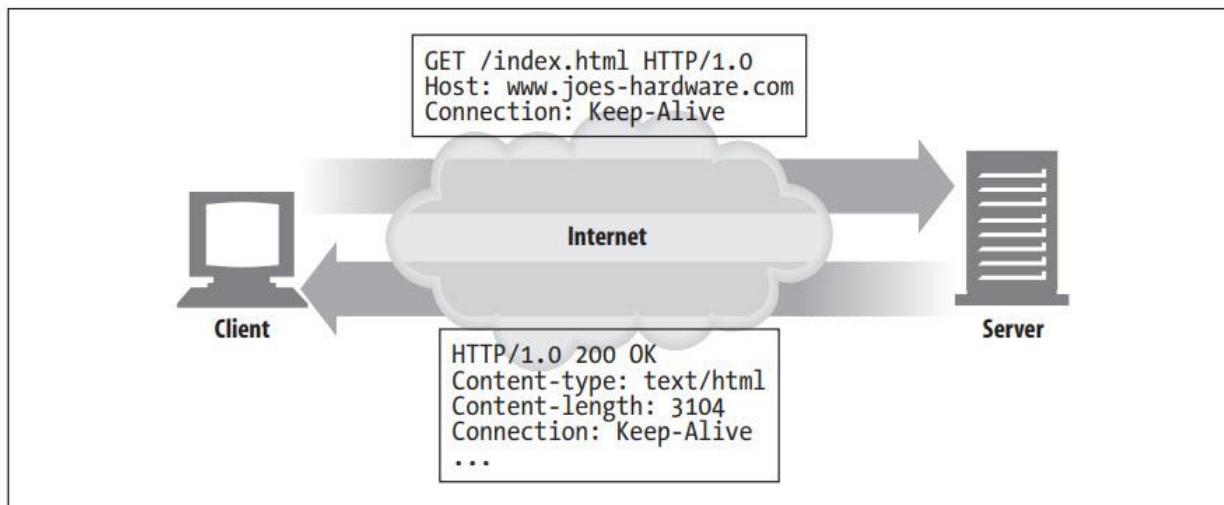


Keep-Alive Operation

Keep-Alive منسخ شده است و دیگر در مشخصات HTTP/1.1 فعلی قرار ندارد. با این حال، استفاده از keep-alive Handshaking هنوز توسط مرورگرها و سرورها رایج است، بنابراین Implementor های HTTP باید برای همکاری با آن آماده باشند. اکنون نگاهی گذرا به عملیات Keep-Alive خواهیم داشت. برای توضیح کامل‌تر درباره Keep-Alive Handshaking به نسخه‌های قدیمی‌تر مشخصات HTTP/1.1 (مانند RFC 2068) مراجعه نمایید.

کلاینت‌هایی که اتصالات HTTP/1.0 keep-alive را پیاده سازی می‌کنند، می‌توانند درخواست کنند که یک اتصال با درج هدر درخواست Connection: Keep-Alive باز نگه داشته شود.

اگر سرور مایل باشد که اتصال را برای درخواست بعدی باز نگه دارد، با همان هدر در پاسخ، پاسخ خواهد داد. اگر هدر Connection: keep-alive در پاسخ وجود نداشته باشد، کلاینت فرض می‌کند که سرور از پشتیبانی نمی‌کند و زمانی که پیام پاسخ ارسال می‌شود، سرور اتصال را می‌بندد.



Keep-Alive Options

توجه داشته باشید که هدرهای Keep-Alive فقط درخواست‌هایی برای زنده نگه داشتن اتصال هستند. در صورت درخواست، کلاینت‌ها و سرورها نیازی به موافقت با یک جلسه نگهدارنده ندارند. آن‌ها می‌توانند در هر زمان اتصالات IDLE را ببندند و تعداد تراکنش‌های پردازش شده در یک اتصال Keep-Alive را محدود کنند.

رفتار Keep-Alive را می‌توان با گزینه‌های جدا شده با کاما که در عنوان کلی Keep-Alive مشخص شده است، تنظیم کرد:





- پارامتر timeout در یک هدر پاسخ Keep-Alive ارسال می‌شود. این مقدار، تخمین می‌زند که سرور برای چه مدت احتمال دارد اتصال را زنده نگه دارد. البته که این موضوع تضمینی نیست.
- پارامتر max در هدر پاسخ Keep-Alive ارسال می‌شود. این مقدار، تخمین می‌زند که سرور احتمالاً برای چند تراکنش HTTP دیگر اتصال را زنده نگه می‌دارد. البته که این موضوع نیز تضمینی نیست.
- هدر Keep-Alive همچنین از ویژگی‌های پردازش نشده دلخواه، عمدتاً برای اهداف تشخیصی و اشکال زدایی پشتیبانی می‌کند. که Syntax آن به صورت [name] [= value] می‌باشد.

وجود هدر Keep-Alive کاملاً اختیاری است، اما فقط زمانی مجاز است که Connection: Keep-Alive نیز وجود داشته باشد. در اینجا نمونه‌ای از هدر پاسخ Keep-Alive وجود دارد که نشان می‌دهد سرور قصد دارد اتصال را حداکثر تا پنج تراکنش دیگر یا تا زمانی که برای دو دقیقه بیکار بماند، باز نگه دارد:

Connection: Keep-Alive

Keep-Alive: max=5, timeout=120

Keep-Alive Connection Restrictions and Rules

در اینجا برخی از محدودیتها و توضیحات در مورد استفاده از اتصالات Keep-Alive وجود دارد:

Connection: به طور پیش فرض در HTTP/1.0 رخ نمی‌دهد. کلاینت باید یک هدر درخواست Keep-Alive برای فعال کردن اتصالات نگه داشتن زنده ارسال کند.

هدر Connection: Keep-Alive باید با تمام پیام‌هایی که می‌خواهند به ماندگاری ادامه دهند، ارسال شود. اگر کلاینت هدر Connection: Keep-Alive را ارسال نکند، سرور پس از آن درخواست، اتصال را می‌بندد.

کلاینت‌ها می‌توانند با تشخیص عدم وجود هدر پاسخ Connection: Keep-Alive، متوجه شوند که آیا سرور پس از پاسخ، اتصال را می‌بندد یا خیر.

تنها در صورتی می‌توان اتصال را باز نگه داشت که طول بدن پیام را بتوان بدون احساس بسته شدن اتصال تعیین کرد - این بدان معنی است که Content-Length Entity Body باید صحیح داشته، دارای نوع رسانه چند بخشی بوده یا با chunked transfer کدگذاری شود. ارسال نامناسب Content-Length به کانال Keep-Alive خوب نیست، زیرا انتهای دیگر تراکنش نمی‌تواند پایان یک پیام و شروع یک پیام دیگر را به دقت تشخیص دهد.





پراکسی‌ها و Gateway‌ها باید قوانین هدر Connection را اجرا کنند. پراکسی یا Connection را قبل از ارسال یا ذخیره پیام حذف کند. هدر نامگذاری شده در هدر Connection و خود هدر Connection را قبل از ارسال یا ذخیره پیام حذف کند.

به طور رسمی، برای جلوگیری از بروز مشکل در پروکسی‌ها که در زیر توضیح داده شده است، باید با سرور پراکسی که تضمینی برای پشتیبانی از هدر Connection ندارد، اتصالات Keep-Alive برقرار شود. البته این موضوع، همیشه در عمل امکان پذیر نیست.

از نظر فنی، هر فیلد هدر Connection (از جمله Connection: Keep-Alive) دریافت شده از دستگاه HTTP/1.0 باید نادیده گرفته شود، زیرا ممکن است به اشتباه توسط یک سرور پراکسی قدیمی‌تر ارسال شده باشد. در عمل، برخی از کلاینت‌ها و سرورها این قانون را منحرف می‌کنند، هر چند که در معرض خطر قرار گرفتن در پروکسی‌های قدیمی‌تر هستند.

کلاینت‌ها باید آماده باشند تا در صورت بسته شدن اتصال قبل از دریافت کل پاسخ، درخواست‌ها را دوباره امتحان کنند، مگر اینکه درخواست در صورت تکرار، Side Effect داشته باشد.

Keep-Alive and Dumb Proxies

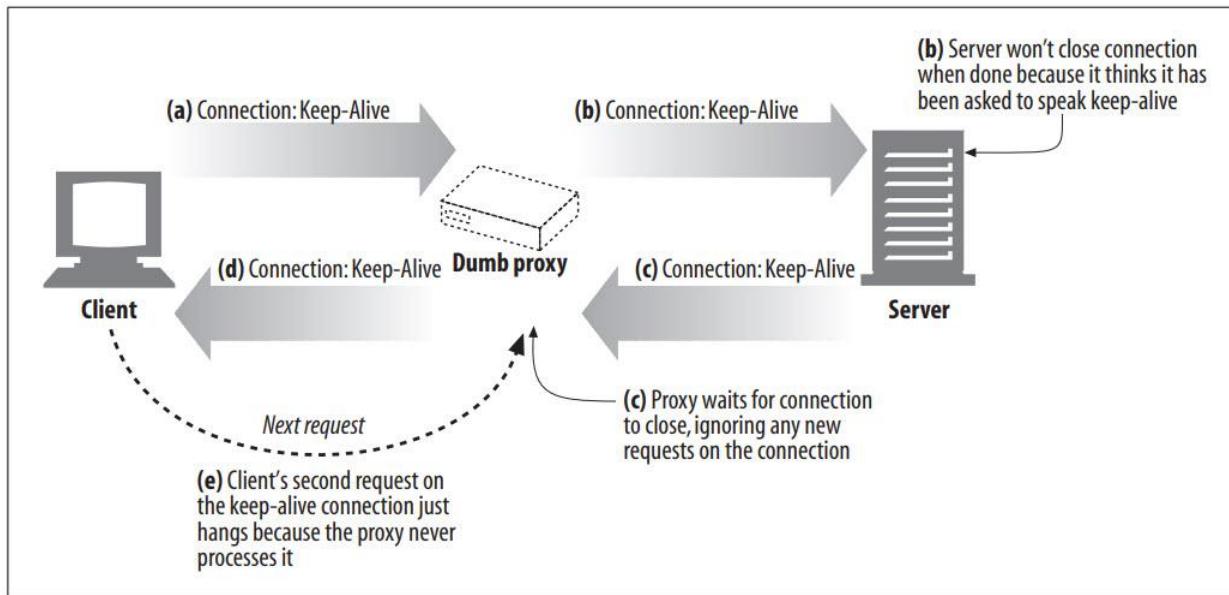
بیایید نگاهی دقیق‌تر به مشکل ظریف پراکسی‌های Dump و Keep-Alive داشته باشیم. یک هدر Connection: Keep-Alive کلاینت وب، تنها بر روی لینک TCP که کلاینت را ترک می‌کند تأثیر می‌گذارد. به همین دلیل است که این هدر، "connection" نامگذاری شده است. اگر کلاینت با یک وب سرور صحبت می‌کند، کلاینت یک هدر Connection: Keep-Alive ارسال می‌کند تا به سرور بگوید که می‌خواهد Keep-Alive بماند. سرور نیز یک هدر Connection: Keep-Alive را در صورت پشتیبانی از Keep-Alive ارسال می‌کند و در صورت عدم پشتیبانی، آن را ارسال نخواهد کرد.

The Connection Header and Blind Relays

مشکل مربوط به پراکسی‌ها است – به‌ویژه، پراکسی‌هایی که هدر Connection را نمی‌فهمند و نمی‌دانند که باید هدر را قبل از پراکسی کردن آن در زنجیره، حذف کنند. بسیاری از پراکسی‌های قدیمی یا ساده به عنوان رله‌های کور (Blind Relays) عمل می‌کنند و بایت‌ها را از یک اتصال به اتصال دیگر تونل نموده، بدون اینکه هدر Connection را پردازش کنند.

تصور کنید یک سرویس گیرنده وب از طریق یک Dump Proxy با یک وب سرور صحبت می‌کند که به عنوان یک رله کور عمل می‌کند. این وضعیت در شکل زیر نشان داده شده است.





مواردی که در این شکل اتفاق می‌افتد عبارتند از:

در بخش a از شکل بالا، یک سرویس گیرنده وب، پیامی را به پروکسی ارسال می‌کند که در آن هدر Connection: Keep-Alive نیز وجود دارد و در صورت امکان درخواست اتصال، Keep-Alive می‌شود. کلاینت منتظر پاسخ می‌ماند تا بداند آیا درخواستش برای کانال Keep-Alive پذیرفته شده است یا خیر.

در قسمت b از شکل بالا، درخواست HTTP را دریافت می‌کند، اما هدر Connection را نمی‌فهمد (فقط آن را به عنوان یک هدر اضافی در نظر می‌گیرد). پروکسی نمی‌داند که Keep-Alive چیست، بنابراین پیام را کلمه به کلمه در زنجیره به سرور ارسال می‌کند (قسمت b از شکل). اما هدر Connection یک هدر hop-by-hop است. این فقط برای یک لینک انتقال اعمال می‌شود و نباید در زنجیره منتقل شود. بدین شکل، اتفاقات بدی در شرف وقوع است.

در قسمت c از شکل بالا، درخواست HTTP رله شده به وب سرور می‌رسد. هنگامی که وب سرور هدر Connection: Keep-Alive را دریافت می‌کند، به اشتباه به این نتیجه می‌رسد که پروکسی (که مانند هر کلاینت دیگری برای سرور به نظر می‌رسد) می‌خواهد به صورت Keep-Alive صحبت کند! این برای وب سرور خوب است - می‌پذیرد که به صورت Keep-Alive صحبت انجام شود و هدر پاسخ Connection: Keep-Alive را مطابق بخش c از شکل، ارسال می‌کند. بنابراین، در این مرحله، وب سرور فکر می‌کند که با پروکسی به صورت Keep-Alive در حال صحبت است و از قوانین Keep-Alive پیروی می‌کند. اما پروکسی اولین موضوع را در مورد Keep-Alive نمی‌داند.





در بخش d از شکل بالا، Dump Proxy پیام پاسخ سرور وب را به کلاینت برمی‌گرداند. در این بخش، هدر Connection: Keep-Alive ای که از سمت وب سرور ارسال شده است نیز به کلاینت ارسال می‌شود. کلاینت این هدر را می‌بیند و فرض می‌کند که پروکسی پذیرفته است که به طور Keep-Alive صحبت کند. بنابراین در این مرحله، هم کلاینت و هم سرور معتقدند که در حال صحبت کردن به صورت Keep-Alive هستند، اما پروکسی که با آن صحبت می‌کنند چیزی در مورد Keep-Alive نمی‌داند.

از آنجایی که پروکسی چیزی در مورد Keep-Alive نمی‌داند، تمام داده‌هایی را که دریافت می‌کند به کلاینت منعکس می‌کند و سپس منتظر می‌ماند تا سرور مبدا اتصال را ببندد. اما سرور مبدا اتصال را نمی‌بندد، زیرا معتقد است که پروکسی صریحاً از سرور خواسته است که اتصال را باز نگه دارد. بنابراین پروکسی منتظر بسته شدن اتصال خواهد ماند.

هنگامی که کلاینت پیام پاسخ را (بخش d از شکل) دریافت می‌کند، مستقیماً به درخواست بعدی حرکت می‌کند و درخواست دیگری را به پروکسی در اتصال Keep-Alive ارسال می‌کند (بخش e از شکل). از آنجایی که پروکسی هرگز انتظار درخواست دیگری در همان اتصال را ندارد، درخواست نادیده گرفته می‌شود و مرورگر فقط می‌چرخد و هیچ پیشرفتی ندارد.

این ارتباط نادرست باعث می‌شود که مرورگر متوقف شود تا زمانی که کلاینت یا سرور اتصال را قطع کند و آن را ببندد.

Proxies and hop-by-hop Headers

برای جلوگیری از این نوع ارتباط نادرست پروکسی، پراکسی‌های مدرن هرگز نباید هدر Connection یا هر هدری که نام آن‌ها در مقادیر Connection ظاهر می‌شود را پراکسی کنند. بنابراین اگر یک پروکسی یک هدر Connection دریافت کند، نباید هدر Connection: Keep-Alive یا هر هدری به نام Connection: Keep-Alive را پروکسی کند.

علاوه بر این، تعدادی هدر hop-by-hop وجود دارد که ممکن است به عنوان مقادیر یک هدر Connection فهرست نشده باشند، اما نباید پروکسی شوند یا به عنوان پاسخ Cache ارائه شوند. این موارد عبارتند از:

- Proxy-Authenticate
- Proxy-Connection
- Transfer-Encoding
- Upgrade





The Proxy-Connection Hack

implementor های مرورگر و پروکسی در Netscape را حلی هوشمندانه برای مشکل رله کور پیشنهاد کردند که نیازی به پشتیبانی همه برنامه‌های وب از نسخه‌های پیشرفته HTTP نداشت. این راه حل یک هدر جدید به نام Proxy-Connection را معرفی کرد و مشکل یک رله کور که مستقیماً پس از کلاینت قرار گرفته بود را حل کرد - اما نه همه موقعیت‌های دیگر. Proxy-Connection زمانی توسط مرورگرهای مدرن پیاده سازی می‌شود که پراکسی‌ها به صراحت پیکربندی شده باشند و توسط بسیاری از پراکسی‌ها قابل درک باشد.

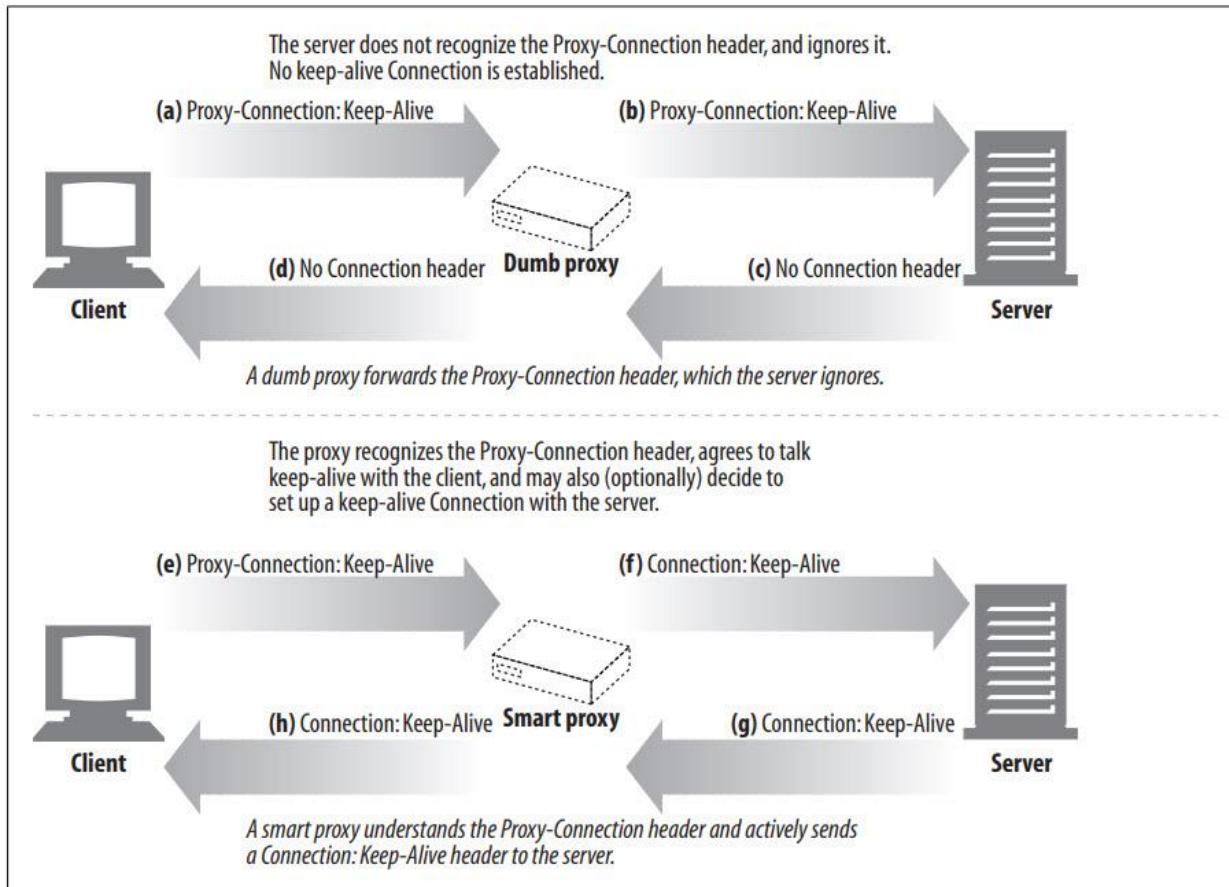
ایده این است که Dump Proxy ها به این دلیل که کورکورانه هدرهای hop-by-hop مانند Connection: Keep-Alive را ارسال می‌کنند، دچار مشکل می‌شوند. هدرهای hop-by-hop فقط مربوط به آن ارتباط منفرد و خاص هستند و نباید فوروارد شوند. هنگامی که هدرهای فوروارد شده توسط سرورهای پایین دستی به عنوان درخواست‌هایی از خود پروکسی برای کنترل اتصال آن اشتباه تفسیر می‌شوند، باعث ایجاد مشکل می‌شود.

در راه حل نت اسکیپ، مرورگرها هدرهای اضافی Proxy-Connection غیر استاندارد را به جای هدرهای Connection که به طور رسمی پشتیبانی می‌شوند و شناخته شده هستند، به پراکسی‌ها ارسال می‌کنند. اگر پروکسی یک رله کور باشد، هدر Proxy-Connection را به سرور وب منتقل می‌کند، که به طور بی ضرر هدر را نادیده می‌گیرد. اما اگر پروکسی یک پروکسی هوشمند باشد (قابلیت درک Persistent Connection) Handshaking Proxy-Connection را با یک هدر Connection جایگزین می‌کند، که سپس به سرور ارسال شده و اثر دلخواه را دارد.

بخش a تا d شکل زیر نشان می‌دهد که چگونه یک رله کور به طور بی خطر هدرهای اتصال پروکسی را به سرور وب منتقل می‌کند، که هدر را نادیده می‌گیرد و باعث می‌شود که هیچ ارتباطی بین کلاینت و پروکسی یا پروکسی و سرور برقرار نشود.

پروکسی هوشمند (بخش e تا h) هدر Proxy-Connection را به عنوان درخواستی برای صحبت به صورت Keep-Alive می‌داند و هدرهای Connection: Keep-Alive خود را برای برقراری اتصالات ارسال می‌کند.

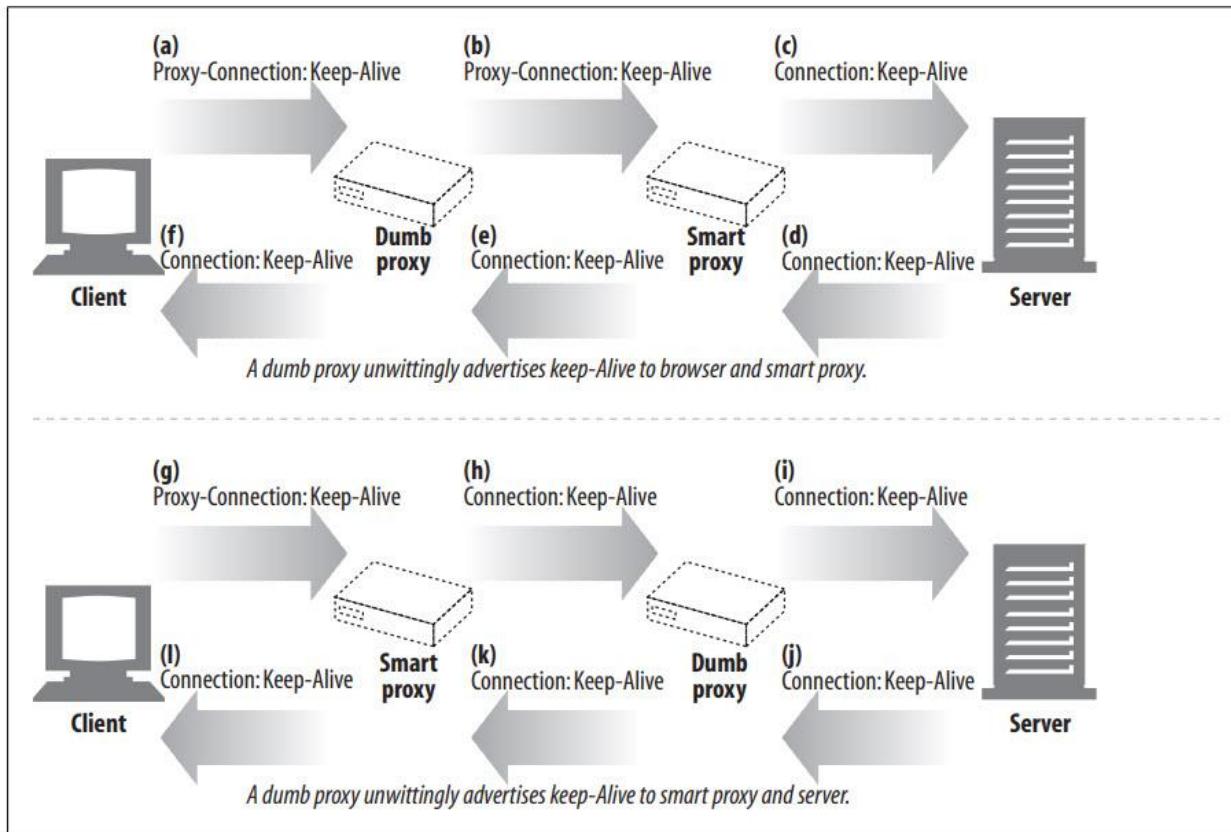




این طرح در شرایطی کار می‌کند که فقط یک پروکسی بین مشتری و سرور وجود دارد. اما اگر یک پروکسی هوشمند در هر دو طرف Dump Proxy وجود داشته باشد، همانطور که در شکل زیر نشان داده شده است، دوباره با مشکل مواجه خواهیم شد.

علاوه بر این، ظاهر شدن invisible Proxy در شبکه‌ها، چه به عنوان فایروال، Intercepting Cache یا Reverse Proxy Server Accelerator، بسیار رایج شده است. از آنجایی که این دستگاه‌ها برای مرورگر نامرئی هستند، مرورگر هدرهای Proxy-Connection را برای آن‌ها ارسال نمی‌کند. بسیار مهم است که برنامه‌های کاربردی وب Transparent اتصالات پایدار را به درستی پیاده سازی کنند.





HTTP/1.1 Persistent Connections

HTTP/1.1 پشتیبانی از اتصالات **Keep-Alive** را متوقف کرد و آن‌ها را با طراحی بهبود یافته‌ای به نام **Persistent Connection** جایگزین کرد. اهداف **Persistent Connection** بوده اما مکانیسم‌ها در این روش، بهتر عمل می‌کنند.

برخلاف اتصالات **HTTP/1.0+ Keep-Alive**، اتصالات **HTTP/1.1** دائمی به طور پیش فرض فعال هستند. HTTP/1.1 فرض می‌کند که همه اتصالات پایدار هستند مگر اینکه خلاف آن مشخص شده باشد. برنامه‌های HTTP/1.1 باید صریحاً یک هدر **Connection: close** را به پیام اضافه کنند تا نشان دهد که یک اتصال باید پس از تکمیل تراکنش بسته شود. این تفاوت قابل توجه با نسخه‌های قبلی پروتکل HTTP است، که در آن اتصالات **Keep-Alive** یا اختیاری بودند یا کاملاً پشتیبانی نمی‌شدند.

یک سرویس گیرنده HTTP/1.1 فرض می‌کند که یک اتصال HTTP/1.1 پس از پاسخ باز می‌ماند، مگر اینکه پاسخ حاوی یک هدر **Connection: Close** باشد. با این حال، کلاینت‌ها و سرورها همچنان می‌توانند اتصالات بی‌کار را در هر زمان بینندند. عدم ارسال **Connection: Close** به این معنی نیست که سرور قول می‌دهد اتصال را برای همیشه باز نگه دارد.





Persistent Connection Restrictions and Rules

در ادامه به محدودیت‌ها و توضیحات در مورد استفاده از اتصالات دائمی یا Persistent Connections اشاره می‌کنیم:

پس از ارسال هدر Connection: Close، کلاینت نمی‌تواند درخواست‌های بیشتری در آن اتصال ارسال کند.

اگر کلاینت نمی‌خواهد درخواست دیگری برای اتصال ارسال کند، باید مقدار Connection: close را در هدر درخواست نهایی خود ارسال کند.

فقط در صورتی می‌توان اتصال را پایدار نگه داشت که همه پیام‌های موجود در اتصال دارای طول پیام درست و مشخص شده‌ای باشند - به عنوان مثال، Content-Length Body Entiy ها باید صحیح داشته باشند یا با Chunked Transfer Encoding، کدگذاری شوند.

پراکسی‌های HTTP/1.1 باید اتصالات دائمی را به طور جداگانه با کلاینت‌ها و سرورها مدیریت کنند—هر اتصال دائمی برای یک Single Transport Hop اعمال می‌شود.

سرورهای پراکسی HTTP/1.1 باید با یک کلاینت HTTP/1.0 ارتباط دائمی برقرار کنند (به دلیل مشکلات پراکسی‌های قدیمی‌تر در ارسال هدرهای Connection) مگر اینکه چیزی در مورد قابلیت‌های کلاینت بدانند. این در عمل دشوار است و بسیاری از Vendor ها این قانون را اعمال نمی‌کنند.

صرف نظر از مقادیر هدرهای Connection، دستگاه‌های HTTP/1.1 ممکن است در هر زمانی اتصال را ببندند، البته سرورها باید سعی کنند تا در میانه انتقال پیام بسته نشوند و همیشه باید حداقل به یک درخواست قبل از بسته شدن پاسخ دهند.

برنامه‌های HTTP/1.1 باید بتوانند پس از بسته شدن ناهمzman (Asynchronous Closes) بازیابی شوند. مشتریان باید درخواست‌ها را تا زمانی که عوارض جانبی انباسته نداشته باشند، دوباره امتحان کنند.

کلاینت‌ها باید آماده باشند تا در صورت بسته شدن اتصال قبل از دریافت کل پاسخ، درخواست‌ها را دوباره امتحان کنند، مگر اینکه درخواست در صورت تکرار، Side Effect داشته باشد.

یک کلاینت تک کاربر باید حداکثر دو اتصال دائمی به هر سرور یا پروکسی داشته باشد تا از Overload سرور جلوگیری کند. از آنجایی که ممکن است پراکسی‌ها برای پشتیبانی از کاربران همزمان به اتصالات بیشتری به سرور نیاز داشته باشند، اگر N کاربر در تلاش برای دسترسی به سرورها باشد، یک پروکسی باید حداکثر 2N اتصال به هر سرور یا پراکسی والد داشته باشد.

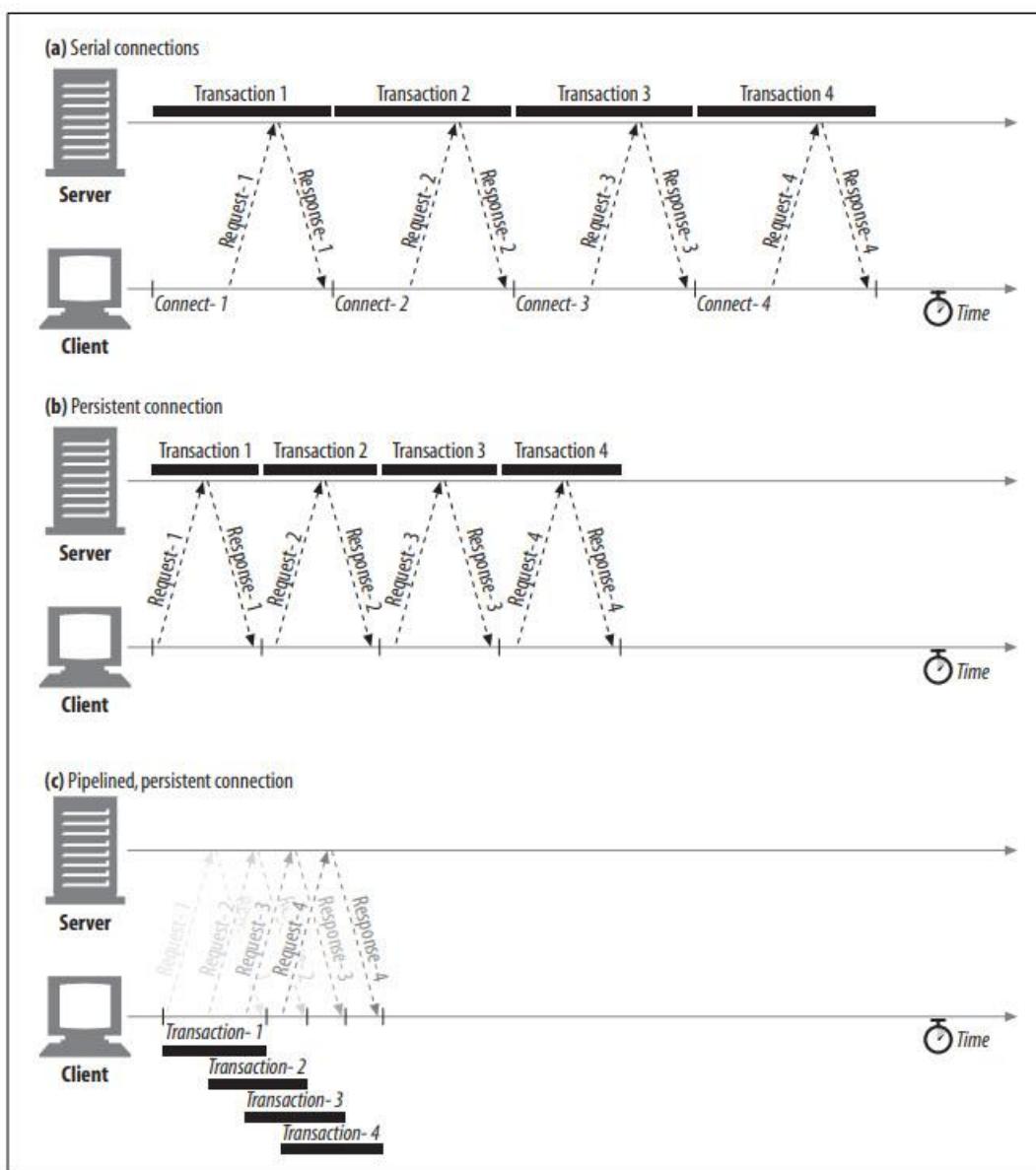




Pipelined Connections

امکان Request Pipelining اختیاری را از طریق اتصالات دائمی فراهم می‌کند. این یک بهینه سازی عملکرد بیشتر نسبت به اتصالات Keep-Alive است. چندین درخواست را می‌توان قبل از رسیدن پاسخ‌ها در صف قرار داد. در حالی که اولین درخواست در سراسر شبکه به سروری در آن سوی کره زمین در حال پخش است، درخواست دوم و سوم می‌تواند انجام شود. این موضوع می‌تواند در شرایطی که شبکه با تأخیر بالایی همراه است، توسط کاهش رفت و برگشت شبکه، منجر به بهبود ارتباط شود.

بخش a تا c شکل زیر نشان می‌دهد که چگونه اتصالات پایدار می‌توانند تاخیرات اتصال TCP را حذف کنند و چگونه Pipelined Request ها (بخش c از شکل) می‌توانند تاخیرهای انتقال را حذف کنند.





چندین محدودیت برای Pipelining وجود دارد:

کلاینت‌های HTTP نباید تا زمانی که از پایداری اتصال مطمئن نشده‌اند، Pipeline شوند.

پاسخ‌های HTTP باید به همان ترتیب درخواست‌ها برگردانده شوند. پیام‌های HTTP با شماره‌های ترتیبی یا Sequence Number برچسب‌گذاری نمی‌شوند، بنابراین اگر پاسخ‌ها نامرتب دریافت شوند، راهی برای تطبیق پاسخ‌ها با درخواست‌ها وجود ندارد.

کلاینت‌های HTTP باید برای بسته شدن اتصال در هر زمان آماده باشند و آماده باشند تا هر درخواست Pipeline ای که تمام نشده است را دوباره انجام دهند. اگر کلاینت یک اتصال دائمی را باز کند و فوراً ۱۰ درخواست صادر کند، سرور آزاد است که فقط پس از پردازش مثلاً ۵ درخواست، اتصال را بیندد.

۵ درخواست باقیمانده با شکست مواجه خواهند شد و کلاینت باید مایل باشد که این بسته‌های زودرس را رسیدگی کرده و درخواست‌ها را مجدداً ارسال کند.

کلاینت‌های HTTP نباید درخواست‌هایی را که دارای Side Effect هستند (مانند POST) Pipeline کنند. به طور کلی، در صورت خطا، Pipelining مانع از این می‌شود که مشتریان بدانند کدام یک از مجموعه درخواست‌های Pipeline شده توسط سرور اجرا شده است. از آنجایی که درخواست‌هایی مانند POST را نمی‌توان با خیال راحت دوباره امتحان کرد، شما این خطر را دارید که برخی از متدها هرگز در شرایط خطا اجرا نشوند.

The Mysteries of Connection Close

مدیریت Connection - به ویژه دانستن زمان و نحوه بستن آن - یکی از هنرهای سیاه عملی در HTTP است. این موضوع ظریفتر از چیزی است که بسیاری از توسعه‌دهنده‌گان در ابتدا متوجه آن می‌شوند و همچنین مطالب کمی در مورد این موضوع نوشته شده است.

“At Will” Disconnection

هر سرویس گیرنده، سرور یا پروکسی HTTP می‌تواند اتصال انتقال TCP را در هر زمانی بیندد. اتصالات معمولاً در انتهای پیام بسته می‌شوند، اما در شرایط خطا، اتصال ممکن است در میان یک خط هدر یا در مکان‌های عجیب دیگری بسته شود.

این وضعیت در اتصالات پایدار Pipeline شده رایج است. برنامه‌های HTTP می‌توانند پس از هر دوره زمانی، اتصالات دائمی را بینندن. به عنوان مثال، پس از اینکه یک اتصال دائمی برای مدتی بیکار بود، سرور ممکن است تصمیم بگیرد که آن را خاموش کند.





با این حال، سرور هرگز نمی‌تواند به طور قطع بداند که کلاینت در انتهای خط در همان زمان که اتصال "IDLE" توسط سرور بسته می‌شود، قصد ارسال داده را نداشته است. اگر این اتفاق بیفتد، کلاینت یک خطای اتصال را در میانه نوشتن پیام درخواست خود، می‌بیند.

Content-Length and Truncation

هر پاسخ HTTP باید یک هدر Content-Length دقیق برای توصیف اندازه بدن پاسخ داشته باشد. برخی از سرورهای HTTP قدیمی‌تر، هدر Content-Length را حذف می‌کنند یا طولی اشتباه دارند.

هنگامی که یک کلاینت یا پروکسی پاسخ HTTP را دریافت می‌کند که در Connection Close خاتمه می‌یابد، و طول موجودیت واقعی منتقل شده با Content-Length مطابقت ندارد (یا هیچ Content-Length وجود ندارد)، گیرنده باید صحت طول را زیر سوال ببرد.

اگر گیرنده یک Caching Proxy می‌باشد، گیرنده نباید پاسخ را در Cache نگه دارد (برای به حداقل رساندن ترکیب بعدی خطای احتمالی). پروکسی باید پیام مشکوک را به صورت دست نخورده و بدون تلاش برای «اصلاح» Content-Length، جهت حفظ شفافیت معنایی، ارسال کند.

Connection Close Tolerance, Retries, and Idempotency

اتصالات می‌توانند در هر زمان بسته شوند، حتی در شرایط بدون خطا. برنامه‌های HTTP باید آماده باشند تا بسته‌شدن‌های غیرمنتظره را به درستی مدیریت کنند.

اگر یک Transport Connection در حالی بسته شود که کلاینت در حال انجام تراکنش است، مشتری باید اتصال را دوباره باز کند و یک بار دوباره امتحان کند، مگر اینکه تراکنش Side Effect داشته باشد. کلاینت می‌تواند تعداد زیادی درخواست را در نوبت قرار دهد، اما سرور ممکن است اتصال را بینند و درخواست‌های زیادی را پردازش نشده و نیاز به زمان‌بندی مجدد بگذارد.

Side Effect ها مهم هستند. هنگامی که یک اتصال پس از ارسال برخی از داده‌های درخواستی بسته می‌شود، قبل از بازگشت پاسخ، مشتری نمی‌تواند ۱۰۰٪ مطمئن باشد که واقعاً چه مقدار از تراکنش توسط سرور فراخوانی شده است. برخی از تراکنش‌ها، مانند دریافت یک صفحه HTML استاتیک، می‌توانند بارها و بارها بدون تغییر چیزی، تکرار شوند. سایر تراکنش‌ها، مانند ارسال سفارش به فروشگاه آنلاین کتاب، نباید تکرار شوند، یا حتی ممکن است چندین سفارش را به خطر بیندازند.

تراکنش، در صورتی فاقد قدرت (Idempotent) است که صرف نظر از اینکه یک بار یا چند بار انجام شده باشد، همان نتیجه را به همراه داشته باشد. Implementor ها می‌توانند فرض کنند که متدهای

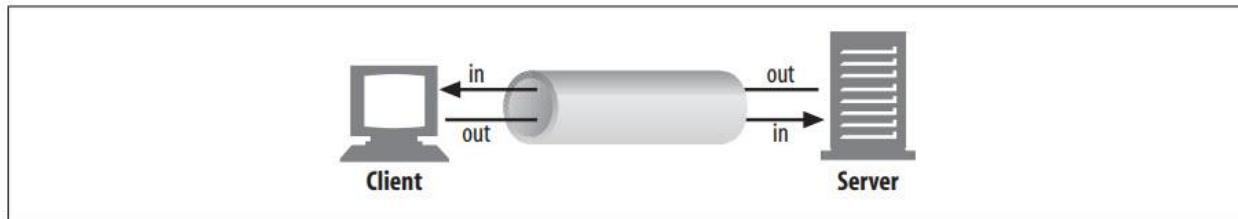


OPTIONS و TRACE، DELETE، PUT، HEAD، GET درخواست‌های POST (مانند Pipeline) را گذارند. کلاینت‌ها نباید درخواست‌های Idempotent می‌تواند منجر به نتایج نامشخص شود. اگر می‌خواهید یک درخواست Transport Connection ارسال کنید، باید منتظر وضعیت پاسخ برای درخواست قبلی باشید Nonidempotent.

متدها یا توالی‌های Nonidempotent نباید به‌طور خودکار دوباره امتحان شوند، اگرچه User Agent‌ها ممکن است انتخاب مجدد درخواست را به اپراتور انسانی پیشنهاد دهند. برای مثال، اکثر مرورگرها هنگام بارگذاری مجدد پاسخ POST ذخیره شده، یک کادر محاوره ای ارائه می‌دهند که از شما می‌پرسد که آیا می‌خواهید تراکنش را دوباره پست کنید.

Graceful Connection Close

همانطور که در شکل زیر نشان داده شده است، اتصالات TCP دو طرفه هستند. هر طرف اتصال TCP دارای یک صف ورودی و یک صف خروجی برای خواندن یا نوشتن اطلاعات است. داده‌های قرار داده شده در خروجی یک طرف در نهایت در ورودی طرف دیگر نشان داده می‌شود.

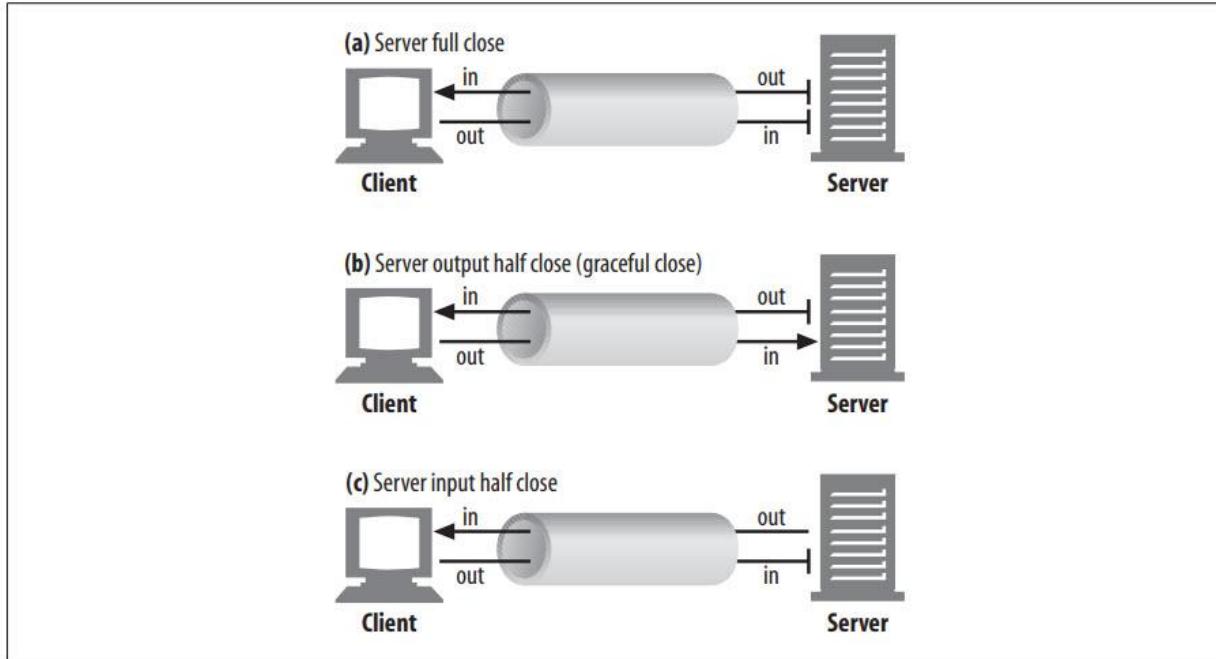


Full and half closes

یک برنامه کاربردی می‌تواند یکی یا هر دو کanal ورودی و خروجی TCP را ببندد. فراخوانی سوکت () Close، هر دو کanal ورودی و خروجی یک اتصال TCP را می‌بندد.

این قابلیت، بستن کامل یا Full Close نامیده می‌شود که در بخش a شکل زیر نشان داده شده است. می‌توانید از فراخوانی سوکت () shutdown برای بستن کanal ورودی یا خروجی به صورت جداگانه استفاده کنید. این قابلیت، Half Close نامیده می‌شود که در بخش b شکل زیر قابل مشاهده است.





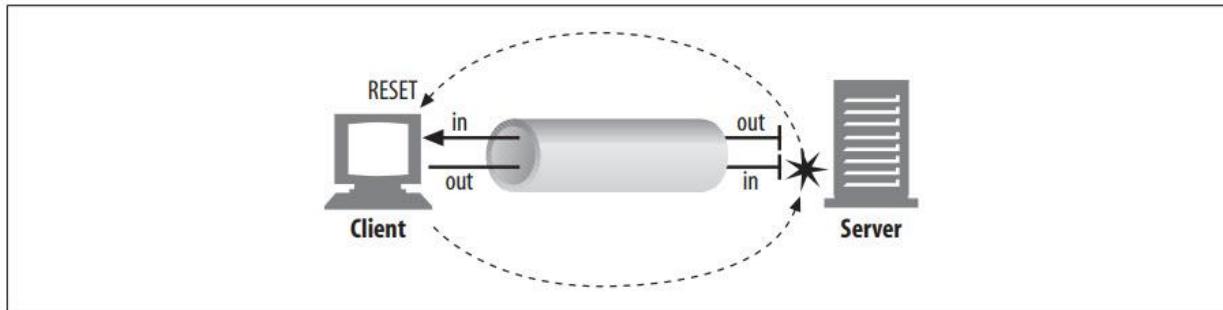
TCP close and reset errors

برنامه‌های ساده HTTP فقط می‌توانند از Full Close استفاده کنند. اما وقتی برنامه‌ها شروع به صحبت با بسیاری از انواع دیگر از کلاینت‌ها، سرورها و پراکسی‌های HTTP می‌کنند و زمانی که شروع به استفاده از اتصالات دائمی شده می‌کنند، استفاده از Half Close برای جلوگیری از دریافت خطاهای نوشتن غیرمنتظره توسط همتایان اهمیت پیدا می‌کند.

به طور کلی، بستن کanal خروجی اتصال شما همیشه ایمن است. در این صورت هنگامی که تمام داده‌ها از بافر آن خوانده شد، به طرف دیگر اتصال به وسیله یک اعلان end-of-stream اطلاع داده می‌شود که اتصال را بسته‌اید.

بستن کanal ورودی اتصال شما خطرناک تر است، مگر اینکه بدانید طرف مقابل قصد ارسال اطلاعات بیشتری را ندارد. اگر طرف دیگر داده‌ها را به کanal ورودی بسته شما ارسال کند، همانطور که در شکل زیر نشان داده شده است، سیستم عامل یک پیام "connection reset by peer" TCP را به دستگاه طرف دیگر ارسال می‌کند. اکثر سیستم‌عامل‌ها این را به عنوان یک خطای جدی در نظر می‌گیرند و هر داده بافری را که طرف مقابل هنوز نخوانده است پاک می‌کند. این برای اتصالات Pipeline شده بسیار بد است.





فرض کنید ۱۰ درخواست Pipeline شده را روی یک اتصال دائمی ارسال کرده اید. در این حالت پاسخ‌ها قبل از رسیده اند و در بافر سیستم عامل شما قرار دارند (اما برنامه هنوز آن‌ها را نخوانده است). اکنون شما می‌گویید که درخواست شماره ۱۱ را ارسال می‌کنید، اما سرور تصمیم می‌گیرد که شما به اندازه کافی از این اتصال استفاده کرده اید و آن را می‌بندد. درخواست شماره ۱۱ شما به یک Closed Connection می‌رسد و یک Reset را به شما منعکس می‌کند. این Reset بافرهای ورودی شما را پاک می‌کند.

هنگامی که در نهایت به خواندن داده‌ها می‌رسید، با خطای Connection Reset by Peer روبرو می‌شوید و داده‌های پاسخ خوانده نشده باfer از بین می‌روند، حتی اگر بیشتر آن‌ها با موفقیت به دستگاه شما رسیده باشند.

Graceful close

در پروتکل HTTP توصیه می‌شود که وقتی کلاینت‌ها یا سرورها می‌خواهند اتصالی را به‌طور غیرمنتظره ببندند، باید «یک Transport Connection برای Graceful Close صادر کنند»، اما نحوه انجام این کار را توضیح نمی‌دهد.

به طور کلی، برنامه‌هایی که Graceful Close را اجرا می‌کنند، ابتدا کانال‌های خروجی خود را می‌بندند و سپس منتظر می‌مانند تا همتا در طرف دیگر اتصال، کانال‌های خروجی خود را ببندند. هنگامی که هر دو طرف به یکدیگر می‌گویند که دیگر داده‌ای ارسال نمی‌کنند (به عنوان مثال، کانال‌های خروجی را می‌بندند)، اتصال می‌تواند بدون خطر Reset، به طور کامل بسته شود.

متأسفانه، هیچ تضمینی وجود ندارد که طرف دیگر ارتباط، Half Close را اجرا یا بررسی نماید. به همین دلیل، برنامه‌هایی که می‌خواهند به خوبی بسته (Graceful Close) شوند، باید کانال‌های خروجی خود را Half Close نمایند و به صورت دوره‌ای وضعیت کانال‌های ورودی خود را بررسی کنند (به‌دلیل داده یا انتهای جریان باشند). اگر کانال ورودی توسط طرف دیگر ارتباط در مدتی بسته نشود، برنامه ممکن است برای صرفه جویی در منابع، اتصال را مجبور به بسته شدن نماید.





For More Information

<http://www.ietf.org/rfc/rfc2616.txt>

<http://www.ietf.org/rfc/rfc2068.txt>

<http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-connection-00.txt>

<http://www.w3.org/Protocols/HTTP/Performance/>

<http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html>

http://www.isi.edu/lsam/publications/phttp_tcp_interactions/paper.html

<ftp://gatekeeper.dec.com/pub/DEC/WRL/research-reports/WRL-TR-95.4.pdf>

<http://www.sun.com/sun-on-net/performance/tcp.slowstart.html>

<http://www.acm.org/sigcomm/CCR/archive/2001/jan01/CCR-200101-mogul.pdf>

<http://www.ietf.org/rfc/rfc2001.txt>

<http://www.ietf.org/rfc/rfc1122.txt>

<http://www.ietf.org/rfc/rfc896.txt>

<http://www.ietf.org/rfc/rfc0813.txt>

<http://www.ietf.org/rfc/rfc0793.txt>





فصل پنجم – Web Servers

وب سرورها روزانه میلیاردها صفحه وب را پخش می‌کنند. آن‌ها وضعیت آب و هوا را به شما می‌گویند، به وسیله آن‌ها می‌توانید خریدهای خود را به صورت آنلاین انجام دهید و به شما اجازه می‌دهند دوستان دبیرستانی را که مدت‌هاست گم کرده‌اید پیدا کنید. وب سرورها، اسباب کار شبکه جهانی وب هستند. در این فصل ما موارد زیر را پوشش خواهیم داد:

- بررسی انواع مختلف سرورهای وب نرم افزاری و سخت افزاری
- تشریح نحوه نوشتن یک برنامه ساده برای تشخیص وب سرور در پرل
- توضیح گام به گام نحوه پردازش تراکنش‌های HTTP توسط سرورهای وب

Web Servers Come in All Shapes and Sizes

یک وب سرور درخواست‌های HTTP را پردازش می‌کند و پاسخ‌های متناسب را ارائه می‌دهد. اصطلاح «وب سرور» می‌تواند به نرم‌افزار وب سرور یا دستگاه یا رایانه خاصی که برای سرویس دادن به صفحات وب اختصاص داده شده است اشاره کند.

وب سرورها در انواع، شکل‌ها و اندازه‌های مختلفی عرضه می‌شوند. سرورهای وب ساده ۱۰ خطی اسکریپت Perl، موتورهای تجاری ایمن ۵۰ مگابایتی و سرورهای کوچک روی کارت (servers-on-a-card) وجود دارند. اما تفاوت‌های عملکردی هرچه که باشد، تمامی سرورهای وب، درخواست‌های HTTP را دریافت نموده و محتوا را به کلاینت‌ها ارائه می‌دهند.

Web Server Implementations

سرورهای وب، HTTP و مدیریت ارتباط TCP مربوطه را پیاده سازی می‌کنند. آن‌ها همچنین منابع ارائه شده توسط وب سرور را مدیریت می‌کنند و ویژگی‌های مدیریتی را برای پیکربندی، کنترل و بهبود وب سرور ارائه می‌دهند.

منطق وب سرور، پروتکل HTTP را پیاده سازی می‌کند، منابع وب را مدیریت نموده و قابلیت‌های مدیریت وب سرور را فراهم می‌کند. منطق وب سرور، مسئولیت به اشتراکت گذاشتن مدیریت اتصالات TCP با سیستم عامل را به عهده دارد.

سیستم عامل موجود در وب سرور نیز جزئیات سخت افزاری سیستم کامپیوتری زیربنایی را مدیریت می‌کند و پشتیبانی از شبکه TCP/IP، سیستم‌های فایل برای نگهداری منابع وب و مدیریت فرآیند برای کنترل فعالیت‌های محاسباتی جاری را فراهم می‌نماید.





وب سرورها به اشکال مختلف در دسترس هستند:

- شما می‌توانید وب سرورهای نرم افزاری همه منظوره را بر روی سیستم‌های کامپیوتراست ایجاد نصب و اجرا کنید.
- اگر زحمت نصب نرم افزار را نمی‌خواهید، می‌توانید یک ابزار وب سرور بخرید، که در آن نرم افزار از پیش نصب شده و از پیش پیکربندی شده روی یک کامپیوتر، اغلب در یک شاسی زیبا قرار دارد.
- با توجه به معجزات ریزپردازنده‌ها، برخی از شرکت‌ها حتی وب سرورهای تعییه شده (Embedded) را ارائه می‌دهند که در تعداد کمی از تراشه‌های کامپیوترا پیاده سازی شده‌اند و آن‌ها را به کنسول‌های مدیریتی عالی برای دستگاه‌های مصرف کننده تبدیل می‌کند.

General-Purpose Software Web Servers

وب سرورهای نرم افزاری همه منظوره بر روی سیستم‌های کامپیوتراست ایجاد و دارای شبکه فعال می‌شوند. می‌توانید نرم افزار منبع باز (مانند Apache یا W3C's Jigsaw) یا نرم افزار تجاری (مانند وب سرورهای مايكروسافت و iPlanet) را انتخاب کنید. نرم افزار وب سرور تقریباً برای هر رایانه و سیستم عامل موجود است.

در حالی که ده‌ها هزار نوع مختلف از برنامه‌های وب سرور وجود دارد (از جمله وب سرورهای سفارشی و با هدف خاص)، اکثر نرم افزارهای وب سرور از تعداد کمی از سازمان‌ها تهیه می‌شوند.

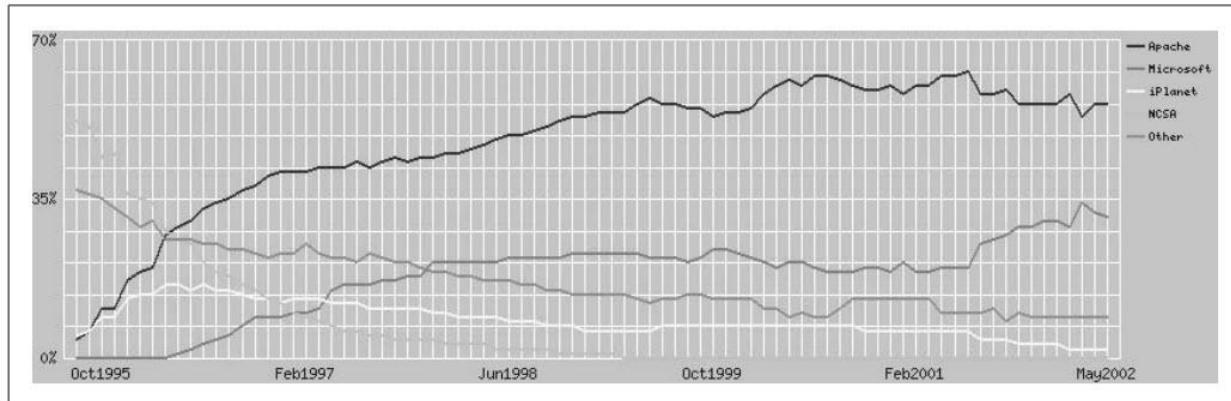
در فوریه ۲۰۰۲، نظرسنجی نت کرافت (<http://www.netcraft.com/survey/>) نشان داد که سه فروشنده بر بازار وب سرورهای اینترنتی عمومی تسلط دارند:

نرم افزار رایگان آپاچی تقریباً ۶۰ درصد از تمام وب سرورهای اینترنتی را تأمین می‌کند.

۳۰ درصد مابقی در اختیار وب سرورهای مايكروسافت است.

۳ درصد از این بازار هم در اختیار Sun iPlanet است.





با توجه به اینکه نت کرافت معمولاً در مورد سلط نرم افزار آپاچی اغراق آمیز صحبت می‌کند، آمار مربوط به آپاچی را کمی اغراق در نظر بگیرید.

مطالعات بر روی سرورهای پروکسی از ISP های بزرگ نشان می‌دهد که تعداد صفحات ارائه شده از سرورهای Apache بسیار کمتر از ۶۰ درصد است اما همچنان از مایکروسافت و Sun iPlanet فراتر می‌رود. علاوه بر این، سرورهای مایکروسافت و iPlanet در داخل شرکت‌های بزرگ از Apache محبوب‌تر هستند.

Web Server Appliances

راه حل‌های نرم افزاری/سخت افزاری از پیش بسته بندی شده هستند. فروشنده یک سرور نرم افزاری را روی یک پلت فرم کامپیوترا انتخاب شده توسط فروشنده نصب می‌کند و نرم افزار را پیکربندی می‌کند. چند نمونه از Web Server Appliances عبارتند از:

- Sun/Cobalt RaQ web appliances (<http://www.cobalt.com>)
- Toshiba Magnia SG10 (<http://www.toshiba.com>)
- IBM Whistle web server appliance (<http://www.whistle.com>)

راه حل‌های Appliance نیاز به نصب و پیکربندی نرم‌افزار را برطرف نموده و اغلب مدیریت را تا حد زیادی ساده می‌کنند. با این حال، وب سرور اغلب انعطاف‌پذیری و ویژگی‌های غنی‌تری دارد و سخت‌افزار سرور به راحتی قابل استفاده مجدد یا ارتقاء نیست.

Embedded Web Servers

سرورهای Embedded، وب سرورهای کوچکی هستند که برای تعییه در محصولات مصرفی (به عنوان مثال، چاپگرهای لوازم خانگی) در نظر گرفته شده‌اند. وب سرورهای Embedded به کاربران این





امکان را می‌دهد که دستگاه‌های مصرف کننده خود را با استفاده از یک رابط مرورگر وب مناسب مدیریت کنند.

برخی از وب سرورهای Embedded را می‌توان حتی در کمتر از یک اینچ مربع پیاده سازی کرد، اما آن‌ها معمولاً یک مجموعه ویژگی حداقلی را ارائه می‌دهند. دو نمونه از وب سرورهای Embedded بسیار کوچک عبارتند از:

- IPic match-head sized web server (<http://www-ccs.cs.umass.edu/~shri/iPic.html>)
- NetMedia SitePlayer SP1 Ethernet Web Server (<http://www.siteplayer.com>)

A Minimal Perl Web Server

اگر می‌خواهید یک سرور HTTP با امکانات کامل بسازید، باید کارهای زیادی انجام دهید. هسته وب سرور آپاچی بیش از ۵۰۰۰۰ خط کد دارد و مازولهای پردازش اختیاری، این تعداد را بسیار بزرگتر می‌کنند.

همه این نرم افزار برای پشتیبانی از ویژگی‌های HTTP/1.1 مورد نیاز است. ویژگی‌هایی از قبیل: پشتیبانی از منابع غنی، میزبانی مجازی(Virtual Hosting)، کنترل دسترسی، ورود به سیستم، پیکربندی، نظارت و ویژگی‌های عملکرد.

همچنین شما می‌توانید یک سرور HTTP با حداقل عملکرد را در کمتر از ۳۰ خط پرل ایجاد کنید. بیانگاهی به آن بیندازیم.

تصویر زیر، یک برنامه Perl کوچک به نام type-o-serve را نشان می‌دهد. این برنامه یک ابزار تشخیصی مفید برای تست تعامل با کلاینت و پروکسی‌ها است. مانند هر وب سرور، type-o-serve منتظر اتصال HTTP است. به محض اینکه type-o-serve پیام درخواست را دریافت می‌کند، پیام را روی صفحه چاپ می‌کند. سپس منتظر می‌ماند تا یک پیام پاسخ را تایپ کنید (یا Paste کنید) که برای کلاینت ارسال می‌شود. به این ترتیب، type-o-serve وانمود می‌کند که یک وب سرور است، پیام‌های درخواست HTTP دقیق را ضبط می‌کند و به شما امکان می‌دهد هر پیام پاسخ HTTP را ارسال کنید.

این ابزار ساده type-o-serve اکثر قابلیت‌های HTTP را پیاده‌سازی نمی‌کند، اما ابزار مفیدی برای تولید پیام‌های پاسخ سرور است. می‌توانید برنامه type-o-serve را از لینک زیر دانلود نمایید.

<http://www.http-guide.com/tools/type-o-serve.pl>





```
#!/usr/bin/perl

use Socket;
use Carp;
use FileHandle;

# (1) use port 8080 by default, unless overridden on command line
$port = (@ARGV ? $ARGV[0] : 8080);

# (2) create local TCP socket and set it to listen for connections
$proto = getprotobynumber('tcp');
socket(S, PF_INET, SOCK_STREAM, $proto) || die;
setsockopt(S, SOL_SOCKET, SO_REUSEADDR, pack("l", 1)) || die;
bind(S, sockaddr_in($port, INADDR_ANY)) || die;
listen(S, SOMAXCONN) || die;

# (3) print a startup message
printf("    <<<Type-0-Serve Accepting on Port %d>>>\n\n",$port);

while (1)
{
    # (4) wait for a connection C
    $cport_caddr = accept(C, S);
    ($cport,$caddr) = sockaddr_in($cport_caddr);
    C->autoflush(1);

    # (5) print who the connection is from
    $cname = gethostbyaddr($caddr,AF_INET);
    printf("    <<<Request From '%s'>>>\n",$cname);

    # (6) read request msg until blank line, and print on screen
    while ($line = <C>)
    {
        print $line;
        if ($line =~ /^$/r/) { last; }
    }

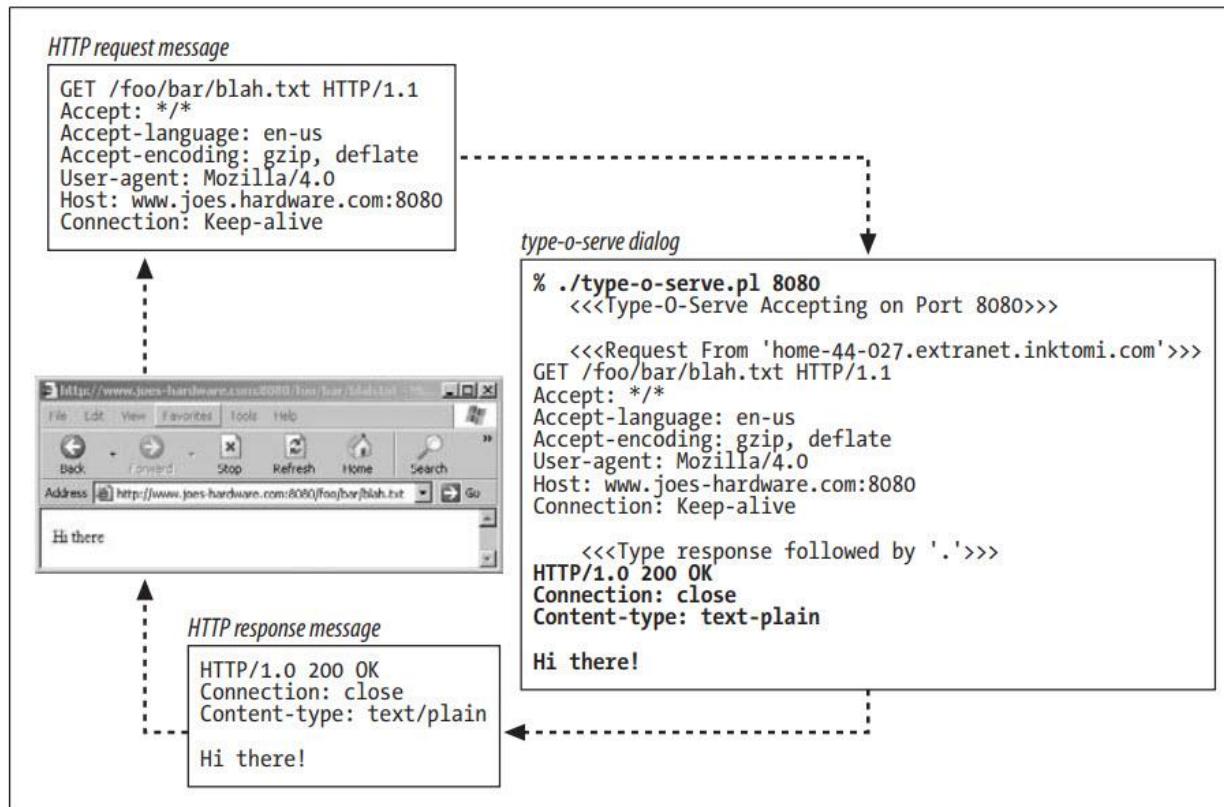
    # (7) prompt for response message, and input response lines,
    #     sending response lines to client, until solitary "."
    printf("    <<<Type Response Followed by '.'>>>\n");

    while ($line = <STDIN>)
    {
        $line =~ s/\r//;
        $line =~ s/\n//;
        if ($line =~ /^./) { last; }
        print C $line . "\r\n";
    }
    close(C);
}
```





شکل زیر نشان می‌دهد که چگونه ممکن است مدیر فروشگاه سخت افزار Joe از type-o-serve برای آزمایش ارتباط HTTP استفاده کند:



ابتدا، مدیر سرور، type-o-serve را Listen نمودن یک پورت خاص راه اندازی می‌کند. از آنجایی که فروشگاه سخت افزار Joe در حال حاضر دارای یک وب سرور در محیط Production بوده که بر روی پورت ۸۰ شده است، مدیر سرور، type-o-serve را در پورت ۸۰۸۰ (شما می‌توانید هر پورت بدون استفاده را انتخاب کنید) با دستور راه اندازی می‌کند:

```
% type-o-serve.pl 8080
```

هنگامی که type-o-serve اجرا می‌شود، می‌توانید یک مرورگر را به این وب سرور هدایت کنید. در شکل بالا، ما به آدرس <http://www.joes-hardware.com:8080/foo/bar/blah.txt> مراجعه می‌کنیم.

برنامه type-o-serve پیام درخواست HTTP را از مرورگر دریافت می‌کند و محتوای پیام درخواست HTTP را روی صفحه چاپ می‌کند. سپس برنامه type-o-serve منتظر می‌ماند تا کاربر یک پیام پاسخ ساده را تایپ کند و سپس یک نقطه در یک خط خالی قرار دهد.

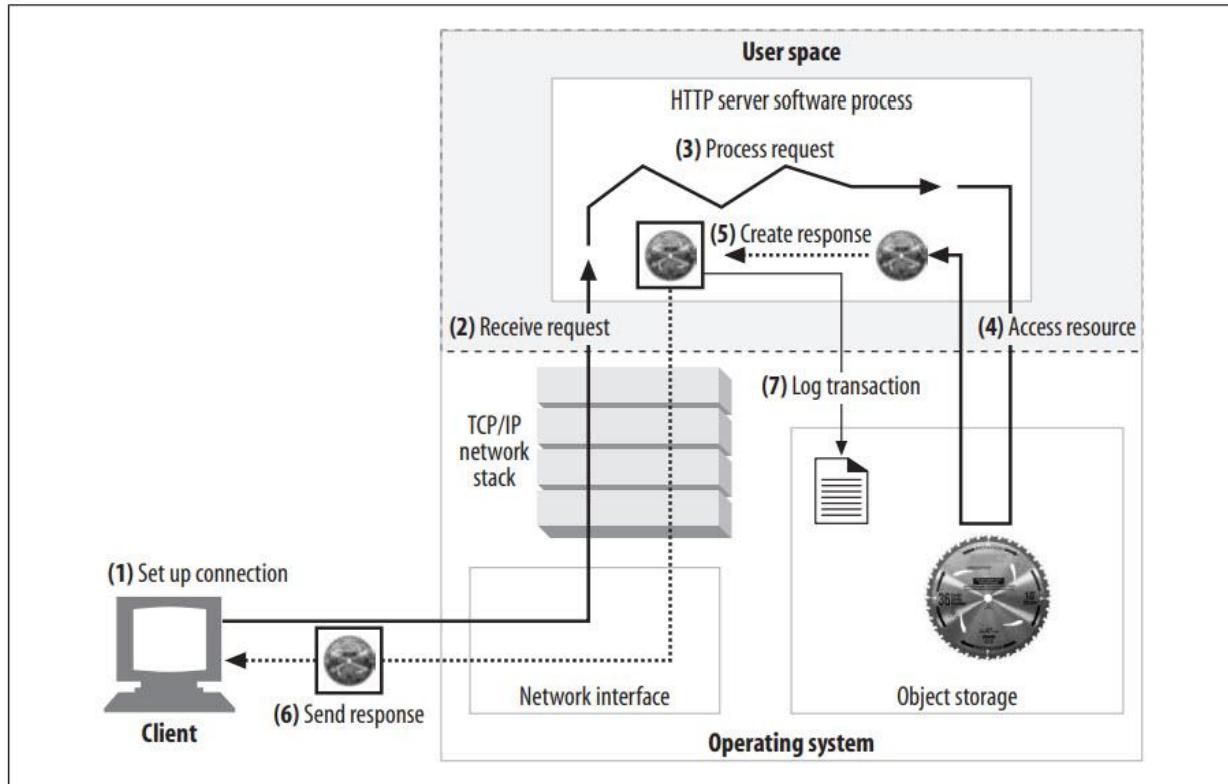
type-o-serve پیام پاسخ HTTP را به مرورگر می‌فرستد و مرورگر متن پیام پاسخ را نمایش می‌دهد.





What Real Web Servers Do

سرور پرل که در بخش قبلی نشان داده شد، یک وب سرور نمونه بی اهمیت است. وب سرورهای تجاری پیشرفته بسیار پیچیده‌تر هستند، اما آن‌ها چندین کار را رایج را انجام می‌دهند:



— یک اتصال کلاینت را می‌پذیرد، یا اگر کلاینت ناخواسته است، آن را می‌بندد. **Set up connection**

— یک پیام درخواست HTTP از شبکه را می‌خواند. **Receive request**

— پیام درخواست را تفسیر نموده و اقدام لازم را بر روی آن انجام می‌دهد. **Process request**

— دسترسی به منبع مشخص شده در پیام — **Access resource**

— پیام پاسخ HTTP را با هدرهای مناسب ایجاد می‌نماید. **Construct response**

— پاسخ را برای مشتری ارسال می‌کند. **Send response**

— لگ‌های مربوط به تراکنش انجام شده را در یک فایل گزارش قرار می‌دهد. **Log transaction**

هفت بخش بعدی نشان می‌دهد که چگونه وب سرورها این وظایف اساسی را انجام می‌دهند.





Step 1: Accepting Client Connections

اگر یک کلاینت قبلاً یک اتصال دائمی به سرور داشته باشد، می‌تواند از آن اتصال برای ارسال درخواست خود استفاده کند. در غیر این صورت، مشتری باید یک اتصال جدید به سرور باز کند (برای بررسی فناوری مدیریت اتصال HTTP به فصل ۴ مراجعه کنید).

Handling New Connections

هنگامی که یک سرویس گیرنده درخواست اتصال TCP به وب سرور را ارسال می‌کند، وب سرور اتصال را برقرار می‌کند و تعیین می‌کند که کدام کلاینت در طرف دیگر اتصال است و آدرس IP را از اتصال TCP استخراج می‌کند. هنگامی که یک اتصال جدید برقرار و پذیرفته شد، سرور اتصال جدید را به لیست اتصالات وب سرور موجود خود اضافه می‌کند و برای مشاهده داده‌های موجود در اتصال آماده می‌شود.

وب سرور در رد کردن و بستن فوری هر گونه اتصال آزاد است. برخی از وب سرورها اتصالات را می‌بندند زیرا آدرس IP کلاینت یا نام میزبان غیرمجاز است یا یک سرویس گیرنده، مخرب شناخته شده است.

Client Hostname Identification

اکثر وب سرورها را می‌توان با استفاده از "Reverse DNS" برای تبدیل آدرس‌های IP کلاینت به نام میزبان کلاینت، پیکربندی کرد. وب سرورها می‌توانند از Hostname کلاینت برای کنترل دسترسی دقیق و ثبت گزارش استفاده کنند. البته هشدار داده می‌شود که جستجوی نام میزبان می‌تواند زمان بسیار زیادی طول بکشد و تراکنش‌های وب را کند کند. بسیاری از وب سرورهای با ظرفیت بالا یا Hostname Resolution را غیرفعال نموده و یا آن را فقط برای محتوای خاصی فعال می‌کنند.

می‌توانید جستجوی Hostname را در آپاچی با دستور پیکربندی HostnameLookups فعال کنید. به عنوان مثال، دستور العمل‌های پیکربندی آپاچی در مثال زیر، Hostname Resolution را فقط برای منابع HTML و CGI فعال می‌کند.

```
HostnameLookups off
```

```
<Files ~ "\.(html|htm|cgi)"$>
```

```
HostnameLookups on
```

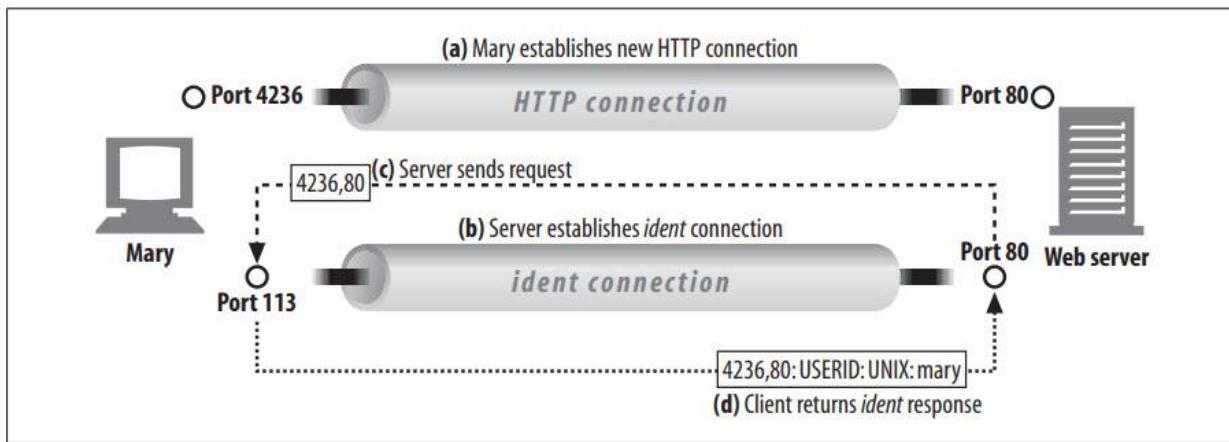
```
</Files>
```



Determining the Client User Through ident

برخی از سرورهای وب نیز از پروتکل IETF ident پشتیبانی می‌کنند. پروتکل ident به سرورها اجازه می‌دهد تا بفهمند که چه نام کاربری اتصال HTTP را آغاز کرده است. این اطلاعات به صورت خاص برای Logging و وب سرور مفید است. فیلد دوم Log Format حاوی نام کاربری ident هر درخواست HTTP است.

اگر یک کلاینت از پروتکل ident پشتیبانی کند، کلاینت به درگاه TCP 113 برای درخواست‌های ident گوش می‌دهد. شکل زیر نحوه عملکرد پروتکل ident را نمایش می‌دهد.



در بخش a از شکل بالا، کلاینت یک اتصال HTTP را باز می‌کند. سپس سرور اتصال خود را به پورت سرور کلاینت (۱۱۳) برقرار می‌کند، یک درخواست ساده می‌فرستد و نام کاربری مربوط به اتصال جدید را می‌پرسد و پاسخ حاوی نام کاربری را از کلاینت دریافت می‌کند.

ident می‌تواند در داخل سازمان‌ها کار کند، اما به دلایل زیادی در سراسر اینترنت عمومی به خوبی کار نمی‌کند، از جمله:

- بسیاری از رایانه‌های شخصی مشتری، نرم افزار دیمون identd Identification Protocol را اجرا نمی‌کنند.
- پروتکل ident به طور قابل توجهی تراکنش‌های HTTP را به تاخیر می‌اندازد.
- بسیاری از فایروال‌ها اجازه ترافیک ident ورودی را نمی‌دهند.
- پروتکل ident نامن است و ساخت آن آسان است.
- پروتکل ident آدرس‌های Virtual IP را به خوبی پشتیبانی نمی‌کند.
- نگرانی‌های مربوط به حریم خصوصی در مورد افشا نامهای کاربری مشتری وجود دارد.

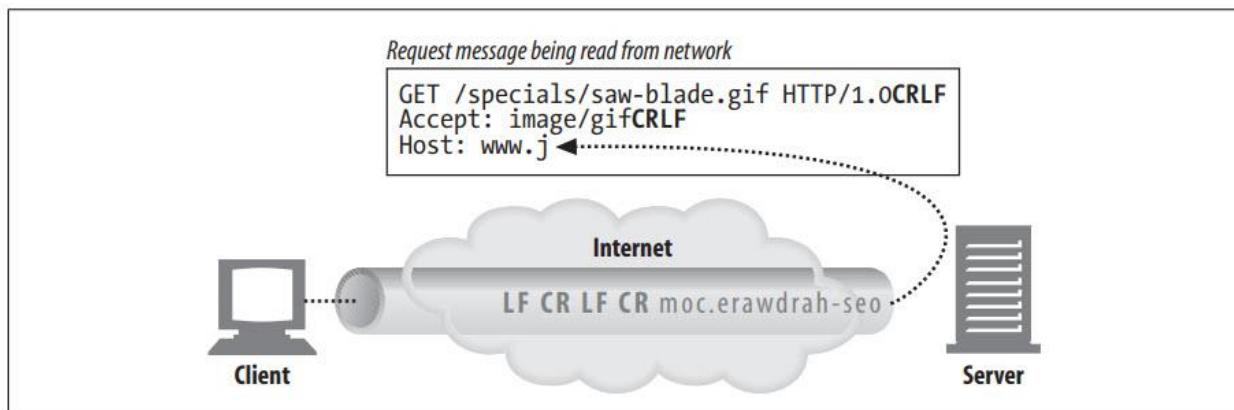




می‌توانید به وب سرورهای آپاچی بگویید که از جستجوی شناسایی با Apache's IdentityCheck استفاده کنند. اگر هیچ اطلاعات شناسایی در دسترس نباشد، آپاچی فیلدهای لاغ ident را با خط تیره (-) پر می‌کند. فایل‌های گزارش با Log Format رایج معمولاً حاوی خط فاصله در فیلد دوم هستند زیرا هیچ اطلاعات ident ای در دسترس نیست.

Step 2: Receiving Request Messages

همانطور که داده‌ها به Connection ها می‌رسد، وب سرور داده‌ها را از Connection شبکه می‌خواند و قطعات پیام درخواست را تجزیه می‌کند.



هنگام تجزیه پیام درخواست، وب سرور:

- در Request Line به دنبال متدهای درخواست، شناسه منبع مشخص شده (URI) و Version Number می‌باشد، که هر کدام با یک فاصله از هم جدا می‌شوند و به یک carriage-return line-feed یا CR LF ختم می‌شوند.
- هدرهای پیام را می‌خواند (هر کدام که به CR LF ختم می‌شوند)
- خط خالی انتهای هدر را که به CR LF ختم می‌شود (در صورت وجود) تشخیص می‌دهد.
- متن درخواست (Request Body) را در صورت وجود می‌خواند (طول مشخص شده توسط هدر Content-Length)

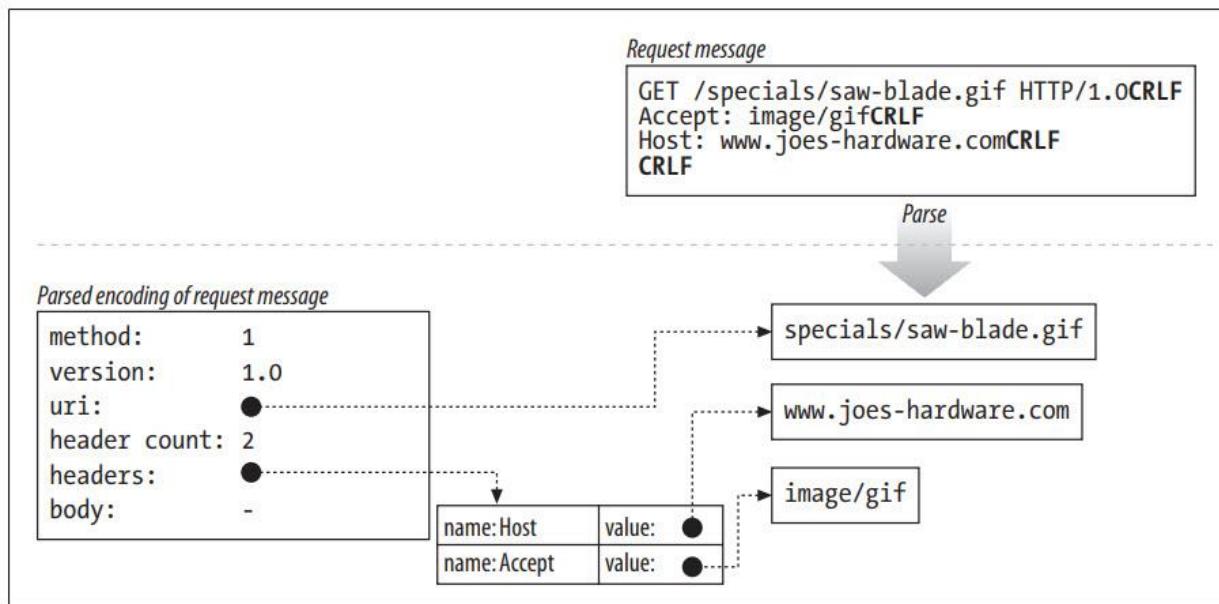
هنگام تجزیه پیام‌های درخواست، وب سرورها داده‌های ورودی را به طور نامنظم از شبکه دریافت می‌کنند. شبکه می‌تواند در هر نقطه‌ای متوقف شود. وب سرور باید داده‌ها را از شبکه بخواند و به طور موقت داده‌های جزئی پیام را در حافظه ذخیره کند تا زمانی که داده‌های کافی برای تجزیه و درک آن را دریافت شود.





Internal Representations of Messages

برخی از وب سرورها نیز پیام‌های درخواست را در ساختارهای داده داخلی ذخیره می‌کنند که دستکاری پیام را آسان می‌کند. به عنوان مثال، ساختار داده ممکن است شامل نشانگرها و طول هر قطعه از پیام درخواست باشد و هدرها ممکن است در یک جدول جستجوی سریع ذخیره شوند تا مقادیر خاص هدرهای خاص به سرعت قابل دسترسی باشند.



Connection Input/Output Processing Architectures

وب سرورهای با کارایی بالا از هزاران اتصال همزمان پشتیبانی می‌کنند. این اتصالات به وب سرور اجازه می‌دهد تا با کلاینت‌ها در سراسر جهان ارتباط برقرار کند که هر کدام دارای یک یا چند اتصال به سرور هستند. برخی از این Connection‌ها ممکن است درخواست‌ها را به سرعت به وب سرور ارسال کنند، در حالی که سایر اتصالات درخواست‌ها را به آرامی یا به ندرت چک می‌کنند و برخی دیگر بیکار بوده و بی سر و صدا منتظر برخی از فعالیت‌های آینده هستند.

وب سرورها دائمًا به دنبال درخواست‌های وب جدید هستند، زیرا درخواست‌ها می‌توانند در هر زمانی وارد شوند.

همانطور که در شکل زیر نشان داده شده است:

وب سرورهای تک رشته‌ای یا Single-threaded (بخش a شکل)

وب سرورهای تک رشته‌ای هر بار یک درخواست را تا زمان تکمیل پردازش می‌کنند. هنگامی که تراکنش کامل شد، اتصال بعدی پردازش می‌شود. پیاده سازی این معماری ساده است، اما در حین





پردازش، همه Connection های دیگر نادیده گرفته می‌شوند. این روش مشکلات عملکرد جدی ایجاد می‌کند و فقط برای سرورهای کم بار و ابزارهای تشخیصی مانند type-o-serve مناسب است.

وب سرورهای چند فرآیندی(Multiprocess) و چند رشته‌ای (Multithreaded) (بخش b شکل)

وب سرورهای چند فرآیندی و چند رشته‌ای چندین فرآیند یا رشته‌های با کارایی بالاتر را به پردازش درخواست‌ها به طور همزمان اختصاص می‌دهند. برخی از سرورها برای هر اتصال، یک thread/process اختصاص می‌دهند، اما زمانی که یک سرور صدها، هزاران یا حتی دهها هزار اتصال همزمان را پردازش می‌کند، تعداد فرآیندها یا رشته‌های حاصل ممکن است حافظه یا منابع سیستم زیادی را مصرف کند. بنابراین، بسیاری از وب سرورهای چند رشته‌ای محدودیتی برای حداکثر تعداد موضوعات/فرآیندها قائل می‌شوند.

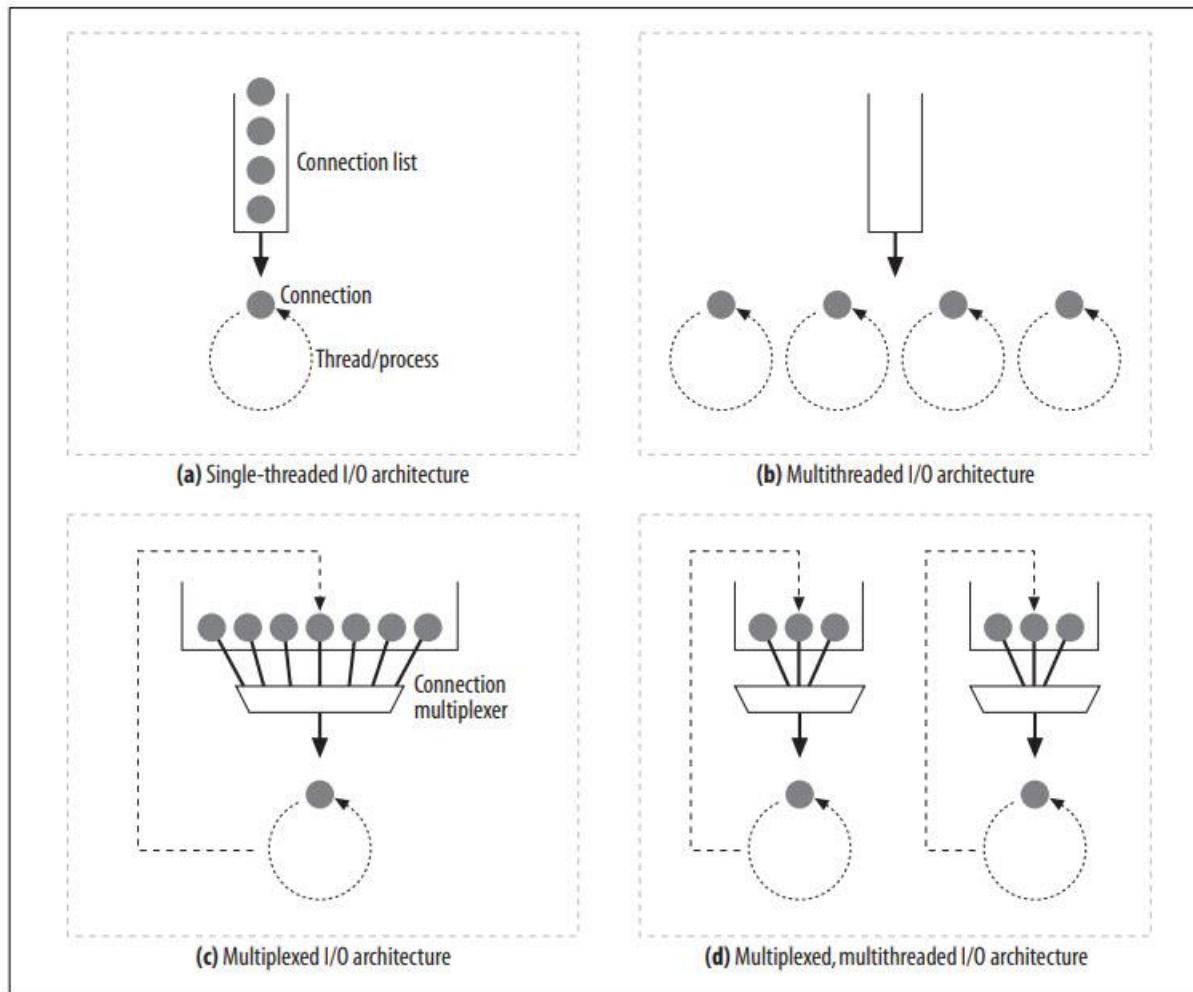
سرورهای ورودی/خروجی چندگانه یا I/O Multiplexed (بخش c شکل)

برای پشتیبانی از تعداد زیادی اتصال، بسیاری از سرورهای وب از معماری‌های چندگانه استفاده می‌کنند. در یک معماری مالتی پلکس، همه Connection ها به طور همزمان برای فعالیت، مشاهده می‌شوند. هنگامی که یک Connection، وضعیت را تغییر می‌دهد (به عنوان مثال، هنگامی که داده‌ها قابل دسترس می‌شوند یا یک خط رخ می‌دهد)، مقدار کمی پردازش روی Connection انجام می‌شود. هنگامی که پردازش کامل شد، Connection برای تغییر وضعیت بعدی به لیست Open Connection بازگردانده می‌شود. کار روی یک Connection تنها زمانی انجام می‌شود که کاری برای انجام دادن وجود داشته باشد. موضوعات و فرآیندها در انتظار Connection های بیکار نیستند.

وب سرورهای چند رشته‌ای چندگانه یا Multiplexed Multithreaded (بخش d شکل)

برخی از سیستم‌ها، چند رشته‌ای و مالتی پلکسی را برای استفاده از چندین CPU در پلتفرم رایانه ترکیب می‌کنند. رشته‌های متعدد (اغلب یک در هر پردازنده فیزیکی) هر کدام Connection های باز (یا زیر مجموعه‌ای از اتصالات باز) را مشاهده می‌کنند و مقدار کمی کار روی هر اتصال انجام می‌دهند.





Step 3: Processing Requests

هنگامی که وب سرور درخواستی را دریافت کرد، می‌تواند درخواست را با استفاده از متدهای دیگر اختیاری پردازش کند.

برخی از متدهای (به عنوان مثال، POST) به داده‌های Entity Body در پیام درخواست نیاز دارند. متدهای دیگر (مثلًا OPTIONS) به بدن درخواست اجازه می‌دهند اما به آن نیاز ندارند. چند متدهای دیگر (به عنوان مثال، GET) داده‌های Entity Body را در پیام‌های درخواست ممنوع می‌کنند.

Step 4: Mapping and Accessing Resources

وب سرورها سرورهای منبع یا Resource Servers هستند. آن‌ها محتوای از پیش ساخته شده، مانند صفحات HTML یا تصاویر JPEG و همچنین محتوای پویا را از برنامه‌های کاربردی تولید کننده منابع در حال اجرا بر روی سرورها ارائه می‌دهند.



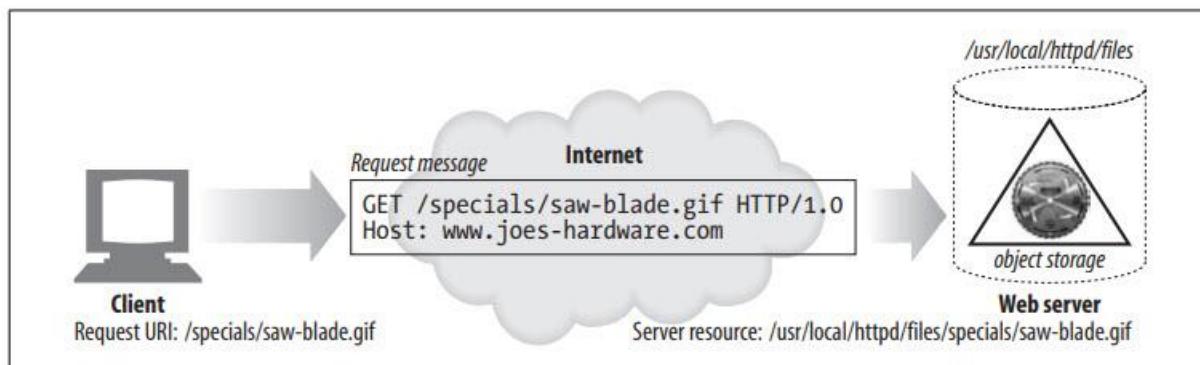


قبل از اینکه وب سرور بتواند محتوا را به کلاینت تحویل دهد، باید منبع محتوا را با Map نمودن URI از پیام درخواست به محتوا یا تولید کننده محتوای مناسب در سرور وب شناسایی کند.

Docroots

وب سرورها انواع مختلفی از نگاشت منابع یا Resource Mapping را پشتیبانی می‌کنند، اما ساده‌ترین شکل نگاشت منابع، از URI درخواست برای نامگذاری فایل در سیستم فایل وеб سرور استفاده می‌کند. به طور معمول، یک پوشش خاص در سیستم فایل وеб سرور برای محتوای وeb رزرو شده است. این پوشش ریشه سند یا docroot نام دارد. وeb سرور URI را از پیام درخواست می‌گیرد و به ریشه سند اضافه می‌کند.

در شکل زیر، درخواستی برای /specials/saw-blade.gif می‌رسد. وeb سرور در این مثال دارای ریشه سند /usr/local/httpd/files/specials/saw- است. بدین ترتیب وeb سرور فایل /usr/local/httpd/files/specials/saw-blade.gif را برمی‌گرداند.



برای تنظیم ریشه سند برای وeb سرور آپاچی، یک خط DocumentRoot را به فایل پیکربندی httpd.conf اضافه کنید:

`DocumentRoot /usr/local/httpd/files`

سرورها مراقب هستند که اجازه ندهند URL های نسبی از یک docroot پشتیبان تهیه کنند و سایر قسمت‌های سیستم فایل را در معرض دید قرار دهند. برای مثال، اکثر وeb سرورهای بالغ به این URI اجازه نمی‌دهند فایل‌های بالای ریشه سند سخت‌افزار Joe را ببینند:

`http://www.joes-hardware.com/..`

Virtually hosted docroots

وب سرورهایی که به صورت مجازی میزبانی می‌شوند، چندین وeb سایت را در یک وeb سرور میزبانی



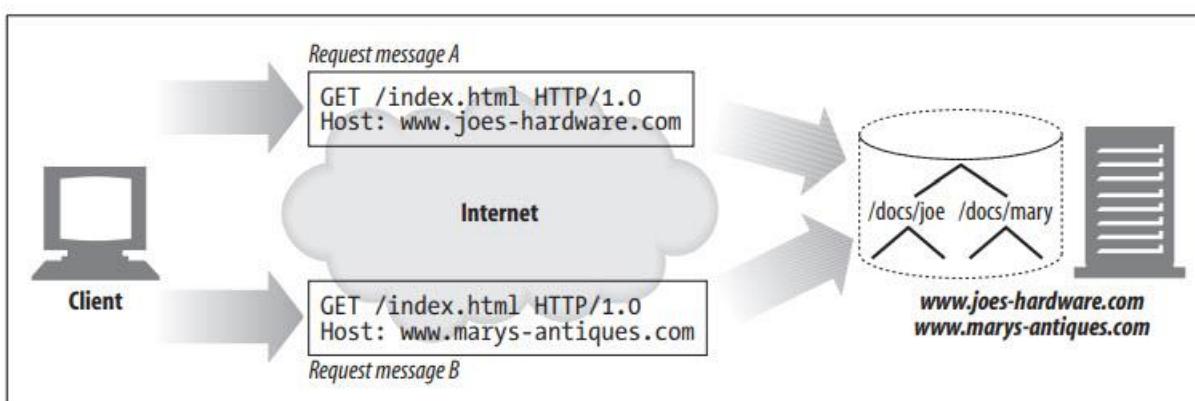


می‌کنند که هر سایت ریشه سند مجازی خود را در سرور دارد. یک وب سرور مجازی (Virtual Host)، ریشه سند صحیح را برای استفاده از آدرس IP یا نام میزبان در URI یا هدر Host شناسایی می‌کند. به این ترتیب، دو وبسایت میزبانی شده روی یک وب سرور می‌توانند محتوای کاملاً متمایز داشته باشند، حتی اگر URLs‌های درخواست یکسان باشند.

در شکل زیر، سرور میزبان دو سایت است:

www.joes-hardware.com

www.marysantiques.com



سرور می‌تواند وب سایتها را با استفاده از هدر Host یا از آدرس‌های IP متمایز کند.

- هنگامی که درخواست A می‌رسد، سرور فایل را برای Apache/.docs/joe/index.html می‌کند.
- هنگامی که درخواست B می‌رسد، سرور فایل را برای Apache/.docs/mary/index.html می‌کند.

پیکربندی docroots به صورت مجازی برای اکثر سرورهای وب، ساده است. برای وب سرور محبوب Apache باید یک блوک VirtualHost برای هر وب سایت مجازی پیکربندی کنید و DocumentRoot را برای هر سرور مجازی اضافه کنید.

```
<VirtualHost www.joes-hardware.com>
```

```
  ServerName www.joes-hardware.com
```

```
  DocumentRoot /docs/joe
```

```
  TransferLog /logs/joe.access_log
```

```
  ErrorLog /logs/joe.error_log
```



```
<VirtualHost>
```

```
<VirtualHost www.marys-antiques.com>
```

```
  ServerName www.marys-antiques.com
```

```
  DocumentRoot /docs/mary
```

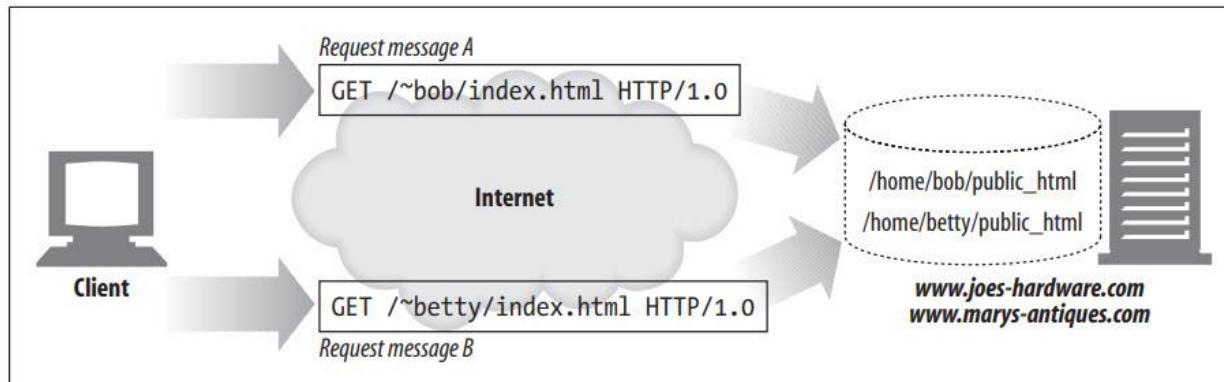
```
  TransferLog /logs/mary.access_log
```

```
  ErrorLog /logs/mary.error_log
```

```
</VirtualHost>
```

User home directory docroots

یکی دیگر از استفاده‌های رایج از docroots، به افراد وب سایتها خصوصی در یک وب سرور می‌دهد. یک قرارداد معمولی URI‌هایی را که مسیرهای آنها با یک اسلش و tilde (~) شروع می‌شود، به همراه یک نام کاربری به یک ریشه سند خصوصی برای آن کاربر Map می‌کند. Docroot خصوصی اغلب پوشه‌ای به نام public_html در دایرکتوری اصلی آن کاربر است، اما می‌توان آن را به شکلی متفاوت پیکربندی کرد.



Directory Listings

یک وب سرور می‌تواند در جایی که مسیر به یک دایرکتوری Resolve می‌شود، درخواست‌هایی را برای URL‌های مربوط به دایرکتوری دریافت کند. اکثر وب سرورها را می‌توان به گونه‌ای پیکربندی کرد که هنگام درخواست یک سرویس گیرنده URL دایرکتوری، اقدامات مختلفی را انجام دهن:





- یک پیام خطا بازگرداند.
- یک خطای خاص، پیش فرض و Index File به جای دایرکتوری بازگرداند.
- دایرکتوری را اسکن کند و یک صفحه HTML حاوی محتويات را برگردانید.

اکثر وب سرورها به دنبال فایلی به نام index.html یا index.htm در داخل یک دایرکتوری برای نمایش آن دایرکتوری می‌گردند. در صورتی که کاربر یک URL را برای یک دایرکتوری درخواست نماید و دایرکتوری شامل یک فایل با نام index.htm (یا index.html) باشد، سرور محتوای آن را باز می‌گردد.

در وب سرور آپاچی، می‌توانید مجموعه‌ای از نام فایل‌ها را که به عنوان فایل‌های فهرست پیش‌فرض تفسیر می‌شوند، با استفاده از دستور العمل پیکربندی DirectoryIndex پیکربندی کنید. دستور العمل DirectoryIndex همه نام‌های فایلی را که به عنوان فایل فهرست، ارائه می‌شوند، به ترتیب ترجیحی فهرست می‌کند. خط پیکربندی زیر باعث می‌شود Apache در پاسخ به درخواست URL، فایل‌های فهرست شده را جستجو نموده و محتوای آن را بازگردد:

`DirectoryIndex index.html index.htm home.html home.htm index.cgi`

اگر زمانی که کاربر URI دایرکتوری را درخواست می‌کند، هیچ فایل فهرست پیش‌فرضی وجود نداشته باشد و اگر فهرست‌های دایرکتوری غیرفعال نباشند، بسیاری از سرورهای وب به طور خودکار یک فایل HTML فهرست فایل‌های موجود که در آن نام فایل‌ها، اندازه آن‌ها و تاریخ‌های اصلاح هر فایل را برمی‌گردانند. این موضوع اما به افراد سودجو اجازه می‌دهد تا فایل‌هایی را که در یک سرور وب قرار دارند، شناسایی نماید که در حالت عادی معمولاً شناسایی آن‌ها امکان پذیر نیست.

شما می‌توانید تولید خودکار فایل‌های فهرست دایرکتوری را با دستور Apache زیر غیرفعال کنید:

Options –Indexes

Dynamic Content Resource Mapping

سرورهای وب همچنین می‌توانند URI ها را به منابع پویا Map کنند (یعنی به برنامه‌هایی که بر اساس تقاضا محتوا تولید می‌کنند). در واقع، یک کلاس کامل از وب سرورها به نام Application Server، سرورهای وب را به برنامه‌های کاربردی پیشرفته متصل می‌کنند. وب سرور باید بتواند تشخیص دهد که چه زمانی یک منبع، یک منبع پویا است، برنامه تولید کننده محتوای پویا در کجا قرار دارد و چگونه برنامه را اجرا کند. اکثر وب سرورها مکانیسم‌های اساسی برای شناسایی و Mapping از منابع پویا ارائه می‌دهند.





آپاچی به شما این امکان را می‌دهد که اجزای مسیر URI را در دایرکتوری‌های برنامه اجرایی Map کنید. هنگامی که یک سرور درخواستی برای یک URI با یک جزء مسیر اجرایی دریافت می‌کند، سعی می‌کند یک برنامه را در یک فهرست سرور مربوطه اجرا کند. برای مثال، دستورالعمل پیکربندی آپاچی زیر مشخص می‌کند که همه URI هایی که مسیرشان با /cgi-bin/ شروع می‌شود، باید برنامه‌های مربوطه موجود در دایرکتوری /usr/local/etc/httpd/cgi-programs/ را اجرا کنند:

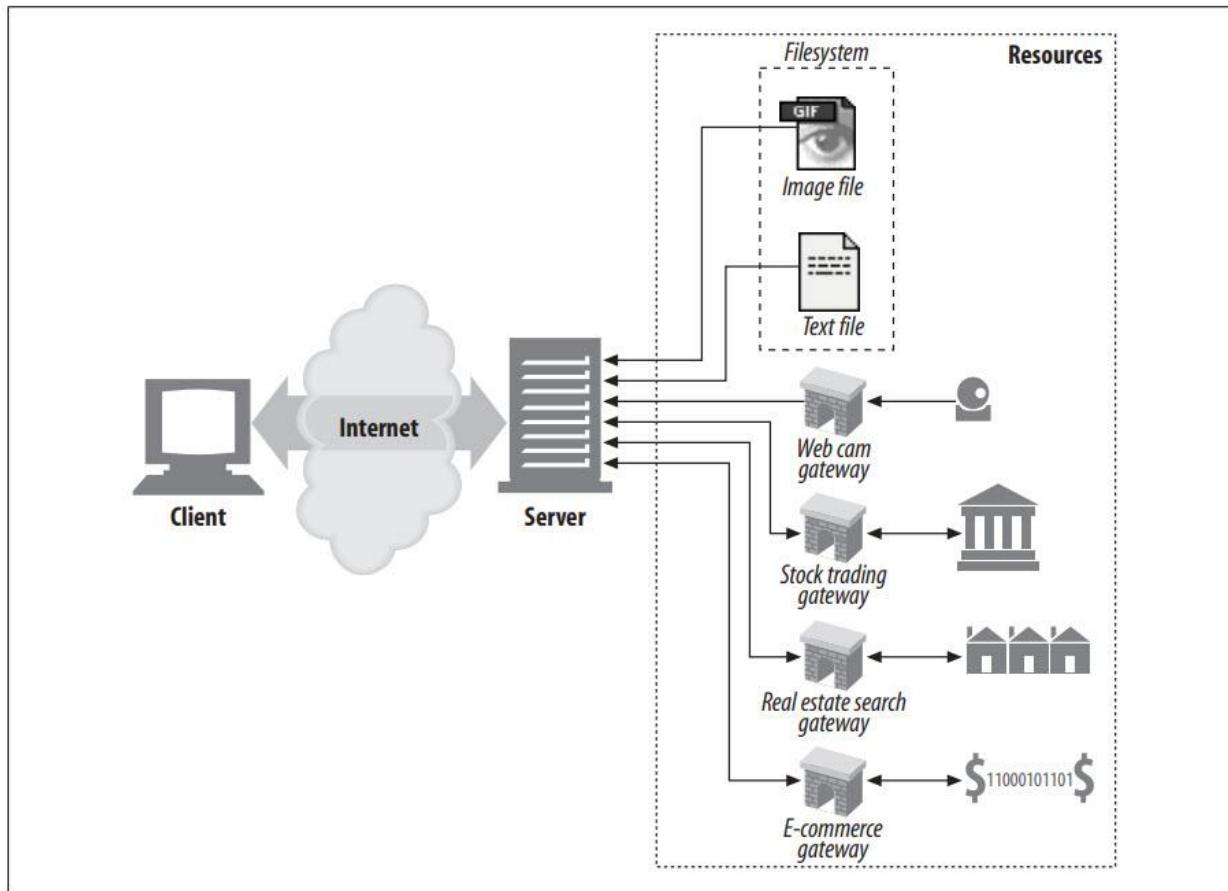
ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-programs/

آپاچی همچنین به شما امکان می‌دهد فایل‌های اجرایی را با پسوند فایل خاص علامت گذاری کنید. به این ترتیب، اسکریپت‌های اجرایی را می‌توان در هر دایرکتوری قرار داد. دستورالعمل پیکربندی آپاچی زیر مشخص می‌کند که تمام منابع وب که به .cgi ختم می‌شوند باید اجرا شوند:

AddHandler cgi-script .cgi

CGI یک رابط اولیه، ساده و محبوب برای اجرای برنامه‌های سمت سرور است. سرورهای کاربردی مدرن از محتوای پویای قدرتمندتر و کارآمدتر در سمت سرور پشتیبانی می‌کنند، از جمله آن‌ها می‌توان به Active Server Pages مایکروسافت و Java servlet اشاره نمود.





Server-Side Includes (SSI)

بسیاری از وب سرورها نیز از **Include**های سمت سرور پشتیبانی می‌کنند. اگر منبعی به عنوان حاوی-**server**-**side include** علامت گذاری شود، سرور محتویات منبع را قبل از ارسال به مشتری پردازش می‌کند.

محتویات برای الگوهای مشخص خاصی اسکن می‌شوند (اغلب در کامنت‌های خاص HTML موجود است)، که می‌توانند نام متغیرها یا اسکریپت‌های **Embedded** باشند. الگوهای ویژه با مقادیر متغیرها یا خروجی اسکریپت‌های اجرایی جایگزین می‌شوند. این یک راه آسان برای ایجاد محتوای پویا است.

Access Controls

وب سرورها همچنین می‌توانند کنترل‌های دسترسی را به منابع خاصی اختصاص دهند. هنگامی که درخواستی به یک منبع کنترل شده می‌رسد، وب سرور می‌تواند دسترسی را بر اساس آدرس IP کلاینت کنترل کند، یا می‌تواند یک چالش رمز عبور برای دسترسی به منبع صادر کند.





Step 5: Building Responses

هنگامی که وب سرور منبع را شناسایی کرد، عملی که در متدهای درخواست توضیح داده شده را انجام می‌دهد و پیام پاسخ را بر می‌گرداند. پیام پاسخ حاوی کد وضعیت پاسخ، هدرهای پاسخ و بدنه پاسخ در صورت ایجاد کد است.

Response Entities

اگر تراکنش یک بدنه پاسخ ایجاد کند، محتوا همراه با پیام پاسخ ارسال می‌شود. اگر بدنه‌ای وجود داشت، پیام پاسخ معمولاً شامل موارد زیر است:

- یک هدر Content-Type که نوع MIME بدنه پاسخ را توصیف می‌کند.
- یک هدر Content-Length که اندازه بدنه پاسخ را توصیف می‌کند.
- محتوای واقعی پیام

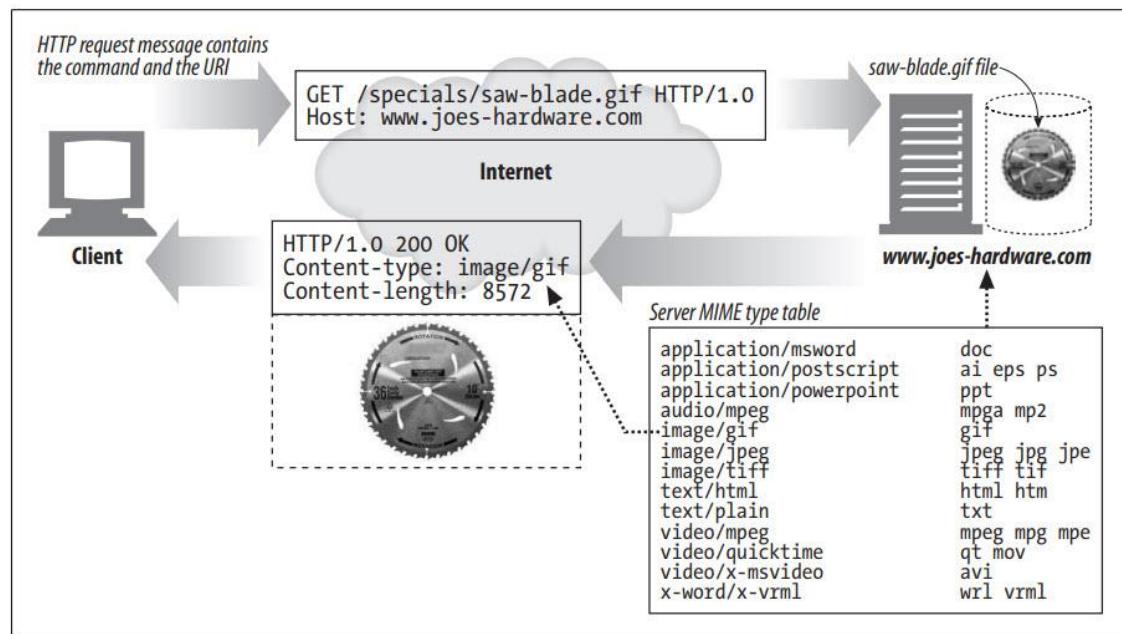
MIME Typing

وب سرور وظیفه تعیین نوع MIME بدنه پاسخ را بر عهده دارد. راههای زیادی برای پیکربندی سرورها برای مرتبط کردن انواع MIME با منابع وجود دارد:

mime.types

وب سرور می‌تواند از پسوند نام فایل برای نشان دادن نوع MIME استفاده کند. وب سرور یک فایل حاوی انواع extension-MIME را برای هر پسوند اسکن می‌کند تا نوع MIME را برای هر منبع محاسبه کند. این مدل-based، رایج ترین حالت در این بخش است.





Magic typing

وب سرور آپاچی می‌تواند محتویات هر منبع را اسکن کند و محتوا را با جدولی از الگوهای شناخته شده (به نام (magic file) مطابقت دهد تا نوع MIME را برای هر فایل تعیین نماید. این فرآیند می‌تواند کند باشد، اما راحت است، به خصوص اگر فایل‌ها بدون پسوند استاندارد نامگذاری شوند.

Explicit typing

سرورهای وب را می‌توان به گونه‌ای پیکربندی کرد که فایل‌ها یا محتویات دایرکتوری خاصی را مجبور به داشتن نوع MIME، صرف نظر از پسوند یا محتویات فایل کنند.

Type negotiation

برخی از وب سرورها را می‌توان به گونه‌ای پیکربندی کرد که یک منبع را در قالب‌های مختلف سند ذخیره کند. در این مورد، وب سرور را می‌توان به گونه‌ای پیکربندی نمود که «بهترین» قالب مورد استفاده (و نوع مربوطه) را توسط فرآیند مذاکره با کاربر تعیین کند. ما در این مورد در فصل‌های آینده بحث خواهد شد.

سرورهای وب همچنین می‌توانند پیکربندی شوند تا فایل‌های خاصی را با انواع MIME مرتبط کنند.

Redirection

وب سرورها گاهی اوقات به جای پیام‌های موفقیت آمیز، پاسخ‌های تغییر مسیر را بر می‌گردانند. یک وب سرور می‌تواند مرورگر را برای انجام درخواست، به جای دیگری هدایت کند. پاسخ تغییر مسیر با یک کد





بازگشتی 3xx نشان داده می شود. هدر Location در پاسخ، شامل یک URI برای مکان جدید یا مکان ترجیحی محتوا است. تغییر مسیر برای موارد زیر مفید است:

Permanently moved resources

ممکن است یک منبع به مکان جدیدی منتقل شده باشد، یا نام آن تغییر کرده باشد و یک URL جدید به آن بدهد. وب سرور می تواند به کلاینت بگوید که نام منبع تغییر کرده است و کلاینت می تواند قبل از Fetch نمودن منبع از مکان جدید، هر Bookmark و موارد دیگر را به روز کند. کد وضعیت 301 Moved Permanently برای این نوع تغییر مسیر استفاده می شود.

Temporarily moved resources

اگر منبعی به طور موقت منتقل یا تغییر نام داده شود، سرور ممکن است بخواهد کلاینت را به مکان جدید هدایت کند. اما، از آنجایی که تغییر نام موقتی است، سرور از کلاینت می خواهد که در آینده با URL قدیمی بازگردد و هیچ Bookmark ای را به روز نکند. کدهای وضعیت 303 See Other و 307 Temporary Redirect برای این نوع تغییر مسیر استفاده می شود.

URL augmentation

سرورها اغلب از تغییر مسیرها برای بازنویسی URLs و همچنین برای Embed Context استفاده می کنند. هنگامی که درخواست به سرور می رسد، سرور یک URL جدید حاوی اطلاعات وضعیت Embedded تولید می کند و کاربر را به این URL جدید هدایت می نماید. این یک روش مفید برای حفظ وضعیت در سراسر تراکنش ها است.

کدهای وضعیت 303 See Other و 307 Temporary Redirect برای این نوع تغییر مسیر استفاده می شود.

Load balancing

اگر سروری که Overloaded شده، درخواستی دریافت کند، سرور می تواند کلاینت را به سروری با Load هدایت کند. کدهای وضعیت 303 See Other و 307 Temporary Redirect برای این نوع تغییر مسیر استفاده می شود.





Server affinity

سرورهای وب ممکن است یکسری اطلاعات محلی برای کاربران خاصی داشته باشند. یک سرور می‌تواند کلاینت را به سروری هدایت کند که حاوی اطلاعاتی درباره کلاینت است. کدهای وضعیت 303 See Other و 307 Temporary Redirect برای این نوع تغییر مسیر استفاده می‌شود.

Canonicalizing directory names

هنگامی که یک کلاینت یک URI برای نام دایرکتوری بدون اسلش آخر درخواست می‌کند، بیشتر سرورهای وب، کلاینت را با اضافه کردن اسلش به یک URI هدایت می‌کنند تا پیوندهای نسبی به درستی کار کنند.

Step 6: Sending Responses

سرورهای وب با مشکلات مشابهی در ارسال داده در سراسر اتصالات مانند دریافت موافق می‌شوند. سرور ممکن است اتصالات زیادی به بسیاری از کلاینت‌ها داشته باشد، برخی غیرفعال هستند، برخی داده‌ها را به سرور ارسال می‌کنند و برخی داده‌های پاسخ را به کلاینت‌ها برمی‌گردانند.

سرور باید وضعیت اتصال را ردیابی کند و اتصالات دائمی را با دقت خاصی مدیریت کند. برای اتصالات غیرمداوم، از سرور انتظار می‌رود که هنگام ارسال کل پیام، سمت اتصال خود را بیندد.

برای اتصالات دائمی، اتصال ممکن است باز بماند، در این صورت سرور باید برای محاسبه صحیح هدر-Content-Length بسیار محظوظ باشد، در غیر این صورت کلاینت راهی برای دانستن زمان پایان پاسخ نخواهد داشت (به فصل ۴ مراجعه کنید).

Step 7: Logging

در نهایت، هنگامی که یک تراکنش کامل شد، وب سرور یک ورودی را در فایل Log یادداشت می‌کند که تراکنش انجام شده را توصیف می‌کند. اکثر وب سرورها چندین فرم قابل تنظیم از Logging را ارائه می‌دهند.





For More Information

<http://www.w3c.org/Jigsaw/>

<http://www.ietf.org/rfc/rfc1413.txt>

Apache: The Definitive Guide

Ben Laurie and Peter Laurie, O'Reilly & Associates, Inc.

Professional Apache

Peter Wainwright, Wrox Press.





فصل ششم - Proxies

سرورهای پروکسی وب، در واقع واسطه هستند. پروکسی‌ها بین کلاینت‌ها و سرورها قرار گرفته و به عنوان «واسطه‌ها» عمل می‌کنند و پیام‌های HTTP را بین طرفین به عقب و جلو می‌برند. این فصل درباره سرورهای پراکسی HTTP، پشتیبانی ویژه از ویژگی‌های پروکسی و برخی از رفتارهای پیچیده‌ای که در هنگام استفاده از سرورهای پروکسی مشاهده خواهد کرد صحبت می‌کند.

مباحثی که در این فصل به آن‌ها پرداخته می‌شود عبارتند از:

- توضیح پروکسی‌های HTTP، نحوه قرار گرفتن آن‌ها در مقابل دروازه‌های و نحوه استقرار پروکسی‌ها
- توضیح برخی از راههایی که پروکسی‌ها در آن مفید بوده و به ما کمک می‌کنند.
- توضیح نحوه استقرار پروکسی‌ها در شبکه‌های واقعی و نحوه هدایت ترافیک به سرورهای پروکسی
- نحوه پیکربندی مرورگر برای استفاده از پروکسی
- نشان دادن درخواست‌های پروکسی HTTP، تفاوت آن‌ها با درخواست‌های سرور و اینکه چگونه پروکسی‌ها می‌توانند رفتار مرورگرها را تغییر دهند.
- توضیح اینکه چگونه می‌توانید مسیر پیام‌های خود را از طریق زنجیره‌های سرورهای پروکسی با استفاده از هدرهای Via و متدهای TRACE ضبط کنید.
- توضیح کنترل دسترسی HTTP مبتنی بر پروکسی
- توضیح اینکه چگونه پروکسی‌ها می‌توانند بین کلاینت‌ها و سرورهایی که هر کدام از ویژگی‌ها و نسخه‌های متفاوتی پشتیبانی می‌کنند، تعامل داشته باشند.

Web Intermediaries

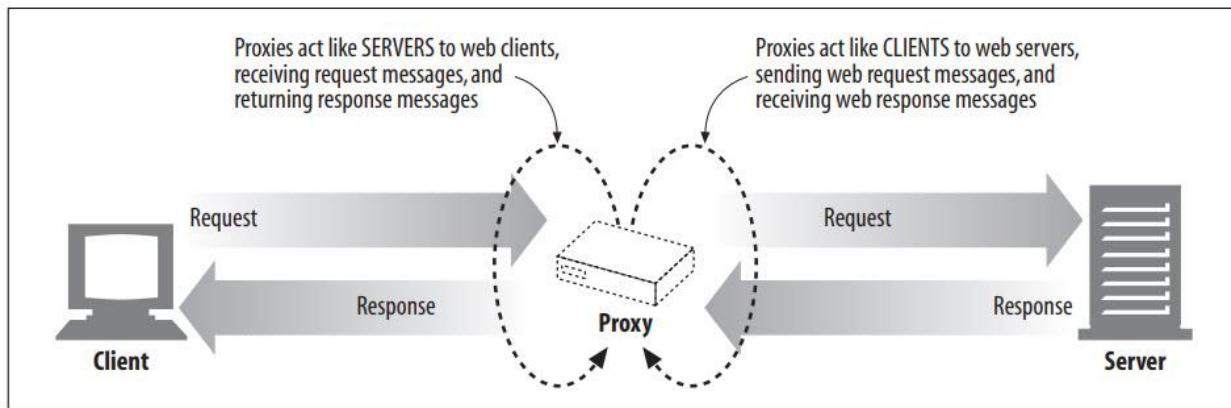
سرورهای پروکسی وب، واسطه‌هایی هستند که تراکنش‌ها را از طرف کلاینت انجام می‌دهند. بدون پروکسی وب، کلاینت‌های HTTP مستقیماً با سرورهای HTTP صحبت می‌کنند. با یک پروکسی وب، کلاینت در عوض با پروکسی صحبت می‌کند که خود از طرف کلاینت با سرور ارتباط برقرار می‌نماید. کلاینت همچنان تراکنش را کامل می‌کند، اما از طریق خدمات خوب سرور پروکسی.

سرورهای پروکسی HTTP، هم سرورهای وب و هم سرویس گیرندگان HTTP پیام‌های درخواستی را به پروکسی‌ها ارسال می‌کنند، سرور پروکسی باید به درستی درخواست‌ها و اتصالات را مدیریت کند و پاسخ‌ها را درست مانند وب سرور بازگرداند. در همان زمان، خود پروکسی درخواست‌هایی را به سرورها ارسال می‌کند، بنابراین باید مانند یک کلاینت HTTP صحیح رفتار کند و





در خواست‌ها را ارسال نموده و پاسخ‌ها را دریافت کند (شکل زیر). اگر در حال ایجاد پروکسی HTTP خود هستید، باید قوانین را برای سرویس گیرنده‌گان HTTP و سرورهای HTTP به دقت دنبال کنید.



Private and Shared Proxies

یک سرور پروکسی می‌تواند به یک کلاینت اختصاص داده شود یا بین بسیاری از کلاینت‌ها به اشتراک گذاشته شود. پروکسی‌هایی که به یک کلاینت اختصاص داده شده‌اند، پراکسی‌های خصوصی نامیده می‌شوند. پروکسی‌هایی که در بین کلاینت‌های متعدد به اشتراک گذاشته می‌شوند، پروکسی‌های عمومی نامیده می‌شوند.

Public proxies

اکثر پروکسی‌ها، پراکسی‌های عمومی و مشترک هستند. مدیریت یک پروکسی متمرکز مقرن به صرفه‌تر و آسان‌تر است. برخی از برنامه‌های پروکسی، مانند ذخیره‌سازی سرورهای پراکسی، زمانی که کاربران بیشتری به همان سرور پراکسی هدایت می‌شوند، مفیدتر می‌شوند، زیرا می‌توانند از درخواست‌های رایج بین کاربران استفاده کنند.

Private proxies

پروکسی‌های خصوصی اختصاصی چندان رایج نیستند، اما جایگاه خود را دارند، به خصوص زمانی که مستقیماً روی رایانه کلاینت اجرا شوند. برخی از محصولات دستیار مرورگر یا Browser Assistant و همچنین برخی از خدمات ISP، پروکسی‌های کوچکی را مستقیماً بر روی رایانه شخصی کاربر اجرا می‌کنند تا ویژگی‌های مرورگر را گسترش دهند، عملکرد را بهبود بخشدند یا تبلیغات را برای خدمات رایگان ISP میزبانی کنند.

Proxies Versus Gateways

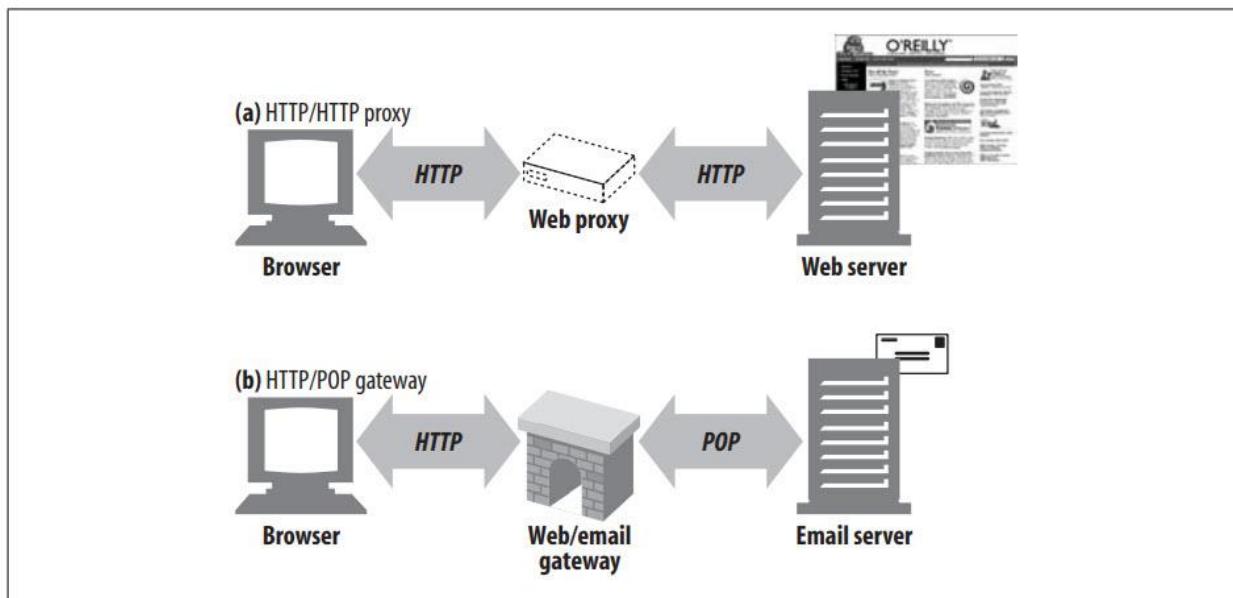
به بیان دقیق‌تر، پروکسی دو یا چند برنامه که با پروتکل یکسانی با یکدیگر صحبت می‌کنند را به هم متصل می‌کند، در حالی که Gateway دو یا چند طرف که پروتکل‌های متفاوتی دارند را به هم متصل





می‌کند. یک Gateway به عنوان یک "Protocol Converter" عمل می‌کند و به کلاینت اجازه می‌دهد تا تراکنش را با یک سرور انجام دهد، حتی زمانی که کلاینت و سرور دارای پروتکل‌های متفاوتی هستند.

شکل زیر تفاوت بین پروکسی‌ها و Gateway‌ها را نشان می‌دهد:



- دستگاه واسطه در بخش a شکل، یک پروکسی HTTP است، زیرا پروکسی از طریق HTTP هم با کلاینت و هم سرور صحبت می‌کند.
- دستگاه واسطه در بخش b شکل یک Gateway یا دروازه HTTP/POP است، زیرا یک فرانت اند HTTP را به پشتیبان ایمیل POP متصل می‌کند. این Gateway تراکنش‌های وب را به تراکنش‌های مناسب POP تبدیل می‌کند تا به کاربر اجازه دهد ایمیل را از طریق HTTP بخواند. برنامه‌های ایمیل مبتنی بر وب مانند MSN Hotmail و Yahoo! Mail یا HTTP Email دروازه‌های ایمیل Gateway هستند.

در عمل، تفاوت بین پروکسی‌ها و Gateway‌ها مبهم است. از آنجایی که مرورگرها و سرورها نسخه‌های مختلف HTTP را پیاده سازی می‌کنند، پروکسی‌ها اغلب مقداری از تبدیل پروتکل را انجام می‌دهند و سرورهای پروکسی تجاری عملکرد Gateway را برای پشتیبانی از پروتکل‌های امنیتی SSL، فایروال‌های SOCKS، دسترسی و برنامه‌های کاربردی مبتنی بر وب پیاده سازی می‌کنند. در فصل ۸ بیشتر در مورد Gateway‌ها صحبت خواهیم کرد.



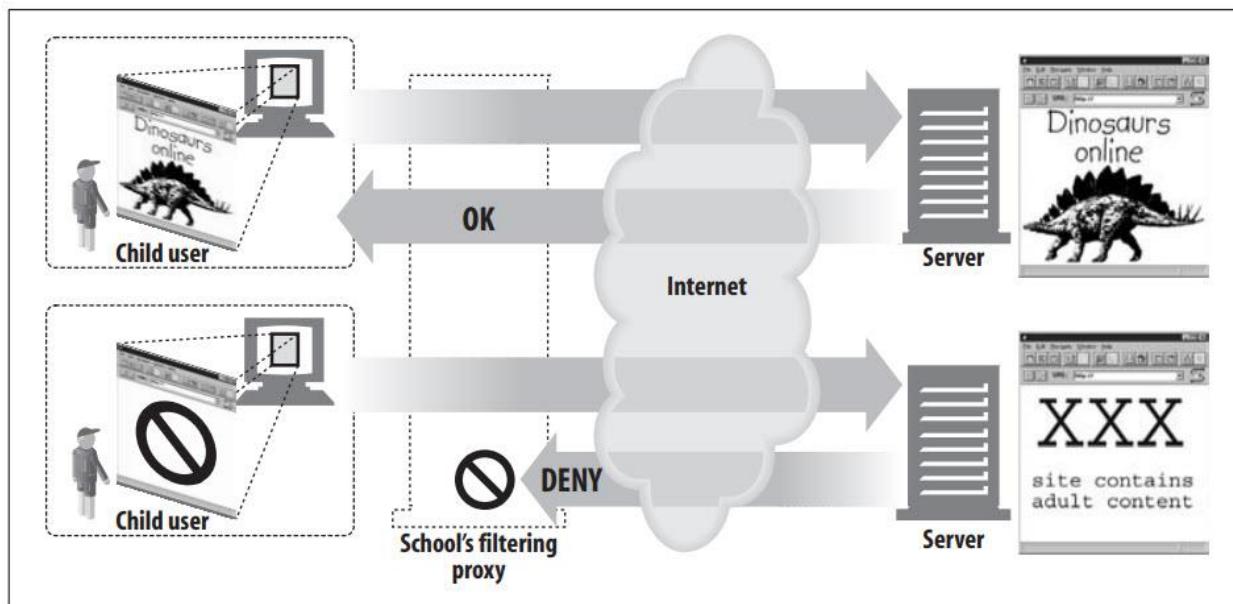


Why Use Proxies?

سرورهای پروکسی می‌توانند انواع کارهای مفید و کاربردی را انجام دهند. آن‌ها می‌توانند امنیت را بهبود بخشد، عملکرد را افزایش دهند و در هزینه صرفه جویی کنند. از آنجایی که سرورهای پروکسی می‌توانند تمام ترافیک عبوری HTTP را ببینند و لمس کنند، پروکسی‌ها می‌توانند ترافیک را برای پیاده‌سازی بسیاری از سرویس‌های وب با ارزش افزوده مفید، نظارت و تغییر دهند. در اینجا چند مورد از روش‌هایی که می‌توان از پروکسی‌ها استفاده نمود آورده شده است:

Child filter

مدارس ابتدایی از پراکسی‌های فیلتر برای مسدود کردن دسترسی به محتوای بزرگسالان استفاده می‌کنند، در حالی که دسترسی بدون مانع به سایتها آموزشی را فراهم می‌کنند. همانطور که در شکل زیر نشان داده شده است، پروکسی ممکن است اجازه دسترسی نامحدود به محتوای آموزشی را بدهد، اما از دسترسی به سایتها که برای کودکان نامناسب است جلوگیری کند.



Document access controller

از سرورهای پروکسی می‌توان برای پیاده‌سازی یک استراتژی کنترل دسترسی یکنواخت در مجموعه بزرگی از وب سرورها و منابع وب و ایجاد یک مسیر حسابرسی استفاده کرد. این در تنظیمات شرکت‌های بزرگ یا سایر بوروکراسی‌های توزیع شده مفید است.

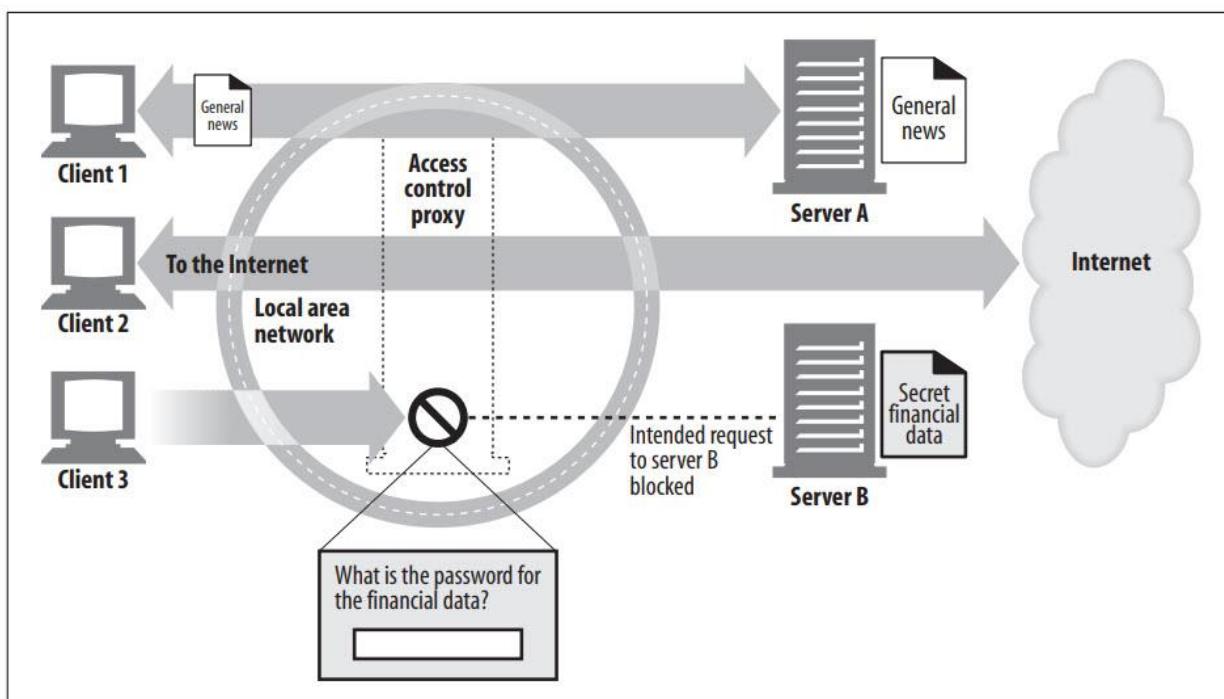




تمام کنترل‌های دسترسی را می‌توان بدون نیاز به بروزرسانی کنترل‌های دسترسی به طور مکرر در سرورهای وب متعدد، با مدل‌ها و ساختارهای مختلف، که توسط سازمان‌های مختلف مدیریت می‌شوند، بر روی سرور پروکسی متتمرکز پیکربندی کرد.

در شکل زیر، پروکسی کنترل دسترسی متتمرکز:

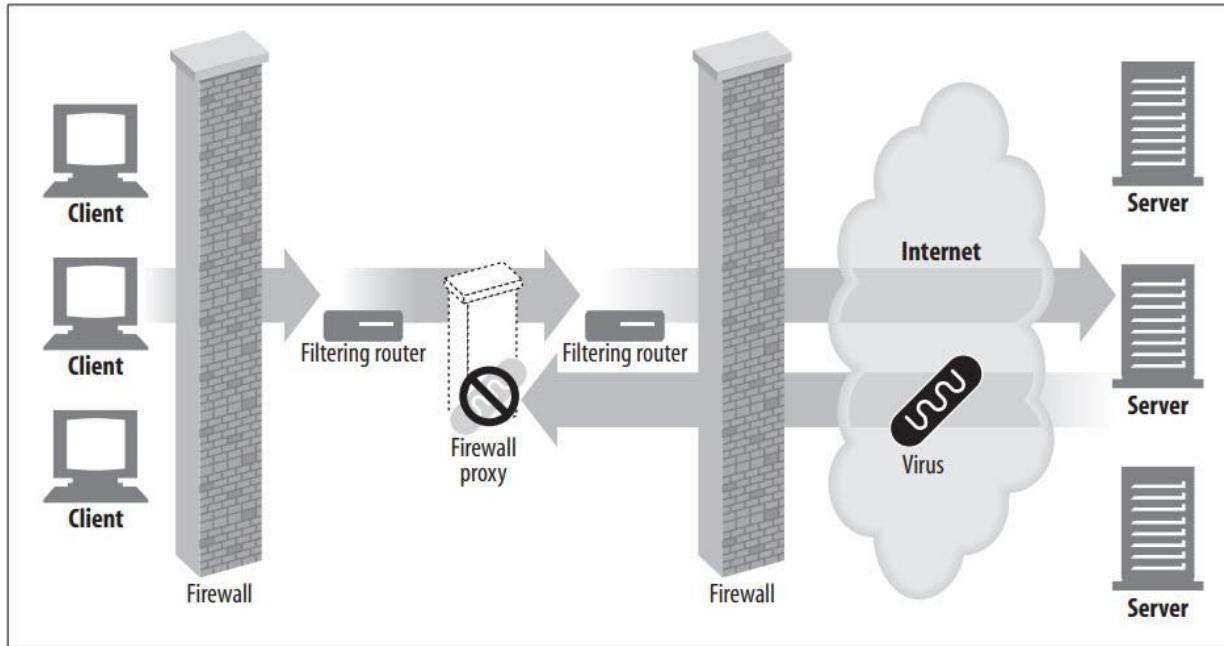
- به کلاینت یک اجازه می‌دهد بدون محدودیت به صفحات اخبار از سرور A دسترسی داشته باشد.
- دسترسی نامحدود به محتوای اینترنت را برای کلاینت دو فراهم می‌کند.
- همچنین قبل از اجازه دسترسی به سرور B، به یک رمز عبور از کلاینت سه نیاز دارد.



Security firewall

مهندسان امنیت شبکه اغلب از سرورهای پروکسی برای افزایش امنیت و اعمال محدودیت استفاده می‌کنند. بدین صورت که کدام پروتکل‌های سطح Application به داخل و خارج از یک سازمان، در یک نقطه امن در شبکه جریان پیدا کنند. آن‌ها همچنین می‌توانند قلاب‌هایی (hooks) برای بررسی دقیق آن ترافیک فراهم کنند، به همان شکلی که توسط پروکسی‌های وب و ایمیل حذف کننده ویروس استفاده می‌شود.



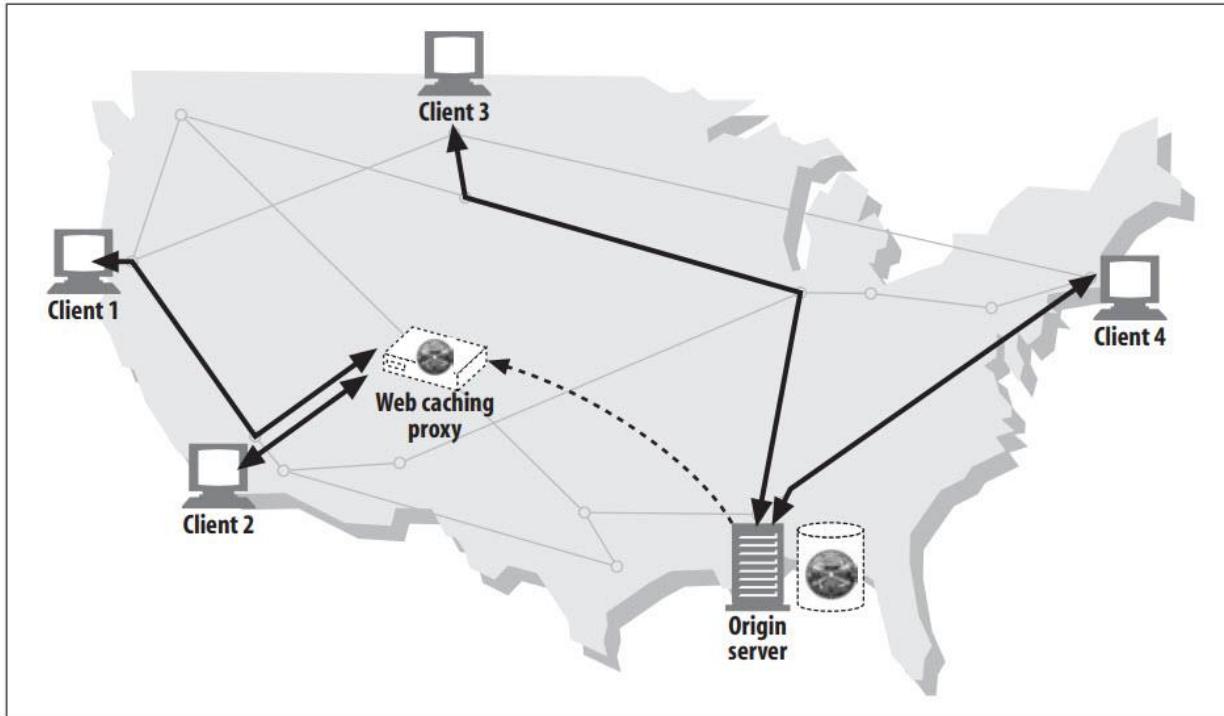


Web cache

کپی‌های محلی اسناد محبوب را نگهداری می‌کند و آن‌ها را در صورت تقاضا ارائه می‌دهد و از ارتباطات اینترنتی کند و پرهزینه می‌کاهمند.

در شکل زیر، کلاینت‌های ۱ و ۲ به شی A از cache وب نزدیک دسترسی دارند، در حالی که کلاینت‌های ۳ و ۴ از سرور مبدأ به سند دسترسی دارند.



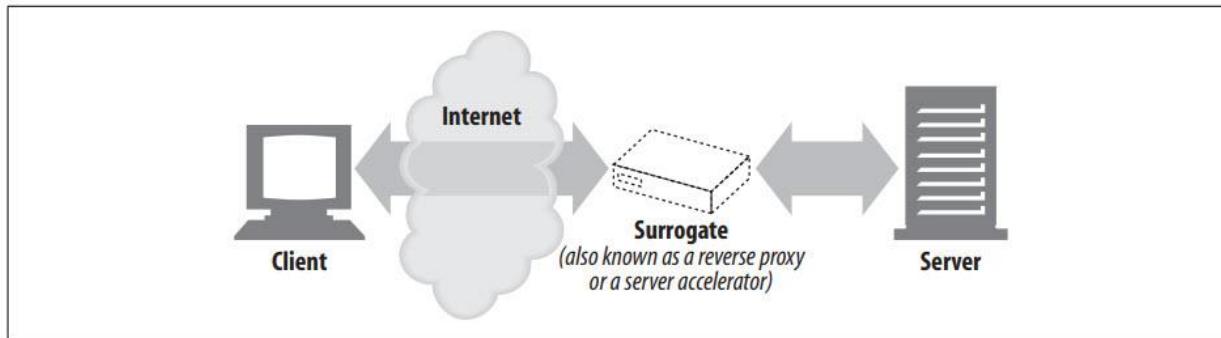


Surrogate

پروکسی‌ها می‌توانند به عنوان سرورهای وب ظاهر شوند. این به اصطلاح جانشین‌ها (Surrogates) یا Reverse Proxy درخواست‌های وب سرور واقعی را دریافت می‌کنند، اما، برخلاف سرورهای وب، ممکن است ارتباط با سرورهای دیگر را برای یافتن محتوای درخواستی در صورت تقاضا آغاز نماید.

Surrogate ها ممکن است برای بهبود عملکرد وب سرورهای کند برای محتوای رایج استفاده شوند. در این پیکربندی، Surrogate ها اغلب شتاب دهنده سرور یا Server Accelerator نامیده می‌شوند. همچنین می‌توانند در ارتباط با عملکرد مسیریابی محتوا برای ایجاد شبکه‌های توزیع شده از محتوای تکراری بر اساس تقاضا استفاده شوند.



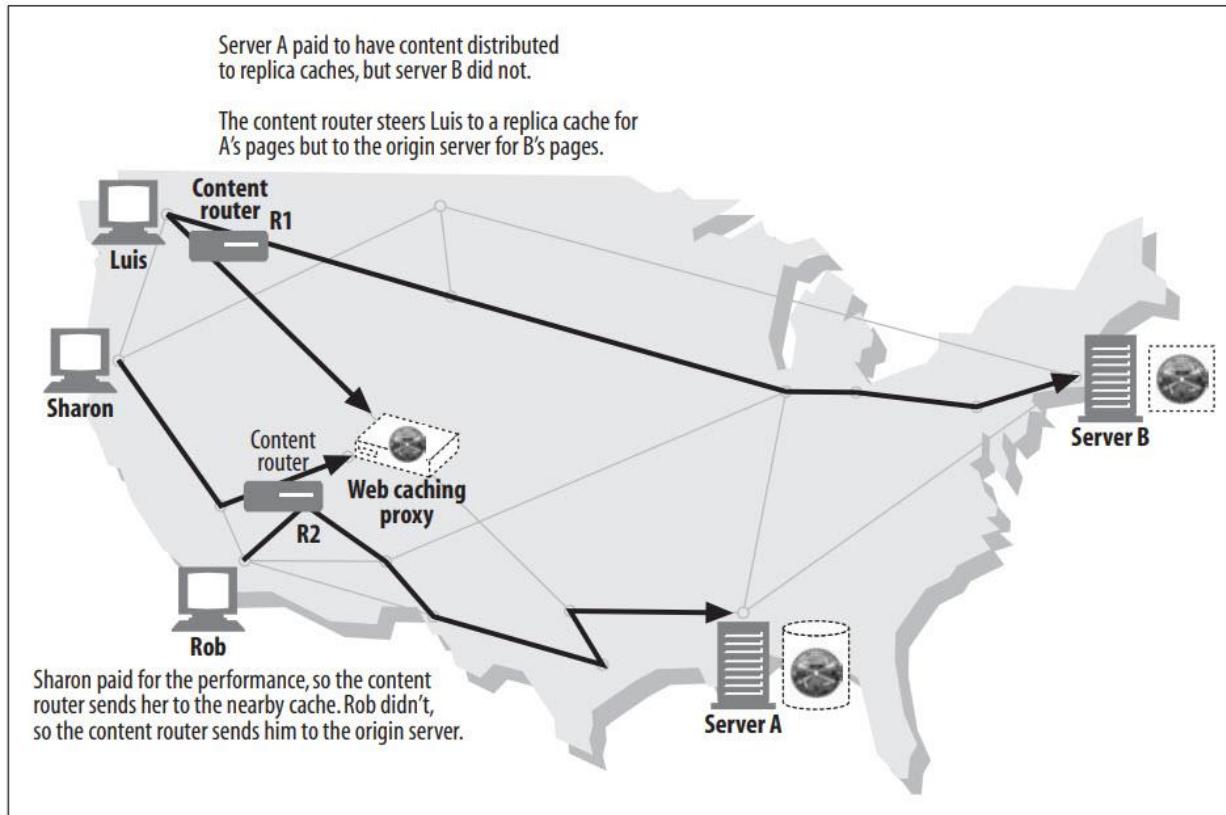


Content Router

سرورهای پروکسی می‌توانند به عنوان "content routers" عمل کنند و درخواست‌ها را بر اساس شرایط ترافیک اینترنت و نوع محتوا به سرورهای وب خاص منتقل کنند.

Content Router ها همچنین می‌توانند برای پیاده سازی پیشنهادات مختلف در سطح خدمات استفاده شوند. برای مثال، اگر کاربر یا ارائه‌دهنده محتوا برای عملکرد بالاتر هزینه پرداخت کرده باشد، روترهای محتوا می‌توانند درخواست‌ها را به Cache مشابه ارسال کنند یا اگر کاربر برای یک سرویس فیلتر ثبت‌نام کرده باشد، درخواست‌های HTTP را از طریق پراکسی‌های فیلتر هدایت کنند. بسیاری از خدمات جالب را می‌توان با استفاده از پروکسی‌های مسیریابی محتوای تطبیقی ساخت.





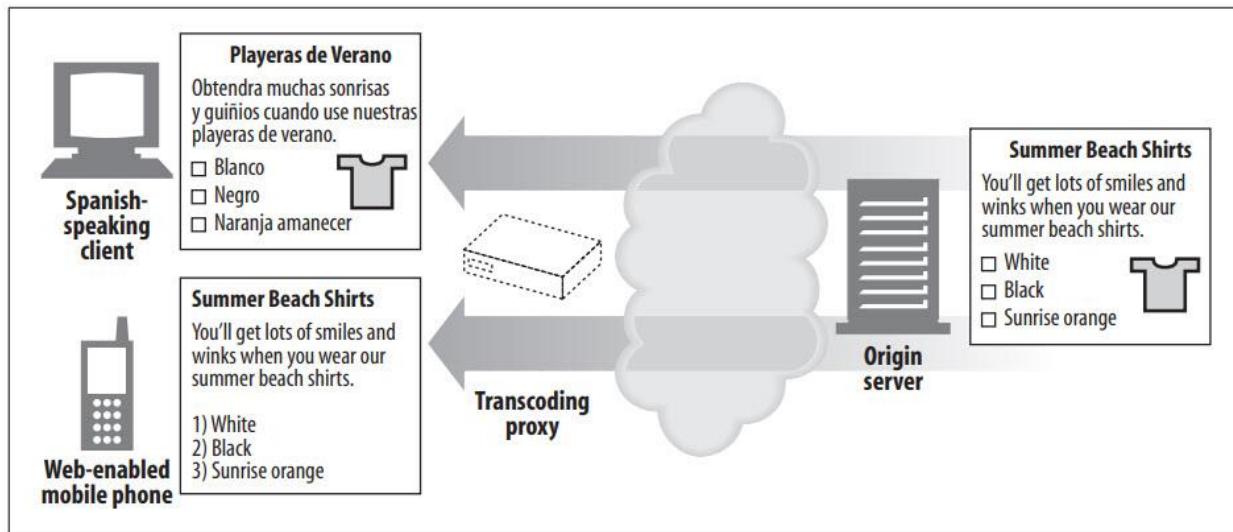
Transcoder

سرورهای پروکسی می‌توانند قالب متنی محتوا را قبل از تحویل آن به کلاینت‌ها تغییر دهند. این ترجمه شفاف (بين نمایش داده‌ها، **Transcoding** / **Transparent Translation**) می‌شود.

پروکسی‌های **Transcoding** می‌توانند تصاویر GIF را در حین انتقال به تصاویر JPEG تبدیل کنند تا اندازه را کاهش دهند. همچنین می‌توان تصاویر را کوچک نموده و شدت رنگ را کاهش داد تا در تلویزیون قابل مشاهده باشند. به همین ترتیب، فایل‌های متنی را می‌توان فشرده کرد و خلاصه‌های متنی کوچکی از صفحات وب برای صفحات فعال اینترنت و تلفن‌های هوشمند تولید کرد. حتی برای پروکسی‌ها امکان تبدیل اسناد به زبان‌های خارجی در حین انتقال وجود دارد!

شکل زیر یک پروکسی **Transcoding** را نشان می‌دهد که متن انگلیسی را به متن اسپانیایی تبدیل می‌کند و همچنین صفحات HTML را به متن ساده‌تری که می‌تواند بر روی صفحه نمایش کوچک تلفن همراه نمایش داده شود، دوباره قالب‌بندی می‌کند.





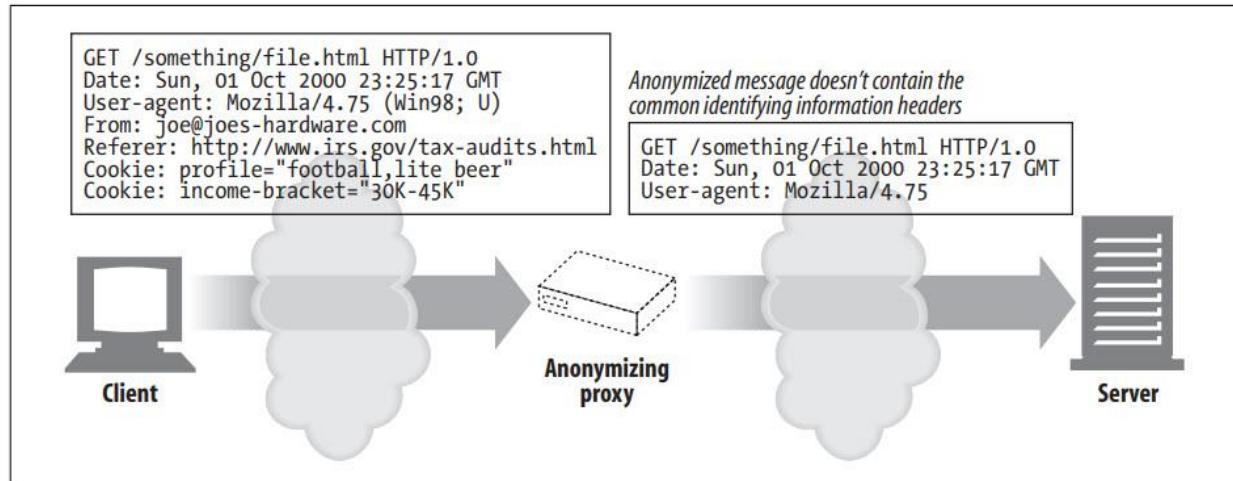
Anonymizer

پروکسی‌های ناشناس با حذف فعال مشخصه‌های شناسایی از پیام‌های HTTP (مانند آدرس IP کلاینت، از هدر، Referer، کوکی‌ها، شناسه‌های نشست URI) حریم خصوصی و ناشناس بودن را افزایش می‌دهند.

در شکل زیر، پروکسی ناشناس، تغییرات زیر را در پیام‌های کاربر برای افزایش حریم خصوصی ایجاد می‌کند:

- حذف نام کاربری سیستم و سیستم عامل کاربر از هدر User-Agent
- حذف هدر From برای محافظت از آدرس ایمیل کاربر
- حذف هدر Referer به منظور اینکه سایتها دیگری که کاربر بازدید کرده است مبهم باشد.
- حذف هدرهای کوکی برای حذف مشخصات و داده‌های هویتی





Where Do Proxies Go?

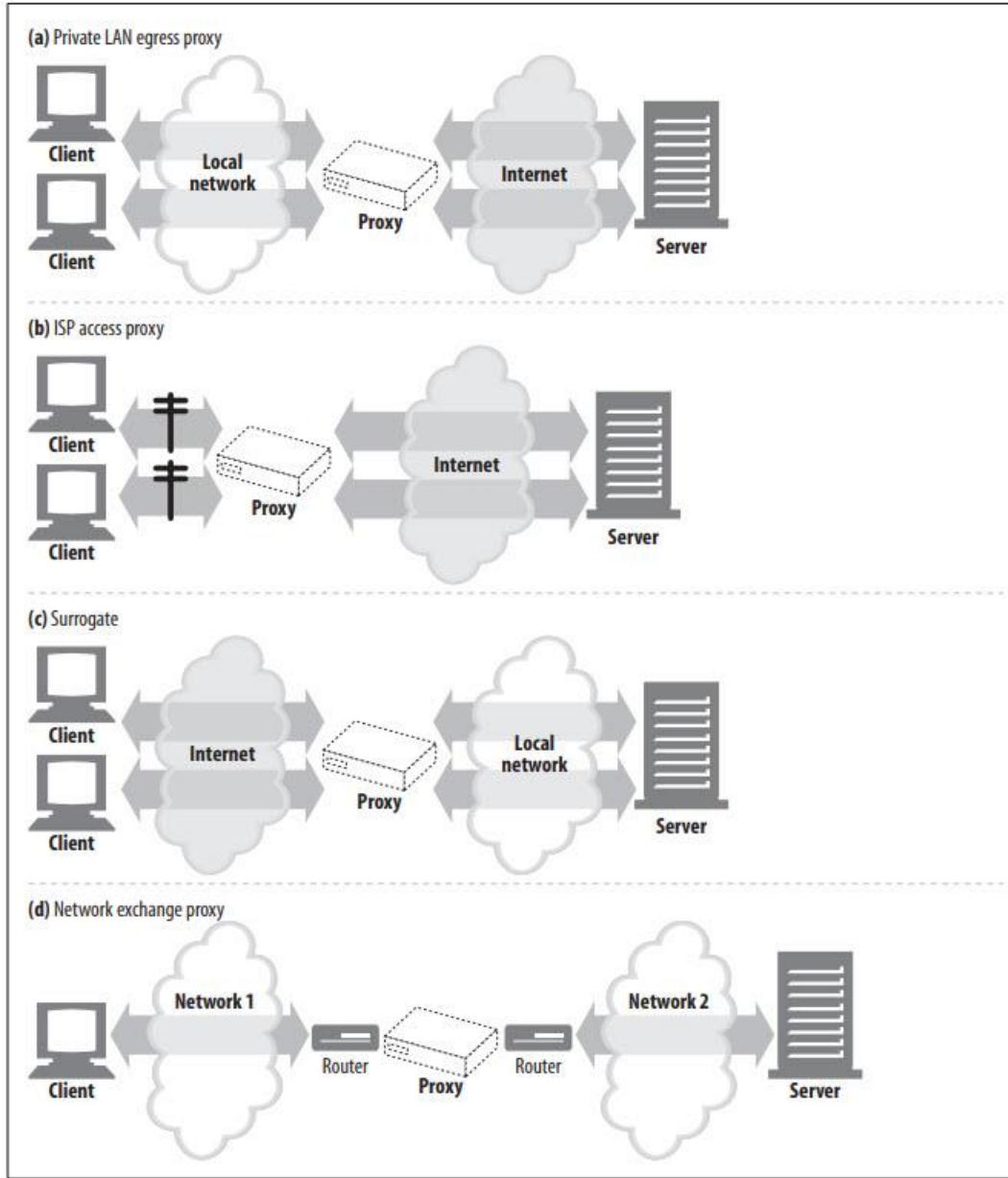
در بخش قبل توضیح داده شد که پروکسی‌ها چه کاری انجام می‌دهند. حالا بباید در مورد جایی که پروکسی‌ها در معماری شبکه مستقر می‌شوند صحبت کنیم. در این بخش به موارد زیر می‌پردازیم:

- چگونه می‌توان پروکسی‌ها را در شبکه‌ها مستقر کرد.
- چگونه پروکسی‌ها می‌توانند با هم به صورت سلسله مراتبی Chain شوند.
- چگونه ترافیک در وله اول به یک سرور پروکسی هدایت می‌شود.

Proxy Server Deployment

می‌توانید پروکسی‌ها را در انواع مکان‌ها، بسته به کاربرد مورد نظرشان، قرار دهید.





بخش a تصویر – Egress proxy

برای کنترل جریان ترافیک بین شبکه محلی و اینترنت، می‌توانید پروکسی‌ها را در نقاط خروجی شبکه‌های محلی قرار دهید. شما ممکن است از Egress proxy‌ها در یک شرکت به عنوان یک لایه محافظتی مشابه فایروال در برابر هکرهای مخرب خارج از شرکت یا کاهش هزینه‌های پهنای باند و بهبود عملکرد ترافیک اینترنت استفاده کنید. یک مدرسه ابتدایی ممکن است از یک Egress proxy فیلتر کننده برای جلوگیری از مرور محتوای نامناسب دانش آموزان استفاده کند.





– بخش b تصویر Access (ingress) proxy

پروکسی‌ها اغلب در نقاط دسترسی ISP قرار می‌گیرند و درخواست‌های کلی کلاینت‌ها را پردازش می‌کنند. ها از پروکسی‌های Cache برای ذخیره یک کپی از اسناد محبوب، برای بهبود سرعت دانلود برای کاربران خود (به ویژه آن‌هایی که اتصالات پرسرعت دارند) و کاهش هزینه‌های پهنای باند اینترنت استفاده می‌کنند.

– بخش c تصویر Surrogates

پروکسی‌ها اغلب به عنوان Reverse Proxy (که معمولاً به آن‌ها Surrogate نیز گفته می‌شود) در لبه شبکه، در مقابل سرورهای وب مستقر می‌شوند، جایی که می‌توانند تمام درخواست‌های ارسال شده به وب سرور را وارد کنند و فقط در صورت لزوم از سرور وب درخواست منابع کنند. جایگزین‌ها می‌توانند ویژگی‌های امنیتی را به سرورهای وب اضافه کنند یا با قرار دادن حافظه پنهان وب سرور سریع در مقابل سرورهای وب کنترل، عملکرد را بهبود بخشنند. Surrogate ها معمولاً نام و آدرس IP سرور وب را مستقیماً فرض می‌کنند، بنابراین همه درخواست‌ها به جای سرور به پروکسی می‌روند.

– بخش d تصویر Network exchange proxy

پروکسی‌ها را می‌توان در نقاط تبادل همتای اینترنت بین شبکه‌ها قرار داد تا ازدحام در اتصالات اینترنت از طریق Cache بکاهد و بر جریان ترافیک نظارت شود.

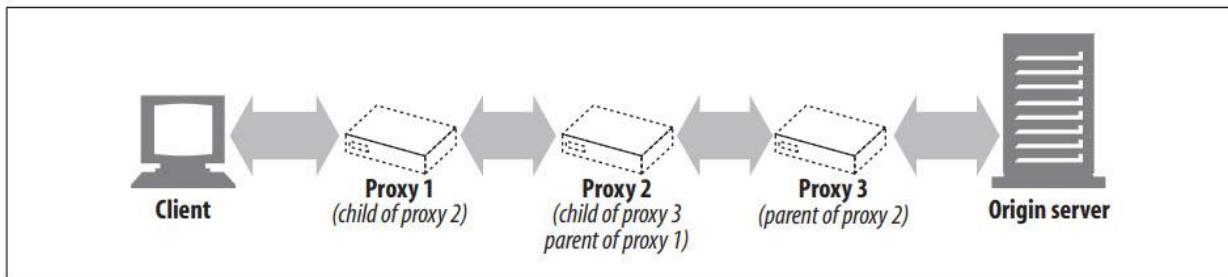
پروکسی‌های اصلی (Core Proxies) اغلب در جاهایی مستقر می‌شوند که پهنای باند اینترنت بسیار گران است (به ویژه در اروپا). برخی کشورها (مانند بریتانیا) نیز در حال ارزیابی استقرار پروکسی بحث برانگیز برای نظارت بر ترافیک اینترنت برای نگرانی‌های امنیت ملی هستند.

Proxy Hierarchies

پروکسی‌ها را می‌توان در زنجیره‌هایی به نام Proxy Hierarchies قرار داد. در یک Proxy Hierarchies پیام‌ها از پروکسی به پروکسی دیگر منتقل می‌شوند تا زمانی که در نهایت به سرور مبدأ برسند (و سپس از طریق پروکسی‌ها به کلاینت بازگردانده می‌شوند).

سرورهای پروکسی در یک Proxy Hierarchies دارای روابط والد و فرزندی هستند. پروکسی ورودی بعدی (نzdیکتر به سرور) والد و پروکسی خروجی بعدی (nzdیکتر به کلاینت) فرزند نامیده می‌شود. در شکل زیر، پروکسی ۱، پروکسی فرزند پروکسی ۲ است. به همین ترتیب، پروکسی ۲، پروکسی فرزند پروکسی ۳، و پروکسی ۳، پروکسی والد پروکسی ۲ است.





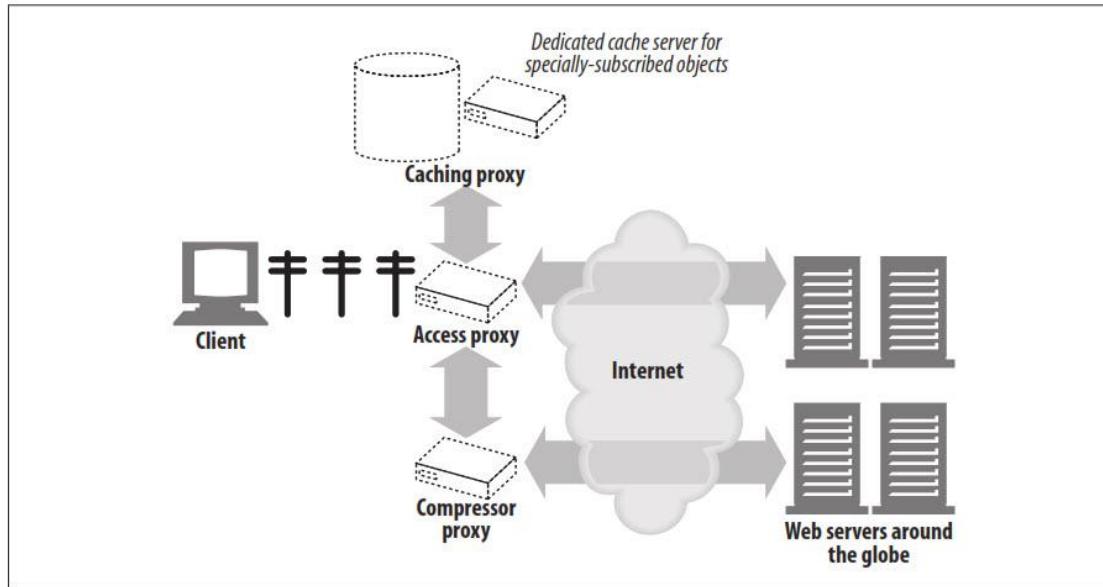
Proxy hierarchy content routing

سلسله مراتب پروکسی در شکل بالا ثابت است—پروکسی ۱ همیشه پیامها را به پروکسی ۲ و پروکسی ۲ همیشه پیامها را به پروکسی ۳ ارسال می‌کند. یک سرور پروکسی بر اساس عوامل بسیاری می‌تواند پیامها را به مجموعه متنوع و متغیری از سرورهای پروکسی و سرورهای مبدا ارسال کند.

به عنوان مثال، در شکل زیر، مسیرهای Access Proxy به پروکسی‌های والد یا سرورهای مبدا در شرایط مختلف: اگر شی درخواستی متعلق به سرور وب است که برای توزیع محتوا پول پرداخت کرده است، پروکسی می‌تواند درخواست را به یک سرور Cache مجاور هدایت کند که یا شی ذخیره شده را برمی‌گرداند یا اگر در دسترس نبود، آن را Fetch می‌کند.

اگر درخواست برای نوع خاصی از تصویر بود، Access Proxy ممکن است درخواست را به یک Compression Proxy اختصاصی هدایت کند که تصویر را Fetch می‌کند و سپس آن را فشرده می‌کند، بنابراین در یک مود کند، سریع‌تر برای کلاینت دانلود می‌شود.





در اینجا چند نمونه دیگر از انتخاب والد پویا آورده شده است:

Load balancing

یک پروکسی فرزند ممکن است یک پروکسی والد را بر اساس میزان فعلی بار کاری والدین انتخاب کند تا بار را به اطراف پراکنده نماید.

Geographic proximity routing

یک پروکسی فرزند ممکن است والدینی را که مسئول منطقه جغرافیایی سرور مبدا است انتخاب کند.

Protocol/type routing

یک پروکسی فرزند ممکن است مسیریابی را بر اساس URI به والدین و سرورهای مبدأ مختلف انجام دهد. انواع خاصی از URI ها ممکن است باعث شوند که درخواستها از طریق سرورهای پروکسی ویژه برای رسیدگی به پروتکل های خاص منتقل شوند.

Subscription-based routing

اگر ناشران برای خدمات با کارایی بالا پول بیشتری پرداخت کرده باشند، URI های آنها ممکن است برای بهبود عملکرد به Cache یا موتورهای فشرده سازی بزرگ هدایت شوند.

منطق مسیریابی والدین پویا در محصولات مختلف، از جمله فایل های پیکربندی، زبان های برنامه نویسی و افزونه های اجرایی پویا، به طور متفاوتی پیاده سازی می شود.





How Proxies Get Traffic

از آنجایی که کلاینت‌ها معمولاً مستقیماً با سرورهای وب صحبت می‌کنند، باید توضیح دهیم که چگونه ترافیک HTTP در وهله اول به یک پروکسی راه پیدا می‌کند. چهار راه متداول برای ایجاد ترافیک کلاینت به یک پروکسی وجود دارد:

Modify the client

بسیاری از کلاینت‌های وب، از جمله مرورگرهای Microsoft و Netscape، از پیکربندی دستی و خودکار پروکسی پشتیبانی می‌کنند. اگر یک کلاینت برای استفاده از یک سرور پروکسی پیکربندی شده باشد، کلاینت درخواست‌های HTTP را مستقیماً و عمداً به جای سرور مبدا به پروکسی ارسال می‌کند (بخش a شکل).

Modify the network

چندین تکنیک وجود دارد که در آن زیرساخت شبکه بدون اطلاع یا مشارکت کلاینت، ترافیک وب را به یک پروکسی هدایت می‌کند. این رهگیری (Interception) معمولاً به دستگاه‌های سوئیچینگ و مسیریابی متکی است که ترافیک HTTP را تماشا می‌کنند، آن را رهگیری می‌کنند و بدون اطلاع کلاینت، ترافیک را به یک پروکسی منتقل می‌کنند (بخش b شکل). این موضوع را Intercepting Proxy می‌گویند. پروکسی‌های رهگیری معمولاً «Transparent Proxies» نامیده می‌شوند، زیرا شما بدون اطلاع از حضورشان، به آن‌ها متصل می‌شوید. از آنجایی که اصطلاح «Transparency» قبلًا در مشخصات HTTP برای نشان دادن عملکردی‌ای که رفتار معنایی را تغییر نمی‌دهند استفاده می‌شود، جامعه استاندارد پیشنهاد می‌کند از عبارت «برای ضبط ترافیک استفاده شود».

Modify the DNS namespace

Surrogates، که سرورهای پروکسی هستند که در مقابل سرورهای وب قرار می‌گیرند، نام و آدرس IP سرور وب را مستقیماً فرض می‌کنند، بنابراین همه درخواست‌ها به جای سرور به آن‌ها می‌روند (بخش c شکل). این را می‌توان با ویرایش دستی جداول نامگذاری DNS یا با استفاده از سرورهای DNS پویا خاص که پروکسی یا سرور مناسب را برای استفاده بر حسب تقاضا محاسبه می‌کنند، ترتیب داد. در برخی از نصب‌ها، آدرس IP و نام سرور واقعی تغییر می‌کند و به Surrogate آدرس و نام قبلی داده می‌شود.

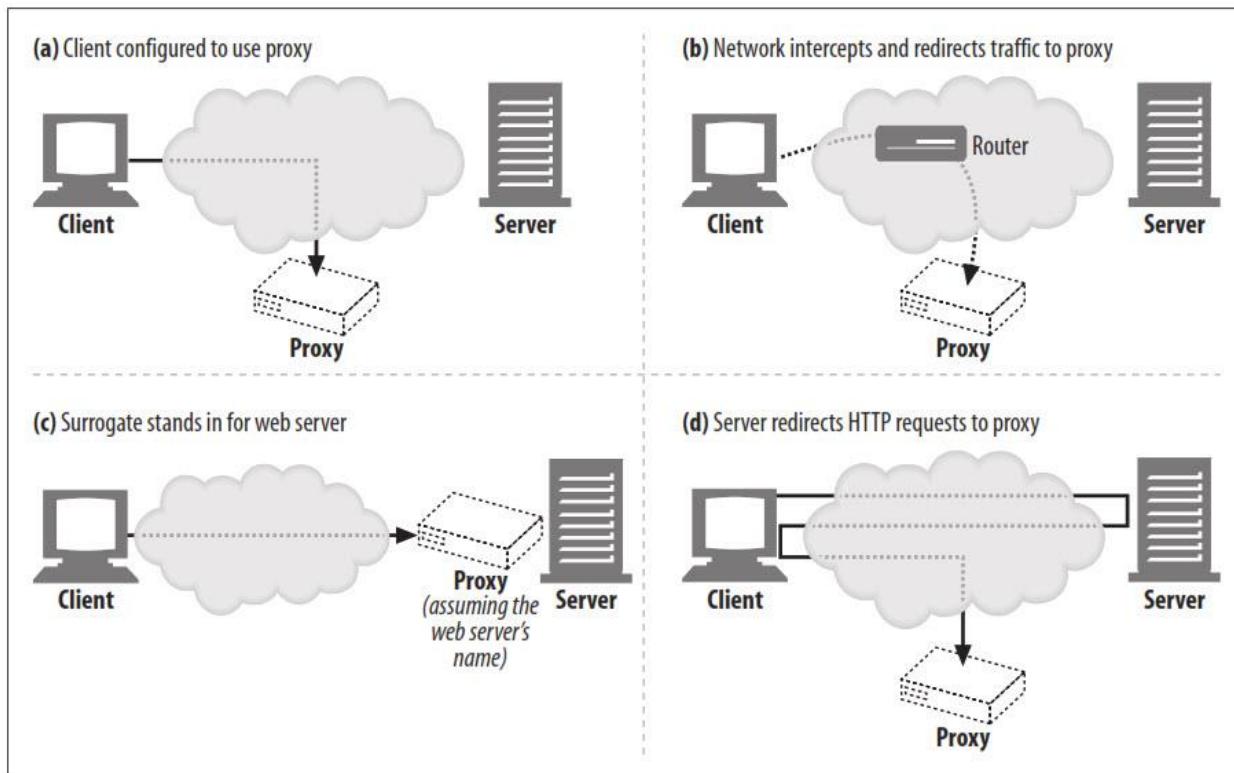




Modify the web server

برخی از وب سرورها نیز می‌توانند پیکربندی شوند تا درخواست‌های کلاینت را به یک پروکسی با ارسال یک فرمان تغییر مسیر HTTP (کد پاسخ ۳۰۵) به کلاینت هدایت کنند. پس از دریافت تغییر مسیر، کلاینت با پروکسی تراکنش می‌کند (بخش d شکل).

بخش بعدی نحوه پیکربندی کلاینت‌ها برای ارسال ترافیک به پروکسی‌ها را توضیح می‌دهد. در فصل ۲۰ نحوه پیکربندی شبکه، DNS و سرورها برای هدایت ترافیک به سرورهای پروکسی توضیح داده خواهد شد.



Client Proxy Settings

همه مرورگرهای وب مدرن به شما این امکان را می‌دهند تا پیکربندی لازم برای استفاده از پروکسی‌ها را انجام دهید. در واقع، بسیاری از مرورگرها راههای مختلفی را برای پیکربندی پروکسی‌ها ارائه می‌دهند، از جمله:

Manual configuration

شما به صراحت یک پروکسی برای استفاده تنظیم کرده اید.

Browser preconfiguration





فروشنده یا توزيع کننده مرورگر به صورت دستی تنظیمات پروکسی مرورگر (یا هر سرویس گیرنده وب دیگر) را قبل از تحويل آن به کلاینت، از قبل پیکربندی می‌کند.

Proxy auto-configuration (PAC)

شما یک URI به فایل پیکربندی خودکار پروکسی جاوا اسکریپت (PAC) ارائه می‌دهید. کلاینت فایل جاوا اسکریپت را Fetch می‌کند و آن را اجرا می‌نماید تا تصمیم بگیرد که آیا باید از پروکسی استفاده کند و اگر چنین است، از کدام سرور پروکسی استفاده شود.

WPAD proxy discovery

برخی از مرورگرها از پروتکل Web Proxy Autodiscovery Protocol (WPAD) پشتیبانی می‌کنند، که به طور خودکار یک "Configuration Server" را شناسایی می‌کند که مرورگر می‌تواند یک فایل پیکربندی خودکار را از آن دانلود کند.

Client Proxy Configuration: Manual

بسیاری از سرویس گیرندهای وب به شما اجازه می‌دهند تا پروکسی‌ها را به صورت دستی پیکربندی کنید. هر دو Microsoft Internet Explorer و Netscape Navigator پشتیبانی مناسبی برای پیکربندی پروکسی دارند.

Client Proxy Configuration: PAC Files

پیکربندی دستی پروکسی ساده اما غیر قابل انعطاف است. شما می‌توانید تنها یک سرور پروکسی را برای همه محتوا مشخص کنید و هیچ پشتیبانی از failover وجود ندارد. پیکربندی دستی پروکسی همچنین منجر به مشکلات مدیریتی برای سازمان‌های بزرگ می‌شود. با پایگاه بزرگی از مرورگرهای پیکربندی شده، پیکربندی مجدد هر مرورگر در صورت نیاز به ایجاد تغییرات دشوار یا غیرممکن است.

فایل‌های پیکربندی خودکار پروکسی (PAC) را حلی پویاتر برای پیکربندی پروکسی هستند، زیرا آن‌ها برنامه‌های کوچک جاوا اسکریپت هستند که تنظیمات پروکسی را در لحظه محاسبه می‌کنند. هر بار که به یک سند دسترسی پیدا می‌شود، یک تابع جاوا اسکریپت، سرور پروکسی مناسب را انتخاب می‌کند.

برای استفاده از فایل‌های PAC، مرورگر خود را با URI فایل PAC جاوا اسکریپت پیکربندی کنید (پیکربندی شبیه به پیکربندی دستی است، اما شما یک URI را در کادر «پیکربندی خودکار» ارائه می‌کنید). مرورگر فایل PAC را از این URI دریافت می‌کند و از منطق جاوا اسکریپت برای محاسبه سرور پراکسی مناسب





برای هر دسترسی استفاده می‌کند. فایل‌های PAC معمولاً دارای پسوند .pac و نوع «ns-proxy-autoconfig» هستند.

هر فایل PAC باید تابعی به نام FindProxyForURL(url,host) تعریف کند که سرور پروکسی مناسب را برای دسترسی به URI محاسبه می‌کند. مقدار برگشته تابع می‌تواند هر یک از مقادیر جدول زیر باشد.

FindProxyForURL return value	Description
DIRECT	Connections should be made directly, without any proxies.
PROXY host:port	The specified proxy should be used.
SOCKS host:port	The specified SOCKS server should be used.

فایل PAC در مثال زیر یک پروکسی را برای تراکنش‌های HTTP، یک پروکسی دیگر را برای تراکنش‌های FTP و اتصالات مستقیم را برای همه انواع دیگر تراکنش‌ها الزامی می‌کند.

```
function FindProxyForURL(url, host) {
  if (url.substring(0,5) == "http:") {
    return "PROXY http-proxy.mydomain.com:8080";
  } else if (url.substring(0,4) == "ftp:") {
    return "PROXY ftp-proxy.mydomain.com:8080";
  } else {
    return "DIRECT";
  }
}
```

Client Proxy Configuration: WPAD

mekanizm دیگر برای پیکربندی مرورگر، پروتکل کشف خودکار پروکسی وب (WPAD) است. WPAD الگوریتمی است که از یک استراتژی افزایشی مکانیسم‌های کشف برای یافتن خودکار فایل PAC مناسب برای مرورگر استفاده می‌کند. کلاینتی که پروتکل WPAD را پیاده سازی می‌کند:

- از WPAD برای پیدا کردن PAC URI استفاده می‌کند.
- فایل PAC را با توجه به URI واکشی می‌کند.
- فایل PAC را برای تعیین سرور پروکسی اجرا می‌کند.
- برای درخواست‌ها از سرور پروکسی استفاده می‌کند.

WPAD از یک سری تکنیک‌های کشف منبع برای تعیین فایل PAC مناسب استفاده می‌کند. تکنیک‌های کشف چندگانه در این بخش استفاده می‌شود، زیرا همه سازمان‌های نمی‌توانند از همه تکنیک‌ها استفاده کنند. WPAD هر تکنیک را یک به یک امتحان می‌کند تا زمانی که موفق شود.





مشخصات WPAD فعلی تکنیک‌های زیر را به ترتیب تعریف می‌کند:

- Dynamic Host Discovery Protocol (DHCP)
- Service Location Protocol (SLP)
- DNS well-known hostnames
- DNS SRV records
- DNS service URIs in TXT records

Tricky Things About Proxy Requests

این بخش برخی از جنبه‌های پیچیده و سوء تفاهم شده درخواست‌های سرور پروکسی را توضیح می‌دهد، از جمله:

- تفاوت URI ها در درخواست‌های پروکسی با درخواست‌های سرور
- چگونه پروکسی‌های رهگیری (Reverse Intercepting) و معکوس (Intercepting) می‌توانند اطلاعات میزبان سرور را مبهم کنند؟
- قوانین اصلاح URI
- چگونه پروکسی‌ها بر تکمیل خودکار URI هوشمندانه مرورگر یا ویژگی‌های گسترش نام میزبان تأثیر می‌گذارند؟

Proxy URIs Differ from Server URIs

وب سرور و پیام‌های پروکسی وب دارای Syntax یکسان هستند، با یک استثنای URI در پیام درخواست HTTP زمانی متفاوت است که یک کلاینت به جای پروکسی، درخواست را به سرور ارسال کند.

هنگامی که یک سرویس گیرنده درخواستی را به یک وب سرور ارسال می‌کند، خط درخواست تنها حاوی یک URI جزئی (بدون طرح، میزبان یا پورت) است، همانطور که در مثال زیر نشان داده شده است:

GET /index.html HTTP/1.0

User-Agent: SuperBrowserv1.3

با این حال، هنگامی که یک مشتری درخواستی را به یک پروکسی ارسال می‌کند، خط درخواست شامل کامل است. مثلاً:

GET http://www.marys-antiques.com/index.html HTTP/1.0

User-Agent: SuperBrowser v1.3





چرا دو فرمت درخواست متفاوت داریم، یکی برای پروکسی‌ها و دیگری برای سرورها؟ در طراحی اصلی HTTP، کلاینت‌ها مستقیماً با یک سرور واحد صحبت می‌کردند. میزبانی مجازی وجود نداشت و هیچ پیش‌بینی‌ای برای پروکسی‌ها در نظر گرفته نشده بود. از آنجایی که یک سرور واحد نام میزبان و پورت خود را می‌داند، برای جلوگیری از ارسال اطلاعات اضافی، کلاینت‌ها فقط URI جزئی را بدون طرح و میزبان (و پورت) ارسال می‌کنند.

هنگامی که پروکسی‌ها ظاهر شدن، URI‌های جزئی به یک مشکل تبدیل شدن. پروکسی‌ها باید نام سرور مقصد را بدانند تا بتوانند اتصالات خود را با سرور برقرار کنند و Gateway‌های مبتنی بر پروکسی برای اتصال به منابع FTP و سایر طرح‌ها به طرح URI نیاز داشتند.

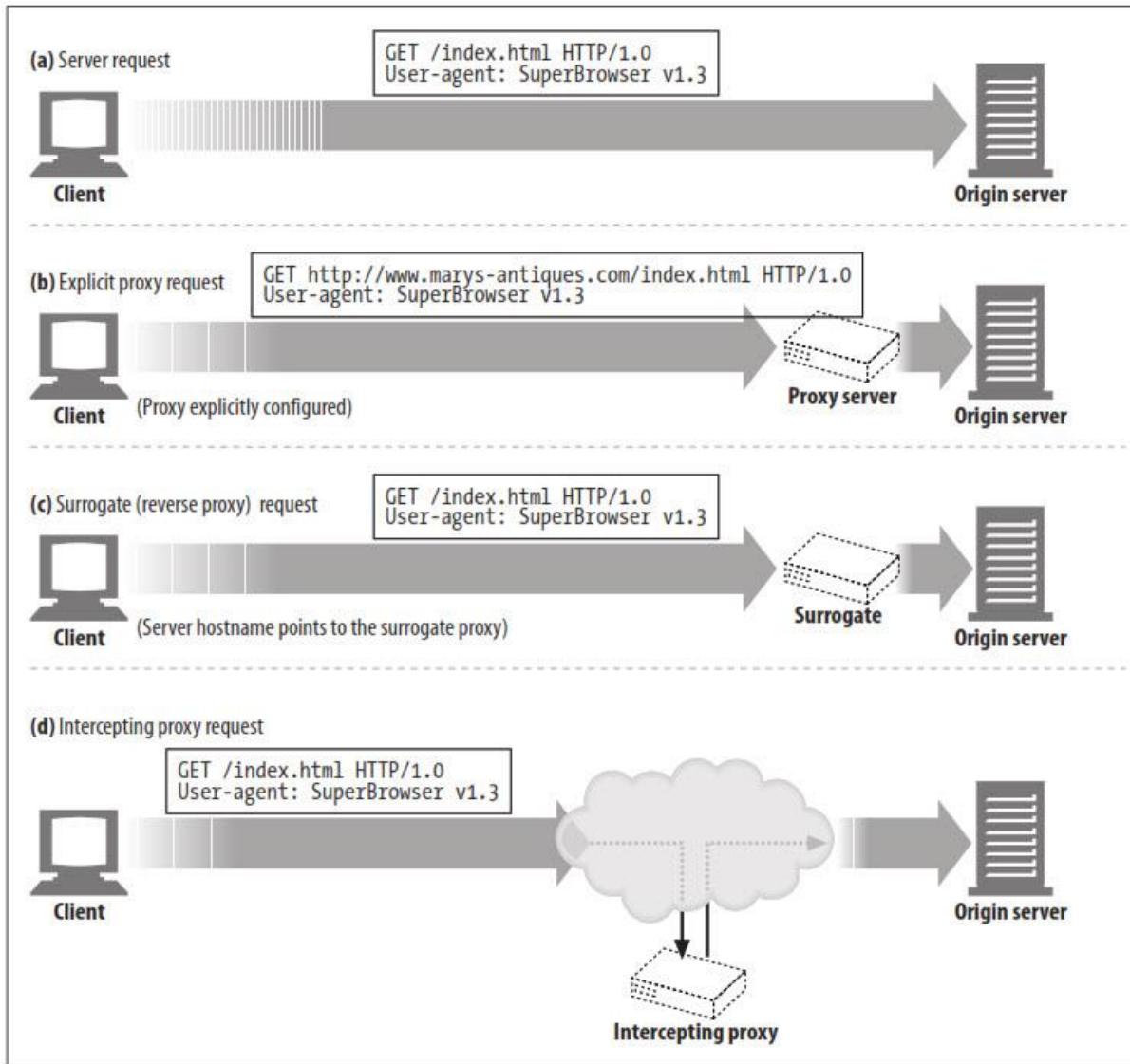
HTTP/1.0 با نیاز به URI کامل برای درخواست‌های پروکسی، مشکل را حل کرد، اما URI‌های جزئی را برای درخواست‌های سرور حفظ کرد (سرورهای زیادی از قبل مستقر شده بودند تا همه آن‌ها را تغییر دهند تا از URI‌های کامل پشتیبانی کنند).

HTTP/1.1 در حال حاضر به سرورها نیاز دارد که URI‌های کامل را برای درخواست‌های پروکسی و سرور مدیریت کنند، اما در عمل، بسیاری از سرورهای مستقر هنوز فقط URI‌های جزئی را می‌پذیرند.

بنابراین باید URI‌های جزئی را به سرورها و URI‌های کامل را به پروکسی‌ها ارسال کنیم. در مورد تنظیمات پروکسی کلاینت به طور صریح پیکربندی شده، کلاینت می‌داند که چه نوع درخواستی را صادر کند:

- هنگامی که کلاینت برای استفاده از پروکسی تنظیم نشده است، URI جزئی را ارسال می‌کند (بخش a از شکل زیر).
- هنگامی که کلاینت برای استفاده از پروکسی تنظیم می‌شود، URI کامل را ارسال می‌کند (بخش b از شکل زیر).





The Same Problem with Virtual Hosting

مشکل پرسنلی «از دست رفته طرح/میزبان/پورت» همان مشکلی است که سرورهای وب میزبان مجازی با آن مواجه هستند. وب سرورهای مجازی میزبان وب سرور فیزیکی یکسانی را در بین بسیاری از وب سایتها به اشتراک می‌گذارند. وقتی درخواستی برای URI جزئی index.html وارد می‌شود، وب سرور مجازی باید نام میزبان وبسایت مورد نظر را بداند.

با وجود مشابه بودن مشکلات، آنها به روش‌های مختلفی حل شدند:

- **Explicit Proxy**: مشکل را با نیاز به یک URI کامل در پیام درخواست حل می‌کنند.





- وب سرورهای میزبان مجازی (Virtual Hosted Web Server) برای حمل اطلاعات میزبان و پورت به هدر Host نیاز دارند.

Intercepting Proxies Get Partial URIs

تا زمانی که کلاینت‌ها HTTP را به درستی پیاده‌سازی کنند، URI‌های کامل را در درخواست‌ها به پروکسی‌هایی که به صورت Explicit پیکربندی شده‌اند ارسال می‌کنند. این بخشی از مشکل را حل می‌کند، اما یک نکته وجود دارد:

یک کلاینت همیشه نمی‌داند که با یک پروکسی صحبت می‌کند، زیرا ممکن است برخی از پروکسی‌ها برای کلاینت نامرئی باشند. حتی اگر کلاینت برای استفاده از پروکسی پیکربندی نشده باشد، ترافیک کلاینت همچنان ممکن است از طریق یک پروکسی Intercepting Surrogate یا انجام شود. در هر دوی این موارد، کلاینت فکر می‌کند که با یک وب سرور صحبت می‌کند و URI کامل را ارسال نمی‌کند:

Surrogate، همانطور که قبلاً توضیح داده شد، یک سرور پروکسی است که معمولاً با فرض نام میزبان یا آدرس IP آن، جای سرور مبدا را می‌گیرد. این مدل پروکسی، درخواست وب سرور را دریافت می‌کند و ممکن است پاسخ‌های کش شده یا درخواست‌های پروکسی را به سرور واقعی ارائه دهد. یک کلاینت نمی‌تواند یک جانشین را از یک وب سرور تشخیص دهد، بنابراین URI‌های جزئی را ارسال می‌کند (بخش c شکل پیشین).

یک پروکسی Intercepting یک سرور پروکسی در جریان شبکه است که ترافیک را از کلاینت به سرور ربوده و یا یک پاسخ کش را ارائه می‌دهد یا آن را پروکسی می‌کند. از آنجایی که پروکسی Intercepting ترافیک کلاینت به سرور را ربوده، URI‌های جزئی را دریافت می‌کند که به سرورهای وب ارسال می‌شود (بخش d شکل پیشین).

Proxies Can Handle Both Proxy and Server Requests

به دلیل راههای مختلفی که می‌توان ترافیک را به سرورهای پروکسی هدایت کرد، سرورهای پروکسی همه منظوره باید از URI‌های کامل و URI‌های جزئی در پیام‌های درخواستی پشتیبانی کنند. اگر یک درخواست پروکسی Explicit است، پروکسی باید از URI کامل استفاده کند یا اگر درخواست سرور وب است از URI جزئی و هدر میزبان مجازی استفاده نماید.

قوانين استفاده از URI‌های کامل و جزئی عبارتند از:

- اگر یک URI کامل ارائه شده باشد، پروکسی باید از آن استفاده کند.





- اگر یک URI جزئی ارائه شده باشد و یک هدر میزبان وجود داشته باشد، باید از هدر میزبان برای تعیین نام سرور مبدا و شماره پورت استفاده شود.
- اگر یک URI جزئی ارائه شده باشد و هدر میزبان وجود نداشته باشد، سرور مبدا باید به روش دیگری تعیین شود:
 - اگر پروکسی Surrogate باشد و برای یک سرور مبدأ قرار گرفته باشد، پروکسی را می‌توان با آدرس سرور واقعی و شماره پورت پیکربندی کرد.
 - اگر ترافیک Intercepting شده باشد و آدرس IP و پورت اصلی را در دسترس قرار دهد، پروکسی می‌تواند از آدرس IP و شماره پورت فناوری interception استفاده کند.
 - اگر همه چیز شکست بخورد، پروکسی اطلاعات کافی برای تعیین سرور مبدا ندارد و باید یک پیغام خطا برگرداند (اغلب به کاربر پیشنهاد می‌کند از یک مرورگر مدرن که از هدرهای میزبان پشتیبانی می‌کند استفاده نماید).

In-Flight URI Modification

سرورهای پروکسی باید در مورد تغییر URI درخواست هنگام ارسال پیامها بسیار مراقب باشند. تغییرات جزئی در URI، حتی اگر خوش خیم به نظر برسند، ممکن است مشکلات قابلیت همکاری با سرورهای پایین دستی را ایجاد کنند.

به طور خاص، برخی از پروکسی‌ها که با نام canonicalize شناخته می‌شوند URI‌ها را قبل از ارسال به پرش بعدی به شکلی استاندارد متعارف می‌کنند. تغییرات ظاهراً خوش خیم، مانند جایگزینی پورت‌های HTTP پیش‌فرض با یک «`:80`» صریح، یا تصحیح URI‌ها با جایگزینی کاراکترهای رزو شده غیرقانونی با جایگزین‌هایی که به درستی Escape شده‌اند، می‌توانند باعث ایجاد مشکلاتی در عملکرد شوند.

به طور کلی، سرورهای پروکسی باید تلاش کنند تا حد امکان Tolerant داشته باشند. آن‌ها نباید "پلیس پروتکل" باشند که به دنبال اجرای دقیق انطباق با پروتکل هستند، زیرا این امر می‌تواند شامل اختلال قابل توجهی در خدمات قبلی باشد.

به طور خاص، مشخصات HTTP، Interception Proxy های عمومی را از بازنویسی قسمت‌های مسیر مطلق URI‌ها هنگام ارسال آن‌ها منع می‌کند. تنها استثنای این است که آن‌ها می‌توانند یک مسیر خالی را با "/" جایگزین کنند.





URI Client Auto-Expansion and Hostname Resolution

مروعگرها، بسته به وجود یا نبودن پروکسی، URI های درخواست را به طور متفاوتی Resolve می کنند. بدون پروکسی، مروعگر URI شما را می گیرد و سعی می کند آدرس IP مربوطه را پیدا کند. اگر نام میزبان پیدا شود، مروعگر آدرس های IP مربوطه را امتحان می کند تا زمانی که اتصال موفقیت آمیز انجام شود.

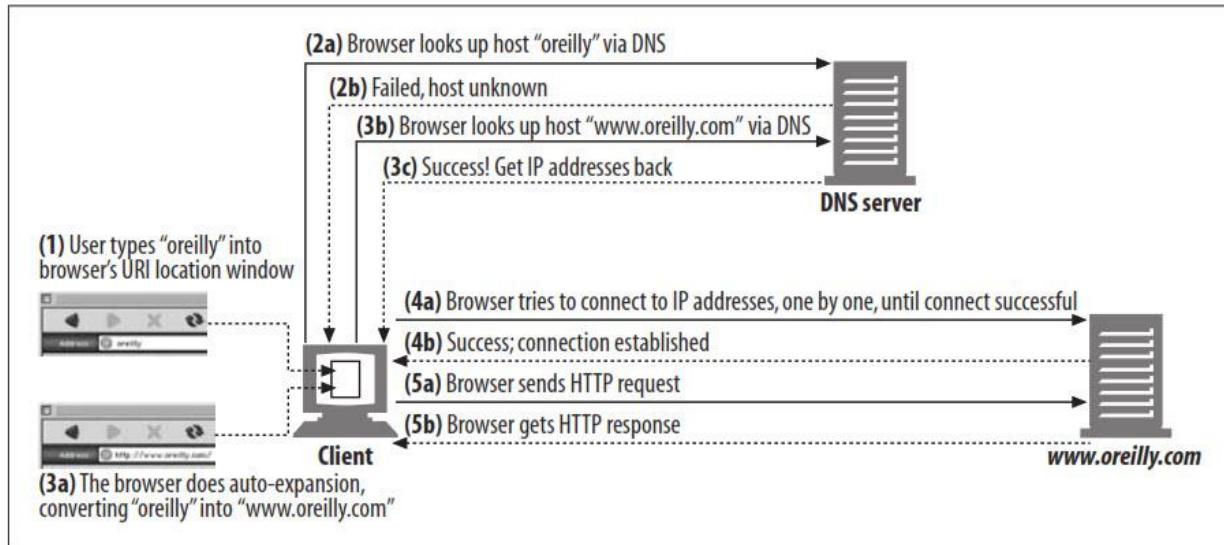
اما اگر میزبان پیدا نشد، بسیاری از مروعگرها سعی می کنند تا مقداری Expansion خودکار نامهای میزبان را ارائه کنند، در صورتی که مخفف میزبان را تایپ کرده باشید (به عنوان مثال، اجازه می دهد افراد به جای کنید):

- بسیاری از مروعگرها سعی می کنند یک "www" اضافه کنند. پیشوند و پسوند «.com»، در صورتی که فقط قسمت میانی نام یک وبسایت رایج را وارد کرده باشید (به عنوان مثال، اجازه می دهد افراد به جای yahoo را وارد کنند).
- برخی از مروعگرها حتی URI غیرقابل Resolve شما را به یک سایت شخص ثالث ارسال می کنند، که سعی می کند اشتباهات املایی را تصحیح کند و URI هایی را که ممکن است مد نظر شما باشد پیشنهاد می کند.
- علاوه بر این، پیکربندی DNS در اکثر سیستم ها به شما امکان می دهد فقط پیشوند نام میزبان را وارد کنید و DNS به طور خودکار، دامنه را جستجو می کند. اگر در دامنه "oreilly.com" هستید و نام میزبان "host7" را تایپ کنید، DNS به طور خودکار سعی می کند با "host7.oreilly.com" مطابقت داشته باشد. این یک نام میزبان کامل و معتبر نیست.

URI Resolution Without a Proxy

شکل زیر نمونه ای از گسترش خودکار نام میزبان مروعگر بدون پروکسی را نشان می دهد. در مراحل 2a تا 3c، مروعگر تغییرات نام میزبان را جستجو می کند تا زمانی که یک نام میزبان معتبر پیدا شود.





این چیزی است که در این شکل اتفاق می‌افتد:

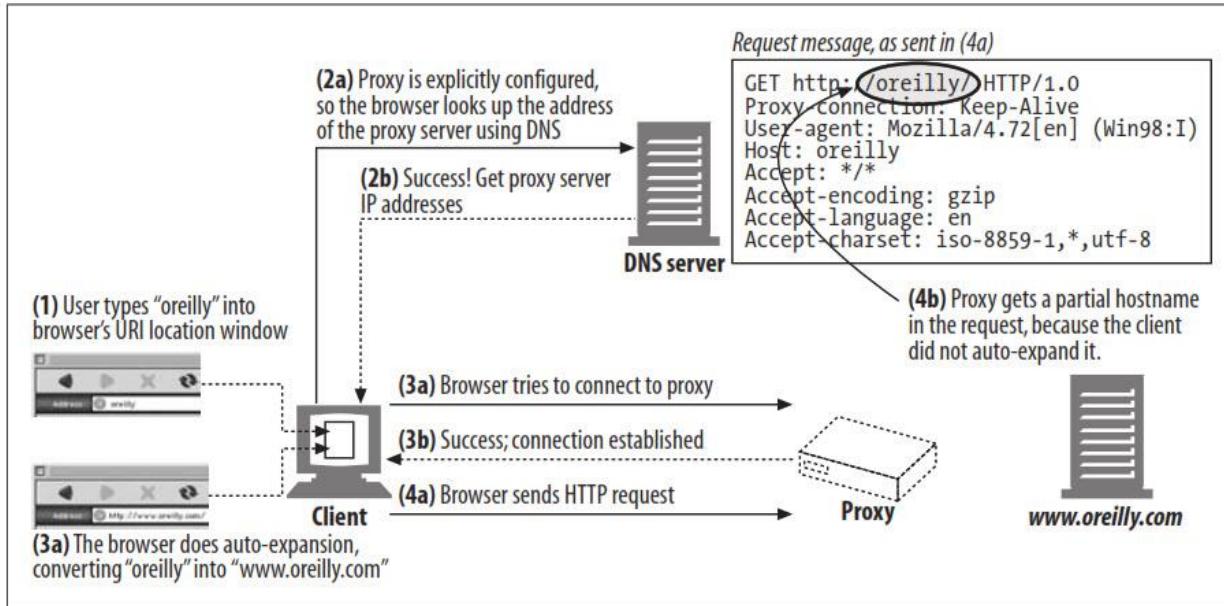
- در مرحله ۱، کاربر "oreilly" را در پنجره URI مورگر تایپ می‌کند. مورگر از "oreilly" به عنوان نام میزبان استفاده می‌کند و یک طرح پیش فرض "http://:http"، یک پورت پیش فرض "۸۰" و یک مسیر پیش فرض "/" را برای آن در نظر می‌گیرد.
- در مرحله ۲a، مورگر میزبان "oreilly" را جستجو می‌کند. این مورد شکست می‌خورد.
- در مرحله ۳a، مورگر به طور خودکار نام میزبان را گسترش می‌دهد و از DNS می‌خواهد که در مرحله ۳a، مورگر به طور خودکار نام میزبان را گسترش می‌دهد و از DNS می‌خواهد که "www.oreilly.com" را Resolve کند. این بخش موفقیت آمیز است.
- سپس مورگر با موفقیت به www.oreilly.com متصل می‌شود.

URI Resolution with an Explicit Proxy

وقتی از یک **Explicit Proxy** استفاده می‌کنید، مورگر دیگر هیچ یک از این بسطهای (Expansions) راحت را انجام نمی‌دهد، زیرا URI کاربر مستقیماً به پروکسی ارسال می‌شود.

همانطور که در شکل زیر نشان داده شده است، وقتی یک **Explicit Proxy** وجود دارد، مورگر نام میزبان جزئی را به طور خودکار گسترش نمی‌دهد. در نتیجه، زمانی که کاربر «oreilly» را در پنجره مکان مورگر تایپ می‌کند، پروکسی «http://oreilly» را ارسال می‌کند (مورگر طرح و مسیر پیش فرض را اضافه می‌کند اما نام میزبان را همانطور که وارد کردہاید باقی می‌گذارد).



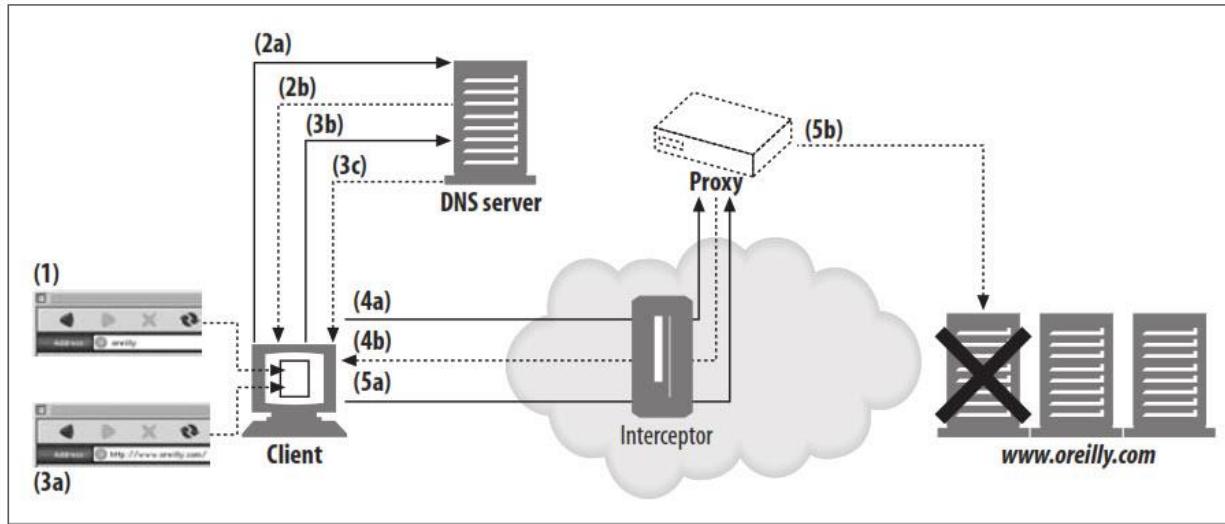


به همین دلیل، برخی از پروکسی‌ها تلاش می‌کنند تا حد امکان از خدمات راحتی مرورگر، از جمله گسترش خودکار «WWW...COM» و افزودن پسوندهای دامنه محلی، تقلید کنند.

URI Resolution with an Intercepting Proxy

Intercepting Proxy با یک Hostname Resolution مربوط می‌شود، هیچ پروکسی وجود ندارد! این رفتار تقریباً مانند مورد سرور پیش می‌رود و مرورگر به طور خودکار نام میزبان را تا موفقیت DNS گسترش می‌دهد. اما همانطور که شکل زیر نشان می‌دهد، زمانی که اتصال به سرور برقرار می‌شود، تفاوت قابل توجهی رخ می‌دهد.





شکل بالا تراکنش زیر را نشان می‌دهد:

در مرحله ۱، کاربر "oreilly" را در پنجره URI Location مرورگر تایپ می‌کند.

در مرحله ۲a، مرورگر، میزبان "oreilly" از طریق DNS جستجو می‌کند، اما سرور DNS از کار می‌افتد و پاسخ می‌دهد که میزبان ناشناخته است، همانطور که در مرحله 2b نشان داده شده است.

در مرحله 3a، مرورگر توسعه خودکار را انجام می‌دهد و "www.oreilly.com" را به ".www.oreilly.com" تبدیل می‌کند. در مرحله 3b، مرورگر، میزبان "www.oreilly.com" را از طریق DNS جستجو می‌کند. این بار، همانطور که در مرحله 3c نشان داده شده است، سرور DNS موفق است و آدرس‌های IP را به مرورگر برمی‌گرداند.

در مرحله 4a، کلاینت قبلًا نام میزبان را با موفقیت Resolve کرده است و لیستی از آدرس‌های IP را دارد. به طور معمول، کلاینت سعی می‌کند به هر آدرس IP متصل شود تا زمانی که موفق شود، زیرا ممکن است برخی از آدرس‌های IP از بین رفته باشند. اما با یک Intercepting Proxy، اولین تلاش برای اتصال توسط سرور پروکسی خاتمه می‌یابد، نه سرور مبدا. در این حالت کلاینت معتقد است که با موفقیت با سرور وب صحبت می‌کند، اما وب سرور حتی ممکن است Alive نباشد.

هنگامی که پروکسی در نهایت آماده تعامل با سرور اصلی واقعی است (مرحله 5b)، پروکسی ممکن است متوجه شود که آدرس IP در واقع به یک سرور پایین اشاره دارد. برای ارائه همان سطح تحمل خطای ارائه شده توسط مرورگر، پروکسی باید آدرس‌های IP دیگر را امتحان کند، این کار یا با Resolve مجدد نام میزبان در هدر میزبان انجام شده و یا با انجام یک جستجوی معکوس DNS در آدرس IP صورت می‌پذیرد. مهم است که هر دو اجرای Explicit Proxy و Intercepting Proxy از تحمل خطای DNS

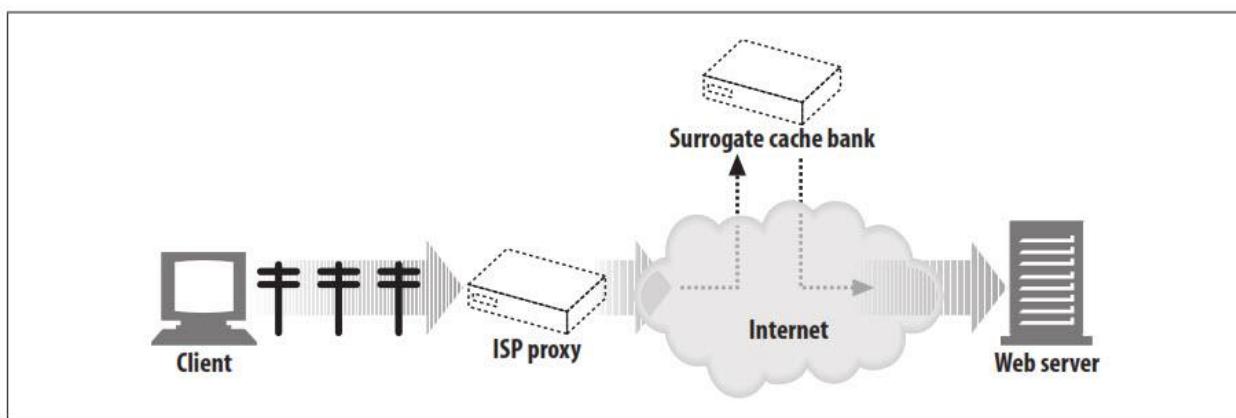




Explicit Proxy به سرورهای مرده پشتیبانی کنند، زیرا وقتی مرورگرها برای استفاده از یک Resolution پیکربندی می‌شوند، برای تحمل خطأ به پروکسی متکی هستند.

Tracing Messages

امروزه، غیرمعمول نیست که درخواست‌های وب از طریق زنجیره‌ای از دو یا چند پروکسی در مسیر خود از کلاینت به سرور عبور کنند (شکل زیر). به عنوان مثال بسیاری از شرکت‌ها از سرورهای پروکسی Cache برای دسترسی به اینترنت، برای صرفه جویی در امنیت و هزینه استفاده نموده و بسیاری از ISP‌های بزرگ از Proxy Cache برای بهبود عملکرد و پیاده سازی ویژگی‌ها استفاده می‌کنند. امروزه درصد قابل توجهی از درخواست‌های وب از طریق پروکسی‌ها انجام می‌شود. در همان زمان، به دلایل عملکردی، Replicate محتوا در بانک‌های Cache پراکنده در سراسر جهان به طور فزاینده‌ای محبوب می‌شود.



پروکسی‌ها توسط فروشنده‌گان مختلف توسعه می‌یابند. آن‌ها دارای ویژگی‌ها و اشکالات مختلفی هستند و توسط سازمان‌های مختلف مدیریت می‌شوند.

همانطور که پروکسی‌ها رایج‌تر می‌شوند، باید بتوانید جریان پیام‌ها را در میان پروکسی‌ها ردیابی کنید و هر گونه مشکلی را شناسایی کنید، همانطور که ردیابی جریان بسته‌های IP در سوئیچ‌ها و روترهای مختلف مهم است، این موضوع نیز اهمیت دارد.

The Via Header

فیلد هدر Via اطلاعات مربوط به هر گره میانی (پروکسی یا Gateway) را فهرست می‌کند که پیام از آن عبور می‌کند. هر بار که پیامی از گره دیگری عبور می‌کند، گره میانی باید به انتهای لیست Via اضافه شود.



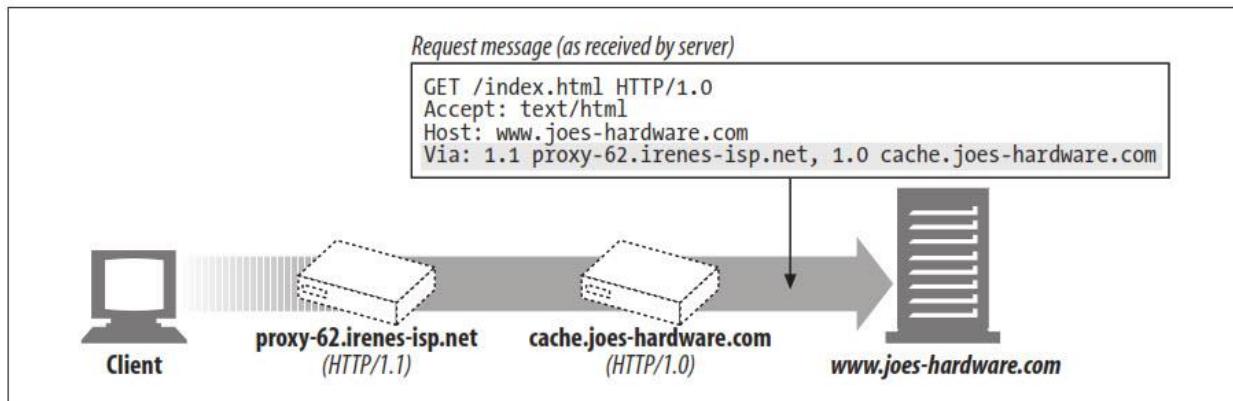


رشته Via زیر به ما می‌گوید که پیام از طریق دو پروکسی عبور کرده است. این نشان می‌دهد که پروکسی اول پروتکل 1.1 HTTP را اجرا کرده و proxy-62.irenes-isps.net نامیده می‌شود، و پروکسی دوم cache.joes-hardware.com را پیاده سازی کرده و cache.joes-hardware.com نامیده می‌شود:

Via: 1.1 proxy-62.irenes-isps.net, 1.0 cache.joes-hardware.com

فیلد Via header برای ردیابی ارسال پیام‌ها، تشخیص حلقه‌های پیام و شناسایی قابلیت‌های پروتکل همه فرستنده‌ها در طول زنجیره درخواست/پاسخ استفاده می‌شود.

پروکسی‌ها همچنین می‌توانند از هدرهای Via برای شناسایی حلقه‌های مسیریابی در شبکه استفاده کنند. یک پروکسی باید قبل از ارسال درخواست، یک رشته منحصر به فرد مرتبط با خودش را در سربرگ Via وارد کند و باید وجود این رشته را در درخواست‌های دریافتی برای شناسایی حلقه‌های مسیریابی در شبکه بررسی نماید.



Via syntax

فیلد هدر Via شامل لیستی از waypoints است که با کاما از هم جدا شده‌اند. هر waypoint نشان دهنده یک سرور پروکسی یا Gateway هاپ است و حاوی اطلاعاتی در مورد پروتکل و آدرس آن گره می‌بینی است. در اینجا یک مثال از یک هدر Via با دو نقطه بین آمده است:

Via = 1.1 cache.joes-hardware.com, 1.1 proxy.irenes-isps.net

رسمی هدر Via در اینجا نشان داده شده است:





```
Via          = "Via" ":" 1#( waypoint )
waypoint     = ( received-protocol received-by [ comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
received-by   = ( host [ ":" port ] ) | pseudonym
```

توجه داشته باشید که هر **Via** در **Waypoint** دارای حداکثر چهار جزء است: یک نام پروتکل اختیاری (به طور پیش فرض برای HTTP)، یک نسخه پروتکل مورد نیاز، یک نام گره مورد نیاز و یک نظر توصیفی اختیاری:

Protocol name

پروتکل دریافت شده توسط یک واسطه است. اگر پروتکل HTTP باشد، نام پروتکل اختیاری است. در غیر این صورت، نام پروتکل به نسخه اضافه شده و با یک "/" از هم جدا شده است. پروتکلهای غیر HTTP زمانی رخ می‌دهند که **Gateway** ها درخواست‌های HTTP را برای پروتکلهای دیگر (FTP، HTTPS و غیره) متصل می‌کنند.

Protocol version

نسخه پیام دریافت شده فرمت نسخه به پروتکل بستگی دارد. برای HTTP، از شماره نسخه استاندارد استفاده می‌شود ("1.0"، "1.1" و غیره). این نسخه در قسمت **Via** گنجانده شده است، بنابراین برنامه‌های بعدی قابلیت‌های پروتکل تمام واسطه‌های قبلی را خواهند دانست.

Node name

هاست و شماره پورت اختیاری واسطه (اگر پورت گنجانده نشده باشد، می‌توانید پورت پیش فرض پروتکل را در نظر بگیرید). در برخی موارد ممکن است یک سازمان به دلایل حفظ حریم خصوصی، مایل به ارائه نام میزبان واقعی نباشد، در این صورت ممکن است با یک نام مستعار جایگزین شود.

Node comment

یک **comment** اختیاری که گره واسطه را بیشتر توضیح می‌دهد. معمول است که اطلاعات فروشنده و نسخه را در اینجا درج کنید و برخی از سرورهای پروکسی نیز از فیلد نظر برای درج اطلاعات عیب‌یابی درباره رویدادهای که در آن دستگاه رخ داده است استفاده می‌کنند.

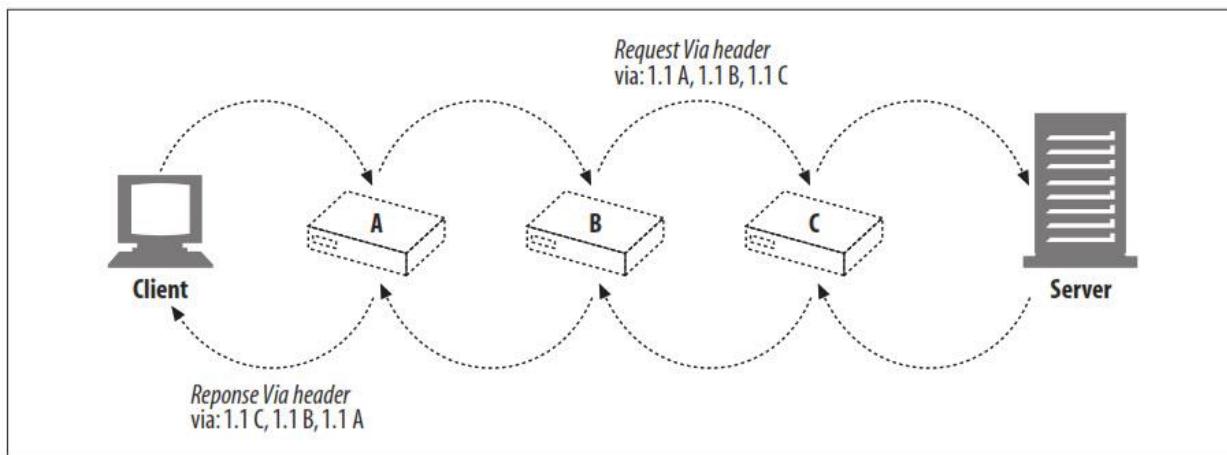
Via request and response paths

هر دو پیام درخواست و پاسخ از طریق پروکسی‌ها عبور می‌کنند، بنابراین هر دو پیام درخواست و پاسخ دارای هدر **Via** هستند.





از آنجایی که درخواست‌ها و پاسخ‌ها معمولاً از طریق یک اتصال TCP انجام می‌شوند، پیام‌های پاسخ در همان مسیر درخواست‌ها به عقب حرکت می‌کنند. اگر یک پیام درخواست از طریق پروکسی‌های A، B و C عبور کند، پیام پاسخ مربوطه از طریق پروکسی‌های C، B و سپس A حرکت می‌کند. بنابراین، هدر Via برای پاسخ‌ها تقریباً همیشه برعکس هدر Via برای پاسخ‌ها است (شکل زیر).



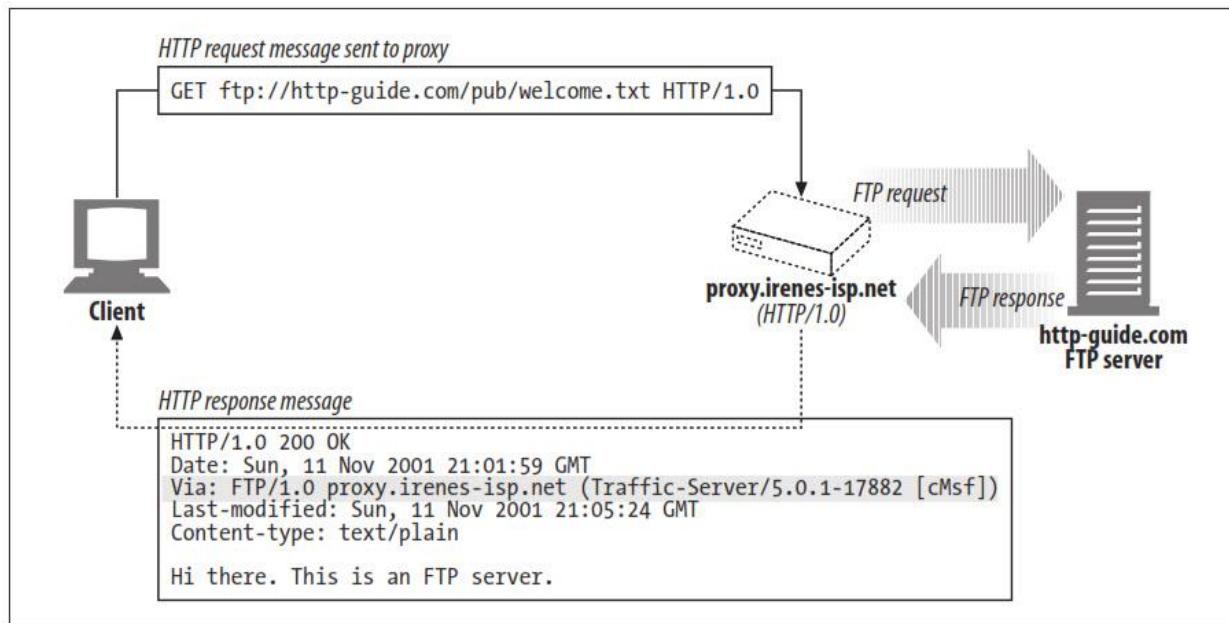
Via and gateways

برخی از پروکسی‌ها عملکرد Gateway را برای سرورهایی که با پروتکل‌های غیر HTTP صحبت می‌کنند ارائه می‌کنند. هدر Via این تبدیل‌های پروتکل را ثبت می‌کند، بنابراین برنامه‌های HTTP می‌توانند از قابلیت‌های پروتکل و تبدیل‌ها در طول زنجیره پروکسی آگاه باشند. شکل زیر یک سرویس گیرنده HTTP را نشان می‌دهد که یک URI FTP را از طریق یک دروازه HTTP/FTP درخواست می‌کند.

کلاینت یک درخواست HTTP برای `ftp://http-guide.com/pub/welcome.txt` به `gateway proxy.irenes-isps.net` ارسال می‌کند. پروکسی که به عنوان Gateway پروتکل عمل می‌کند، شی مورد نظر را با استفاده از پروتکل FTP از سرور FTP بازیابی می‌کند. سپس پروکسی با این فیلد هدر Via، شیء را در یک پاسخ HTTP به کلاینت می‌فرستد:

`Via: FTP/1.0 proxy.irenes-isps.net (Traffic-Server/5.0.1-17882 [cMs f])`





توجه کنید پروتکل دریافتی **Comment** اختیاری شامل نام تجاری و شماره نسخه سرور پروکسی و برخی اطلاعات تشخیصی فروشند است.

The Server and Via headers

فیلد هدر پاسخ سرور، نرم افزار مورد استفاده توسط سرور مبدا را توصیف می‌کند. در اینجا چند نمونه هستند:

Server: Apache/1.3.14 (Unix) PHP/4.0.4

Server: Netscape-Enterprise/4.1

Server: Microsoft-IIS/5.0

اگر یک پیام پاسخ از طریق یک پروکسی ارسال می‌شود، مطمئن شوید که پروکسی هدر سرور را تغییر نمی‌دهد. هدر **Server** برای سرور مبدا در نظر گرفته شده است. در عوض، پروکسی باید یک ورودی **Via** اضافه کند.

Privacy and security implications of Via

مواردی وجود دارد که می‌خواهیم نامهای میزبان دقیق را در رشته **Via** وجود نداشته باشد. به طور کلی، مگر اینکه این رفتار به طور صریح فعال باشد، زمانی که یک سرور پروکسی بخشی از فایروال شبکه است، نباید نام و پورت هاستها را در پشت فایروال ارسال کند، زیرا دانش معماری شبکه در پشت فایروال ممکن است برای یک طرف مخرب مفید باشد. (افراد مخرب می‌توانند از نام رایانه‌ها و شماره نسخه‌ها برای آشنایی با معماری شبکه در پشت محیط امنیتی استفاده کنند. این اطلاعات ممکن است در حملات امنیتی مفید باشد. علاوه بر این، نام رایانه‌ها ممکن است سرخهایی برای پروژه‌های خصوصی در یک سازمان باشد.)





اگر **Via node-name forwarding** فعال نباشد، پروکسی‌هایی که بخشی از محیط امنیتی هستند باید نام میزبان را با نام مستعار مناسب برای آن میزبان جایگزین کنند. با این حال، به طور کلی، پروکسی‌ها باید سعی کنند برای هر سرور پروکسی یک ورودی **Via** را حفظ کنند، حتی اگر نام واقعی مبهم باشد.

برای سازمان‌هایی که الزامات حفظ حریم خصوصی بسیار قوی برای پنهان کردن طراحی و توپولوژی معماری‌های شبکه داخلی دارند، یک پروکسی ممکن است دنباله‌ای مرتب از ورودی‌های **Via** (با مقادیر پروتکل دریافتی یکسان) را در یک ورودی واحد و متصل ترکیب کند. مثلا:

Via: 1.0 foo, 1.1 devirus.company.com, 1.1 access-logger.company.com

می‌تواند جمع شود به:

Via: 1.0 foo, 1.1 concealed-stuff

چندین ورودی را نباید با یکدیگر ترکیب نکنید مگر اینکه همه آن‌ها تحت کنترل سازمانی یکسانی باشند و میزبان‌ها قبلًا با نام مستعار جایگزین شده باشند. همچنین، ورودی‌هایی را که دارای مقادیر پروتکل دریافتی متفاوتی هستند، با یکدیگر ترکیب نکنید.

The TRACE Method

سرورهای پروکسی می‌توانند پیام‌ها را در حین ارسال پیام‌ها تغییر دهند. هدرها اضافه، اصلاح و حذف می‌شوند و بدندها را می‌توان به فرمتهای مختلف تبدیل کرد. همانطور که پروکسی‌ها پیچیده‌تر می‌شوند و فروشنده‌گان بیشتری محصولات پروکسی را مستقر می‌کنند، مشکلات قابلیت همکاری افزایش می‌یابد. برای تشخیص آسان شبکه‌های پروکسی، ما به روشی نیاز داریم تا به راحتی شاهد تغییر پیام‌ها در حین ارسال، از **hop by hop** طریق شبکه پروکسی **HTTP** باشیم.

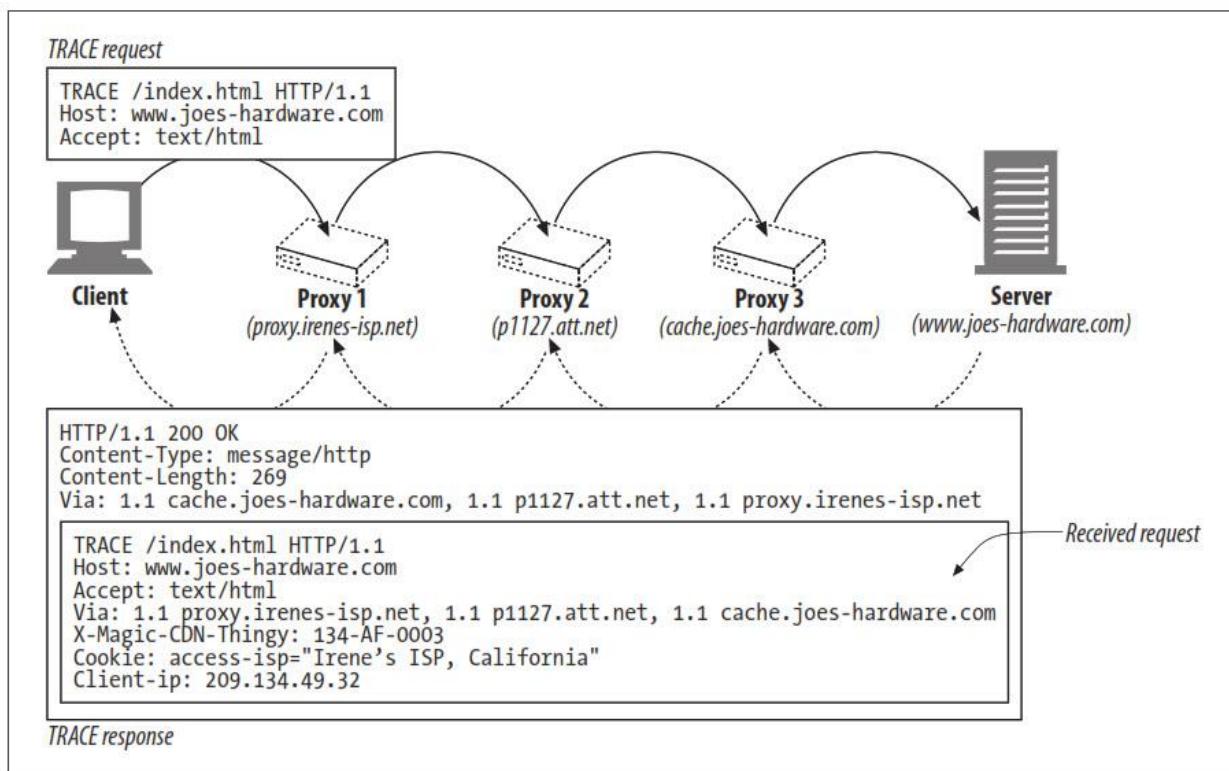
متدهای **TRACE** در **HTTP/1.1** به شما این امکان را می‌دهد که پیام درخواست را از طریق زنجیره‌ای از پروکسی‌ها ردیابی کنید. همچنین مشاهده کنید که پیام از چه پروکسی‌هایی عبور می‌کند و چگونه هر پروکسی پیام درخواست را تغییر می‌دهد. **TRACE** برای اشکال زدایی جریان‌های پروکسی بسیار مفید است.

هنگامی که درخواست **TRACE** به سرور مقصد می‌رسد، کل پیام درخواست به فرستنده منعکس می‌شود و در بدنده یک پاسخ **HTTP** جمع شده است. هنگامی که پاسخ **TRACE** می‌رسد، کلاینت می‌تواند دقیقاً پیام را که سرور دریافت کرده و لیست پروکسی‌هایی که از آن عبور کرده است (در هدر **Via**) بررسی کند. پاسخ **TRACE** دارای پیام با **Content-Type: message/http** و وضعیت **200 OK** است.



Max-Forwards

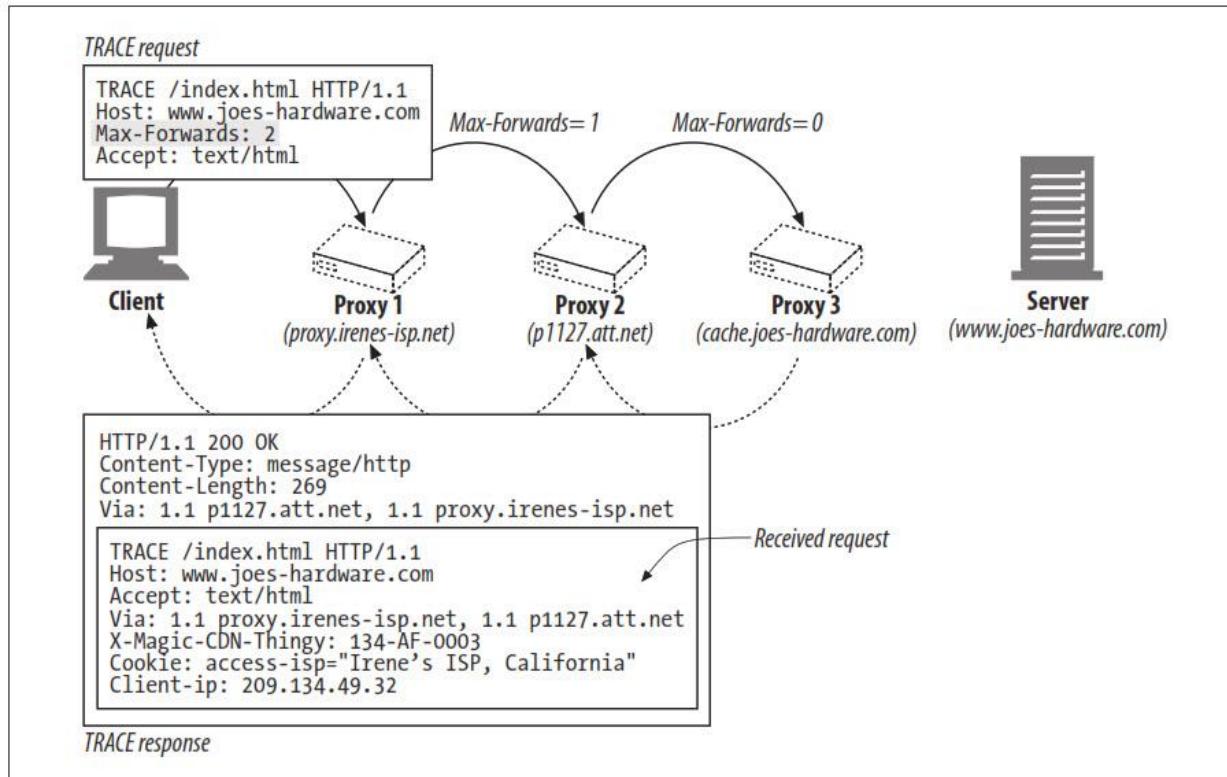
به طور معمول، پیام‌های TRACE بدون در نظر گرفتن تعداد پروکسی‌های مداخله‌گر، تمام مسیر را تا سرور مقصد طی می‌کنند. می‌توانید از هدر Max-Forwards برای محدود کردن تعداد پروکسی‌ها برای درخواست‌های OPTIONS و TRACE استفاده کنید، که برای آزمایش زنجیرهای از پروکسی‌ها که پیام‌ها را در یک حلقه نامحدود Max-Forwards ارسال می‌کنند یا برای بررسی اثرات سرورهای پراکسی خاص در وسط یک زنجیر مفید است. همچنان ارسال پیام‌های OPTIONS را محدود می‌کند.



فیلد هدر درخواست Max-Forwards حاوی یک عدد صحیح است که تعداد دفعات باقیمانده ارسال این پیام درخواست را نشان می‌دهد. اگر مقدار Max-Forwards صفر باشد (Max-Forwards: 0)، گیرنده باید پیام را بدون ارسال بیشتر به سمت کلاینت منعکس کند، حتی اگر گیرنده سرور اصلی نباشد.

اگر مقدار Max-Forwards دریافتی بزرگتر از صفر باشد، پیام ارسال شده باید حاوی یک قسمت Max-Forwards به روز شده باشد که مقدار آن یک کاهش یافته است. همه پروکسی‌ها و Gateway‌ها باید از Max-Forwards پشتیبانی کنند. می‌توانید از Max-Forwards برای مشاهده درخواست در هر hop در یک زنجیره پروکسی استفاده کنید.





Proxy Authentication

پروکسی‌ها می‌توانند به عنوان دستگاه‌های کنترل دسترسی عمل کنند. HTTP مکانیزمی به نام احراز هویت پروکسی را تعریف می‌کند که درخواست‌های محتوا را تا زمانی که کاربر اعتبار مجوز دسترسی معتبر را به پروکسی ارائه کند، مسدود می‌کند:

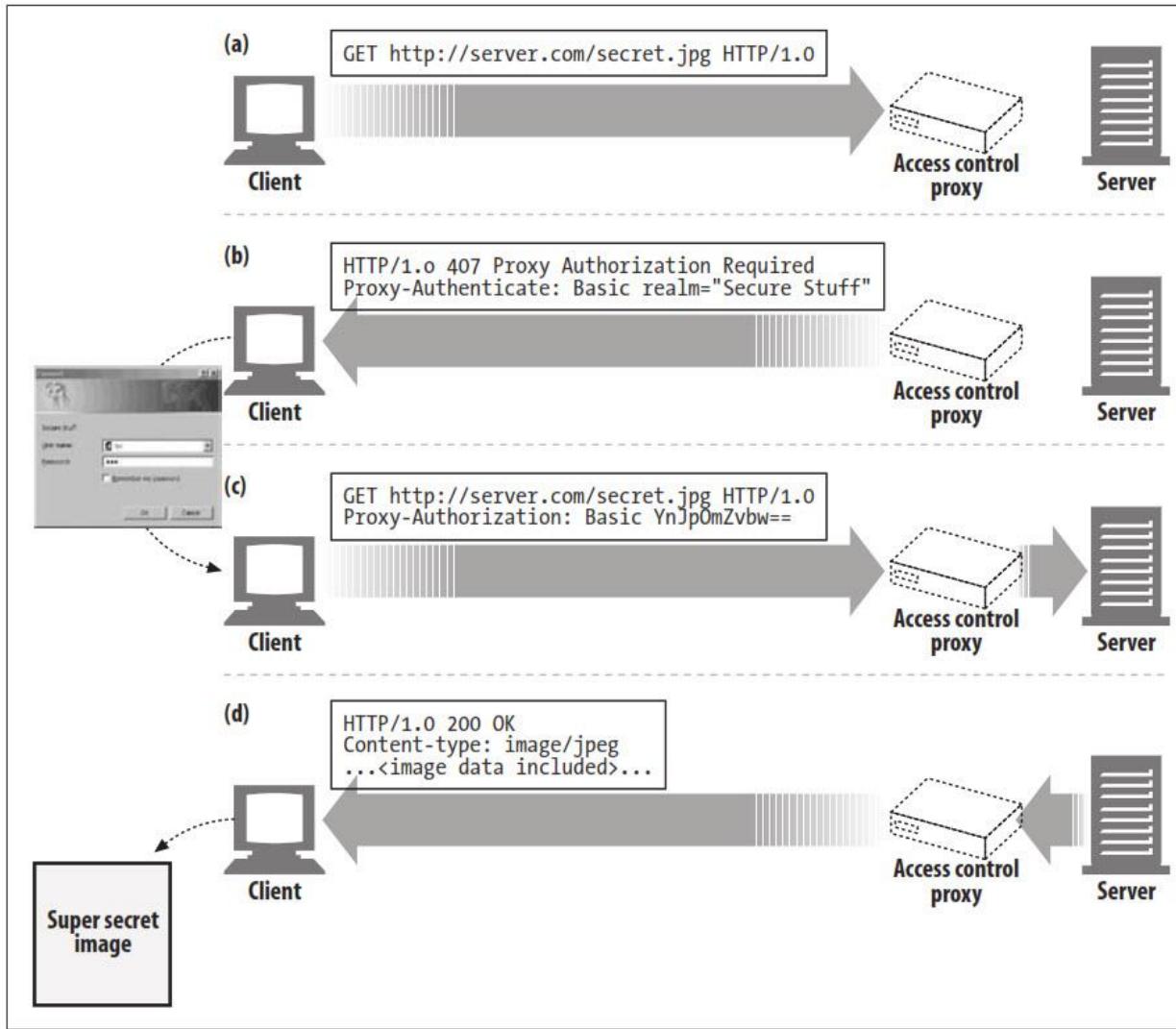
هنگامی که درخواستی برای محتوای محدود شده به یک سرور پروکسی می‌رسد، سرور پروکسی می‌تواند یک کد وضعیت **407 Proxy Authorization Required** را بازگرداند که خواستار مجوزهای دسترسی است، همراه با یک فیلد هدر **Proxy-Authenticate** که نحوه ارائه این اعتبارنامه‌ها را توضیح می‌دهد (بخش b شکل).

هنگامی که مشتری پاسخ **407** را دریافت می‌کند، سعی می‌کند اعتبار مورد نیاز را از یک پایگاه داده محلی یا درخواست از کاربر جمع آوری کند.

هنگامی که اعتبارنامه‌ها به دست آمد، کلاینت درخواست را دوباره ارسال می‌کند و اعتبار مورد نیاز را در فیلد هدر **Proxy-Authorization** ارائه می‌کند.

اگر اعتبارنامه‌ها معتبر باشند، پروکسی درخواست اصلی را در امتداد زنجیره ارسال می‌کند (بخش c شکل). در غیر این صورت یک پاسخ **407** دیگر ارسال می‌شود.





احراز هویت پروکسی معمولاً زمانی که چندین پروکسی در یک زنجیره وجود دارد که هر کدام در احراز هویت شرکت می‌کنند، به خوبی کار نمی‌کند. افراد پیشرفتهایی را برای HTTP پیشنهاد کرده‌اند تا اعتبارنامه‌های احراز هویت را با **waypoint** های خاص در یک زنجیره پروکسی مرتبط کنند، اما این پیشرفت‌ها به‌طور گسترده پیاده‌سازی نشده‌اند.

در فصل ۱۲ توضیح دقیق مکانیسم‌های احراز هویت HTTP، آورده می‌شود.

Proxy Interoperation

کلاینت‌ها، سرورها و پروکسی‌ها توسط چندین فروشنده، با نسخه‌های مختلف مشخصات HTTP ساخته می‌شوند. آن‌ها از ویژگی‌های مختلف پشتیبانی می‌کنند و باگ‌های مختلفی دارند. سرورهای پروکسی





باید بین دستگاههای سمت سرویس گیرنده و سمت سرور واسطه شوند، که ممکن است پروتکل @های مختلفی را پیاده سازی کنند و خصلت‌های دردرساز داشته باشند.

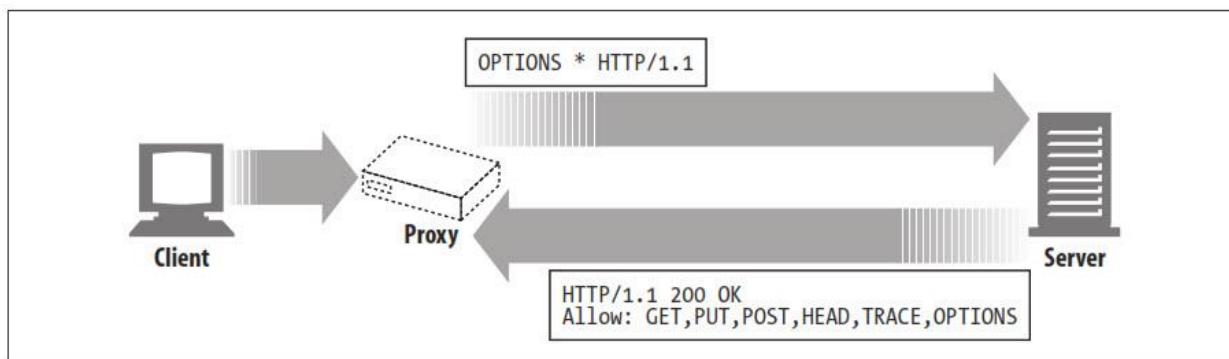
Handling Unsupported Headers and Methods

سرور پروکسی ممکن است تمام فیلدهای هدر را که از آن عبور می‌کنند درک نکند. برخی از هدرها ممکن است جدیدتر از خود پروکسی باشند. برخی دیگر ممکن است فیلدهای هدر سفارشی شده برای یک برنامه خاص باشد. پروکسی‌ها باید فیلدهای هدر ناشناخته را ارسال نموده و باید ترتیب نسبی فیلدهای هدر را با همان نام حفظ کنند.

پروکسی‌هایی که نمی‌توانند متدهای پشتیبانی نشده را تونل کنند، ممکن است امروزه در اکثر شبکه‌ها قابل اجرا نباشند، زیرا دسترسی Hotmail از طریق Microsoft Outlook استفاده گسترده‌ای از روش‌های پسوند HTTP می‌کند.

OPTIONS: Discovering Optional Feature Support

متدهای OPTIONS به یک کلاینت (یا پروکسی) این امکان را می‌دهد تا عملکرد پشتیبانی شده (به عنوان مثال، متد پشتیبانی شده) یک وب سرور یا یک منبع خاص در یک وب سرور را شناسایی کند. کلاینت‌ها می‌توانند از OPTIONS برای تعیین قابلیت‌های سرور قبل از تعامل با سرور استفاده کنند و کار با پروکسی‌ها و سرورهای سطوح مختلف ویژگی را آسان‌تر کند.



اگر URI درخواست OPTIONS ستاره (*) باشد، این درخواست به کل عملکرد پشتیبانی شده سرور مربوط می‌شود. مثلا:

OPTIONS * HTTP/1.1

اگر URI یک منبع واقعی باشد، درخواست OPTIONS در مورد ویژگی‌های موجود برای آن منبع خاص سوال می‌کند:





OPTIONS http://www.joes-hardware.com/index.html HTTP/1.1

در صورت موفقیت، متدهای OPTIONS را بر می‌گرداند که شامل فیلدهای هدر مختلف است که ویژگی‌های اختیاری را که در سرور پشتیبانی می‌شوند یا در دسترس منبع هستند، توصیف می‌کنند. تنها فیلد هدری که HTTP/1.1 در پاسخ مشخص می‌کند، هدر Allow است، که توصیف می‌کند چه روش‌هایی توسط سرور (یا منبع خاصی در سرور) پشتیبانی می‌شوند. (همه منابع از هر روشی پشتیبانی نمی‌کنند. به عنوان مثال، یک پرس و جو اسکریپت CGI ممکن است از یک فایل PUT پشتیبانی نکند، و یک فایل HTML ایستا روش POST را نمی‌پذیرد.)

The Allow Header

فیلد Allow entity header مجموعه‌ای از متدها را فهرست می‌کند که توسط منبع شناسایی شده توسط درخواست یا کل سرور پشتیبانی می‌شود اگر URI درخواست * باشد. مثلا:

Allow: GET, HEAD, PUT

هدر Allow را می‌توان به عنوان سرفصل درخواست برای توصیه روش‌هایی که باید توسط منبع جدید پشتیبانی شوند استفاده کرد. سرور نیازی به پشتیبانی از این متدها ندارد و باید یک عنوان Allow را در پاسخ منطبق گنجانده و متدهای واقعی پشتیبانی شده را فهرست کند.

یک پروکسی نمی‌تواند فیلد Allow header را تغییر دهد، حتی اگر تمام متدهای مشخص شده را درک نکند، زیرا ممکن است کلاینت مسیرهای دیگری برای صحبت با سرور اصلی داشته باشد.





For More Information

<http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

RFC 2616, “Hypertext Transfer Protocol,” by R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Mastinter, P. Leach, and T. Berners-Lee.

<http://search.ietf.org/rfc/rfc3040.txt>

RFC 3040, “Internet Web Replication and Caching Taxonomy.”

Web Proxy Servers

Ari Luotonen, Prentice Hall Computer Books.

<http://search.ietf.org/rfc/rfc3143.txt>

RFC 3143, “Known HTTP Proxy/Caching Problems.”

Web Caching

Duane Wessels, O'Reilly & Associates, Inc.





Caching - فصل هفتم

Cache های وب دستگاه های HTTP هستند که به طور خودکار کپی اسناد محبوب را نگه می دارند. هنگامی که یک درخواست وب به یک Cache می رسد، اگر یک نسخه Cache شده محلی در دسترس باشد، سند به جای سرور اصلی، از ذخیره سازی محلی ارائه می شود. Cache ها مزایای زیر را دارند:

- انتقال داده های اضافی را کاهش داده و منجر به صرفه جویی در هزینه های شبکه می شود.
- منجر به کاهش bottleneck شبکه شده و صفحات بدون پهنای باند بیشتر، سریعتر بازگذاری می شوند.
- کاهش تقاضا برای سرورهای مبدا را فراهم می کند و سرورها سریعتر پاسخ می دهند و از اضافه بار جلوگیری می کنند.
- تاخیرهای مسافتی را کاهش می دهد، زیرا صفحاتی که دور هستند، کندر Load می شوند.

در این فصل، ما توضیح می دهیم که چگونه Cache ها عملکرد را بهبود می بخشنند و هزینه را کاهش می دهند، چگونه اثربخشی آن ها را اندازه گیری می کنیم و برای به حداقل رساندن تأثیر، آن ها را کجا قرار می دهیم. همچنین توضیح می دهیم که چگونه HTTP کپی های Cache شده را تازه نگه می دارد و چگونه Cache ها با سرورهای وب می شوند.

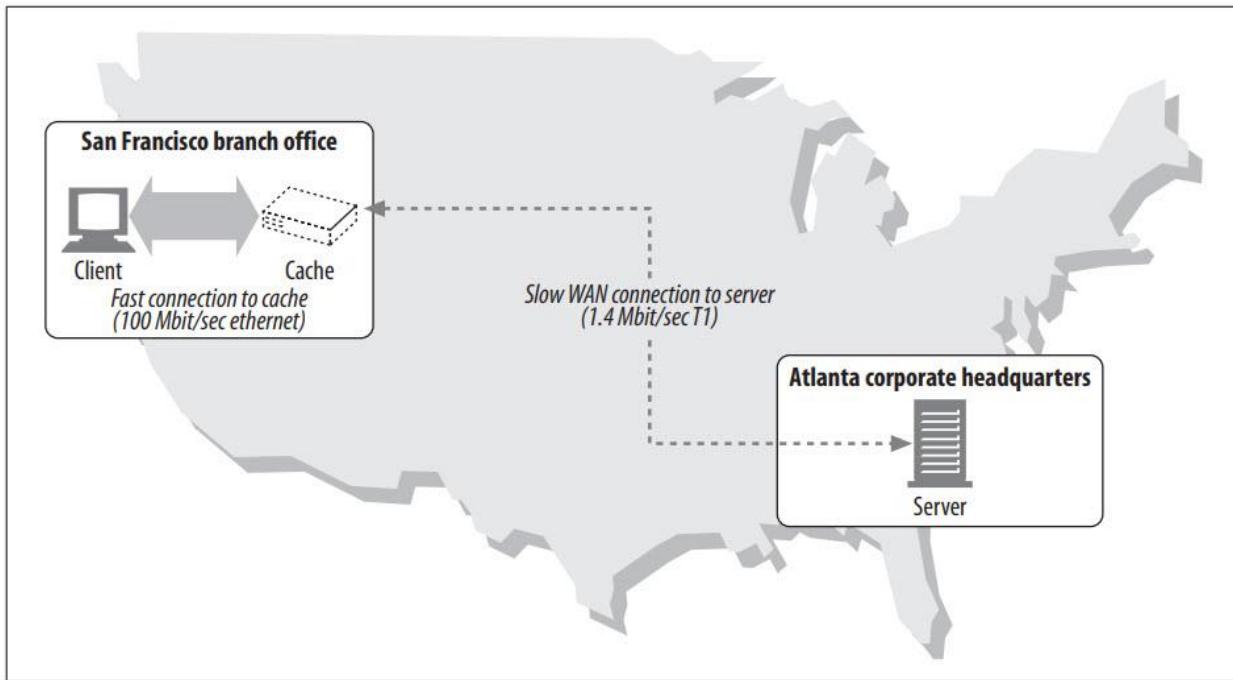
Redundant Data Transfers

هنگامی که چندین کلاینت به یک صفحه سرور مبدا محبوب دسترسی پیدا می کنند، سرور یک سند را چندین بار و یک بار به هر کلاینت ارسال می کند. همان بایت ها بارها و بارها در سراسر شبکه حرکت می کنند. این انتقال داده های اضافی، پهنای باند گران شبکه را مصرف می کنند، سرعت انتقال را کاهش می دهند و منجر به را overload سرورهای وب می شوند. Cache ها یک کپی از اولین پاسخ سرور را نگه می دارند. درخواست های بعدی را می توان از نسخه ذخیره شده انجام داد، و ترافیک تکراری و بیهوده به و از سرورهای اصلی را کاهش داد.

Bandwidth Bottlenecks

Cache ها همچنین می توانند Bottleneck های شبکه را کاهش دهند. بسیاری از شبکه ها پهنای باند بیشتری را به کلاینت های شبکه محلی نسبت به سرورهای راه دور ارائه می کنند (شکل زیر). کلاینت ها با سرعت پایین ترین شبکه در راه به سرورها دسترسی دارند. اگر یک کلاینت یک کپی از Cache در یک LAN سریع دریافت کند، ذخیره Cache می تواند عملکرد را افزایش دهد - به خصوص برای اسناد بزرگتر.





در شکل بالا، ممکن است ۳۰ ثانیه طول بکشد تا کاربر در شعبه سانفرانسیسکو Joe's Hardware, Inc یک فایل موجودی ۵ مگابایتی را از دفتر مرکزی آتلانتا از طریق اتصال اینترنت T1 با سرعت ۱.۴ مگابایت در ثانیه دانلود کند. اگر سند در دفتر سانفرانسیسکو ذخیره شده باشد، یک کاربر محلی ممکن است بتواند همان سند را در کمتر از یک ثانیه از طریق اتصال ارتباط دریافت کند.

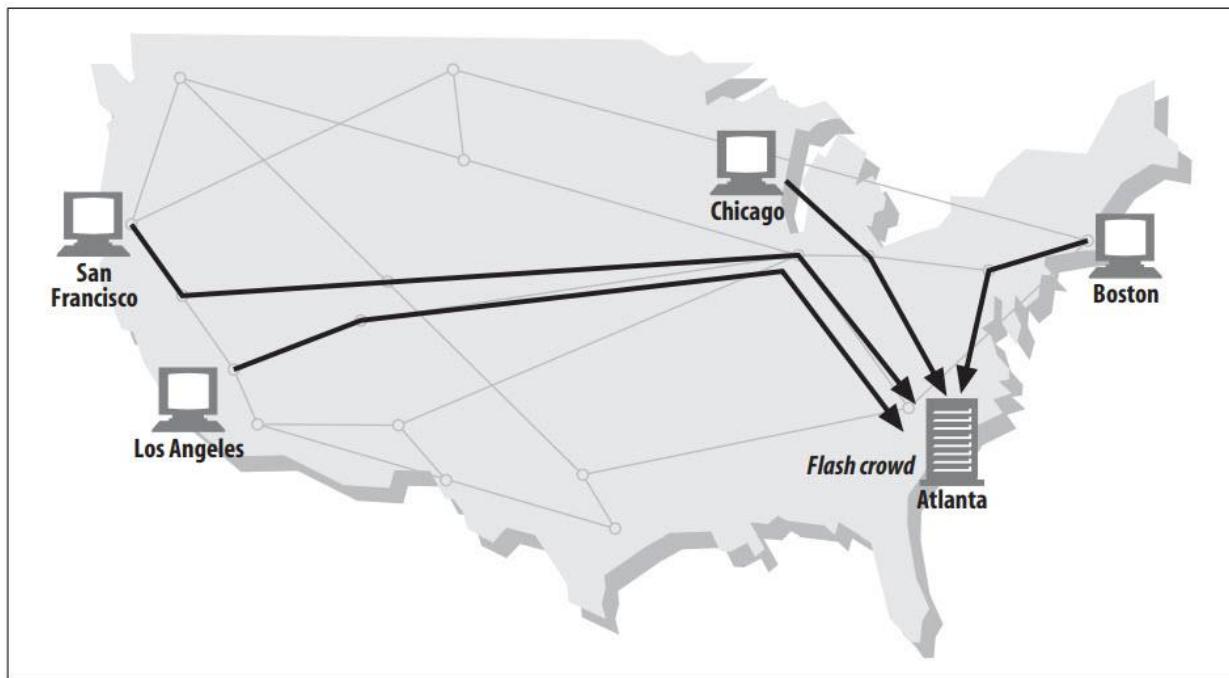
جدول زیر نشان می‌دهد که چگونه پهنای باند بر زمان انتقال برای چند سرعت شبکه مختلف و چند اندازه مختلف اسناد تأثیر می‌گذارد. پهنای باند باعث تأخیر قابل توجهی برای اسناد بزرگتر می‌شود و تفاوت سرعت بین انواع مختلف شبکه بسیار چشمگیر است. یک مودم ۵۶ کیلوبیت بر ثانیه برای انتقال یک فایل ۵ مگابایتی که می‌تواند در کمتر از یک ثانیه در شبکه LAN منتقل شود، ۷۴۹ ثانیه (بیش از ۱۲ دقیقه) طول می‌کشد.



	Large HTML (15 KB)	JPEG (40 KB)	Large JPEG (150 KB)	Large file (5 MB)
Dialup modem (56 Kbit/sec)	2.19	5.85	21.94	748.98
DSL (256 Kbit/sec)	.48	1.28	4.80	163.84
T1 (1.4 Mbit/sec)	.09	.23	.85	29.13
Slow Ethernet (10 Mbit/sec)	.01	.03	.12	4.19
DS3 (45 Mbit/sec)	.00	.01	.03	.93
Fast Ethernet (100 Mbit/sec)	.00	.00	.01	.42

Flash Crowds

به ویژه برای از بین بردن Flash Crowds زمانی رخ می‌دهد که یک رویداد ناگهانی (مانند اخبار فوری، اعلامیه ایمیل انبوه، یا یک رویداد مشهور) باعث شود بسیاری از افراد تقریباً همزمان به یک سند وب دسترسی پیدا کنند. افزایش ترافیک اضافی ناشی از آن می‌تواند باعث فروپاشی فاجعه بار شبکه‌ها و سرورهای وب شود.



هنگامی که "Starr Report" که جزئیات تحقیقات Kenneth Starr از رئیس جمهور ایالات متحده کلینتون را شرح می‌دهد در ۱۱ سپتامبر ۱۹۹۸ در اینترنت منتشر شد، وب سرورهای مجلس نمایندگان ایالات متحده بیش از ۳ میلیون درخواست در ساعت دریافت کردند که ۵۰ برابر میانگین Load سرور بود. یک وب سایت خبری، CNN.com، به طور متوسط بیش از ۵۰۰۰۰ درخواست در هر ثانیه به سرورهای خود گزارش داد.



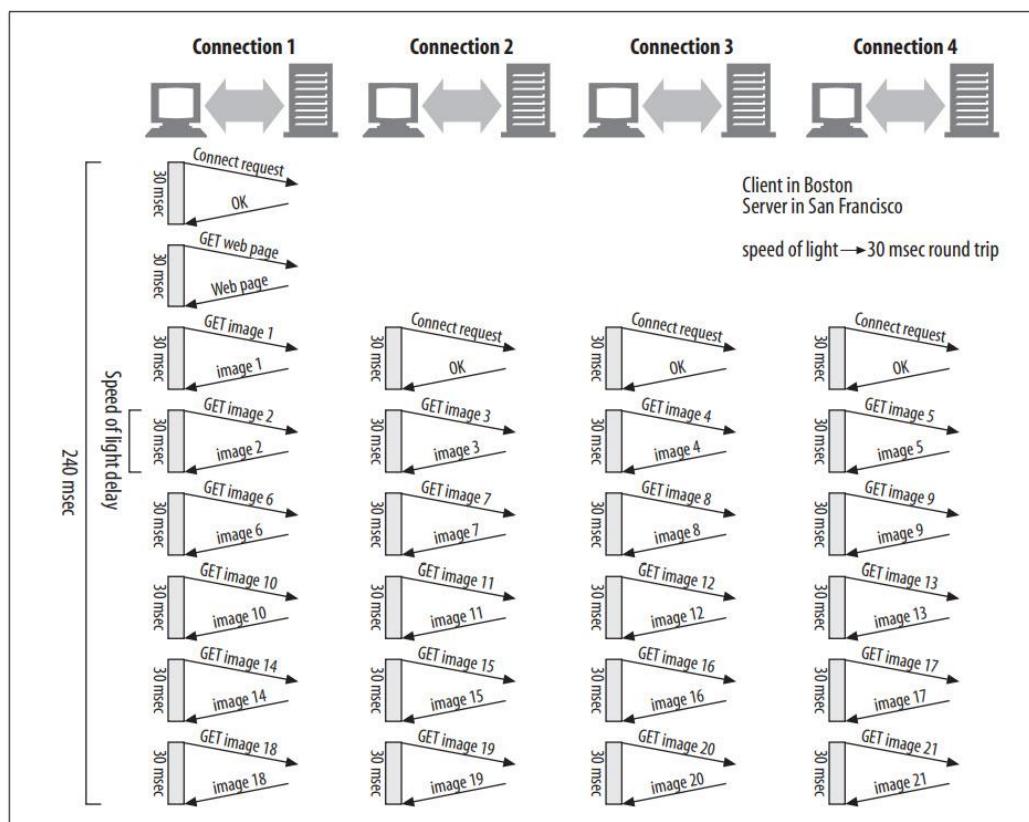


Distance Delays

حتی اگر پهنهای باند مشکلی نداشته باشد، ممکن است فاصله وجود داشته باشد. هر روتر شبکه تاخیرهایی را به ترافیک اینترنت اضافه می‌کند و حتی اگر تعداد زیادی روتر بین کلاینت و سرور وجود نداشته باشد، سرعت نور به تنها یابی می‌تواند باعث تاخیر قابل توجهی شود.

فاصله مستقیم بوسoton تا سانفرانسیسکو حدود ۲۷۰۰ مایل است. در بهترین حالت، با سرعت نور (۱۸۶۰۰۰ مایل در ثانیه)، یک سیگنال می‌تواند از بوسoton به سانفرانسیسکو در حدود ۱۵ میلی ثانیه حرکت کند و یک سفر رفت و برگشت را در ۳۰ میلی ثانیه کامل کند. (در واقعیت، سیگنال‌ها با سرعتی کمتر از نور حرکت می‌کنند، بنابراین تأخیر در مسافت حتی بدتر است)

فرض کنید یک صفحه وب حاوی ۲۰ تصویر کوچک است که همگی در سروری در سانفرانسیسکو قرار دارند. اگر کلاینت در بوسoton چهار اتصال موازی را به سرور باز کند و اتصالات را Alive نگه دارد، سرعت نور به تنها یابی تقریباً ۴۱ ثانیه (۲۴۰ میلی ثانیه) به زمان دانلود کمک می‌کند. اگر سرور در توکیو باشد (۶۷۰۰ مایل از بوسoton)، تاخیر به ۶۰۰ میلی ثانیه افزایش می‌یابد. صفحات وب نسبتاً پیچیده می‌توانند چندین ثانیه تاخیر در سرعت نور داشته باشند.

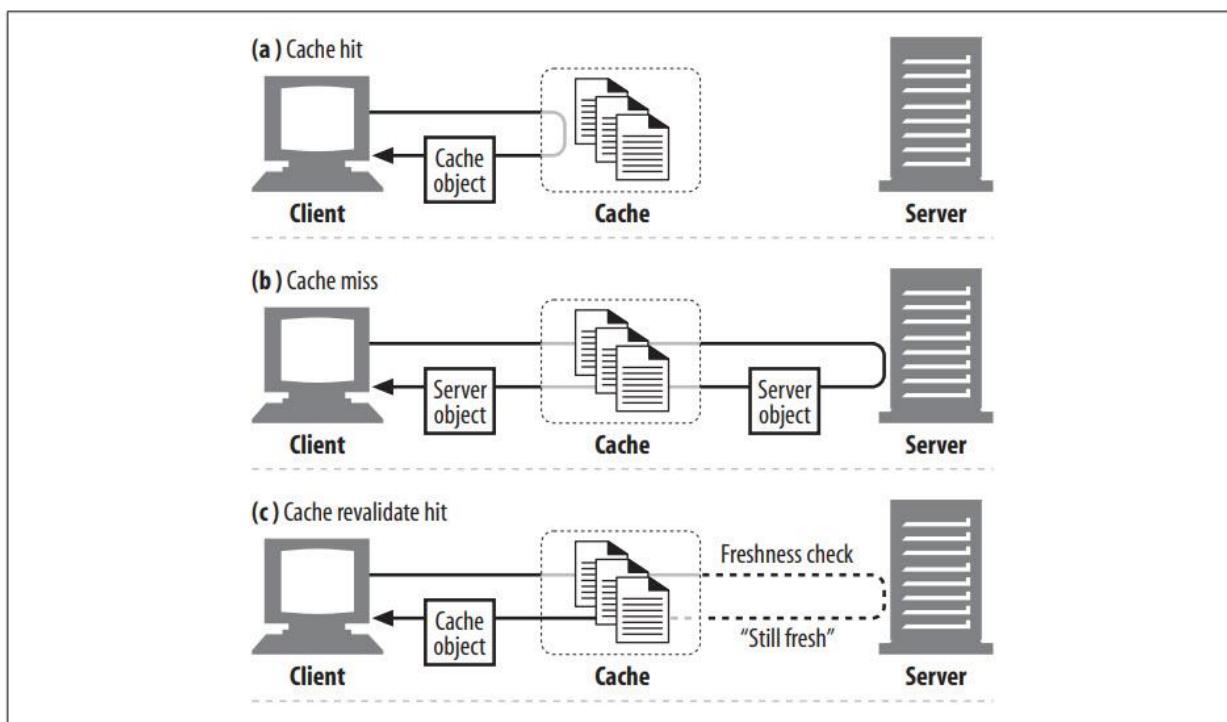


قرار دادن Cache در اتاق‌های ماشین مجاور می‌تواند مسافت سفر سند را از هزاران مایل به دهها یارد کاهش دهد.

Hits and Misses

بنابراین Cache‌ها می‌توانند به ما از جهات مختلفی کمک کنند. اما Cache یک کپی از هر سند در جهان را ذخیره نمی‌کند. (تعداد کمی از مردم می‌توانند یک Cache آنقدر بزرگ بخرند که تمام اسناد وب را در خود جای دهد. حتی اگر بتوانید «whole-Web caches» غول پیکر را بپردازید، برخی اسناد به قدری تغییر می‌کنند که در بسیاری از کش‌ها تازه نخواهند بود).

برخی از درخواست‌هایی که به یک Cache می‌رسند را می‌توان از یک نسخه موجود ارائه کرد. این cache hit نامیده می‌شود (بخش a از شکل زیر). درخواست‌های دیگر فقط به یک Cache می‌رسند تا به سرور مبداءرسال شوند، زیرا هیچ نسخه‌ای در دسترس نیست. به این cache miss می‌گویند (بخش b از شکل زیر).



Revalidations

از آنجایی که محتوای سرور اصلی می‌تواند تغییر کند، Cache‌ها باید هر از چند گاهی بررسی کنند که نسخه‌های آن‌ها هنوز با سرور به روز هستند. این «بررسی‌های تازه‌سازی» HTTP Revalidations نامیده می‌شوند.

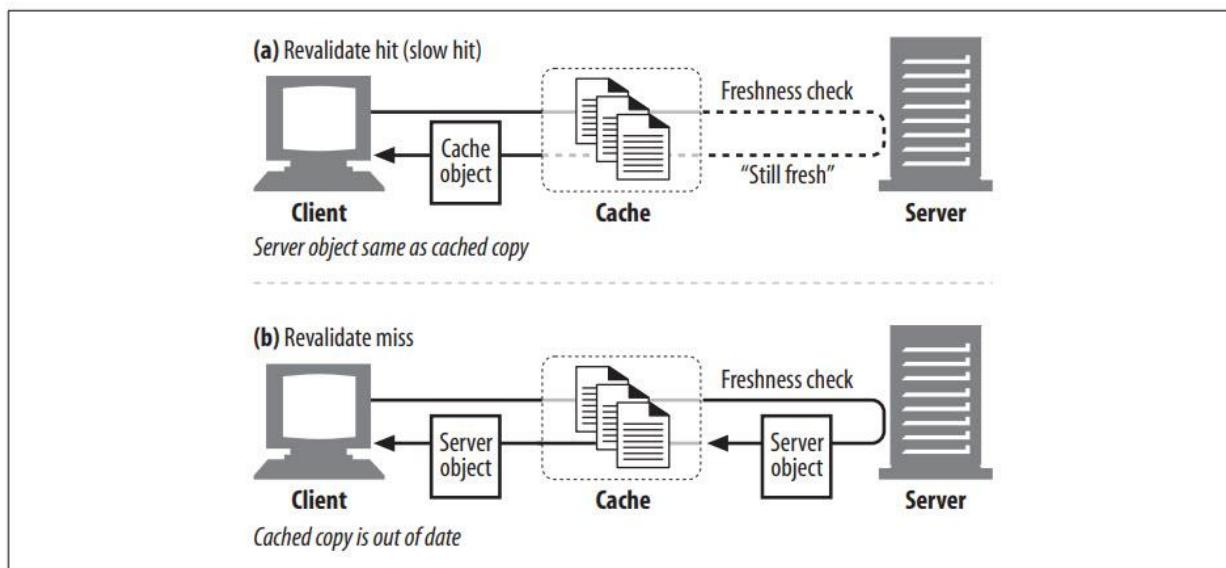




(بخش ۵ از شکل بالا). برای کارآمد کردن HTTP درخواست‌های ویژه‌ای را تعریف می‌کند که می‌تواند بدون واکشی کل شی از سرور، به سرعت بررسی کند که آیا محتوا هنوز تازه است یا خیر.

یک Cache می‌تواند هر زمان که بخواهد و هر چند وقت یکبار که بخواهد یک کپی را Revalidate کند. اما از آنجایی که Cache‌ها اغلب حاوی میلیون‌ها سند هستند، و به دلیل اینکه پهنای باند شبکه محدود است، بیشتر Cache‌ها یک نسخه را تنها زمانی که توسط کلاینت درخواست می‌شود و زمانی که نسخه به اندازه کافی قدیمی است که چک را تضمین کند، مجددأً تأیید می‌کنند.

هنگامی که یک Cache نیاز به Revalidation یک کپی Cache دارد، یک درخواست Cache کوچک به سرور مبدا ارسال می‌کند. اگر محتوا تغییر نکرده باشد، سرور با یک پاسخ Not Modified ۳۰۴ پاسخ می‌دهد. به محض اینکه Cache متوجه شد که کپی هنوز معتبر است، کپی را دوباره به طور موقت با علامت گذاری می‌کند و کپی را در اختیار کلاینت قرار می‌دهد. به این موضوع revalidate hit یا slow hit می‌گویند. این بخش کندر از یک pure cache hit است، زیرا نیاز به بررسی با سرور اصلی دارد، اما سریعتر از cache miss است، زیرا هیچ داده شی از سرور بازیابی نمی‌شود.



چند ابزار برای Revalidate HTTP مجدد اشیاء Cache در اختیار ما قرار می‌دهد، اما محبوب‌ترین آن‌ها هدر If-Modified-Since است. هنگامی که این هدر به یک درخواست GET اضافه می‌شود، به سرور می‌گوید که فقط در صورتی شیء را بفرستد که از زمان ذخیره شدن کپی تغییر یافته باشد.

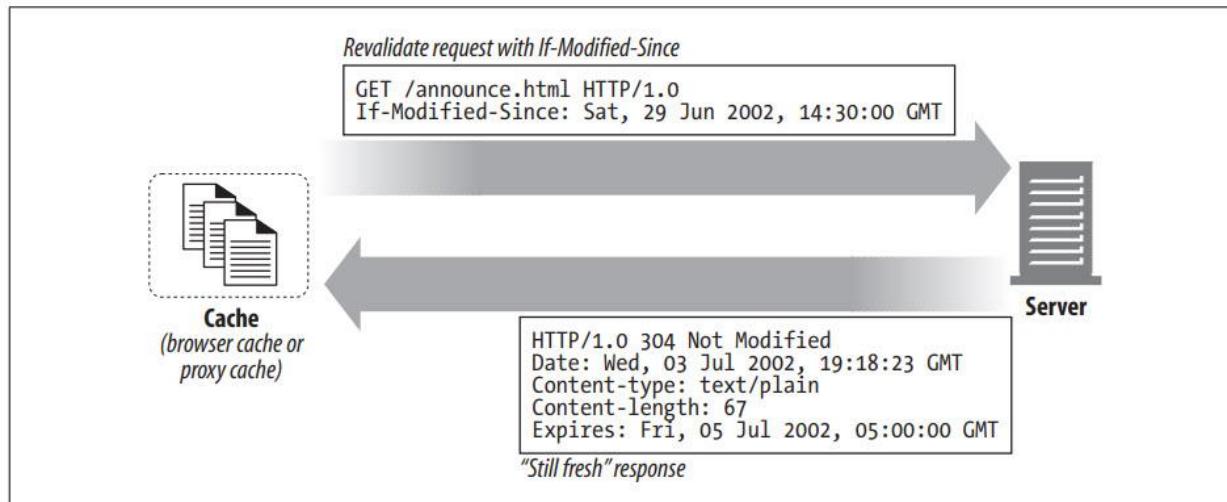




وقتی یک درخواست GET If-Modified-Since در سه حالت (زمانی که محتوای سرور تغییر نمی‌کند، زمانی که محتوای سرور تغییر کرده است و زمانی که شی روی سرور حذف می‌شود) به سرور می‌رسد، در اینجا چه اتفاقی می‌افتد:

Revalidate hit

اگر شی سرور اصلاح نشود، سرور یک پاسخ کوچک HTTP 304 Not Modified را برای کلاینت ارسال می‌کند. این در شکل زیر نشان داده شده است.



Revalidate miss

اگر شی سرور با کپی ذخیره شده متفاوت باشد، سرور یک پاسخ معمولی HTTP 200 OK را با محتوای کامل به کلاینت ارسال می‌کند.

Object deleted

اگر شی سرور حذف شده باشد، سرور یک پاسخ 404 Not Found را ارسال می‌کند و Cache کپی آن را حذف می‌کند.

Hit Rate

موضوع بعدی که باید به آن اشاره کرد Cache Hit Rate است. نرخ بازدید یا Hit Rate از ۰ تا ۱ متغیر است، اما اغلب به صورت درصدی توصیف می‌شود، که در آن ۰٪ به این معنی است که هر درخواستی اشتباه بود (باید سند را در سراسر شبکه دریافت کرد) و ۱۰۰٪ به این معنی است که هر درخواست Hit خورده است (یک کپی در Cache وجود داشته است)





Cache Hit Rate ها مایلند Cache Administrator واقعی که دریافت می‌کنید بستگی به بزرگی Cache، میزان مشابه بودن علائق کاربران Cache، تعداد دفعات تغییر یا شخصی‌سازی داده‌های ذخیره‌شده در Cache و نحوه پیکربندی Cache‌ها دارد.

پیش‌بینی Hit Rate بسیار دشوار است، اما Web Cache متوسط امروزی، مناسب است.

نکته خوب در مورد Cache این است که حتی یک Cache با اندازه متوسط ممکن است حاوی اسناد محبوب کافی باشد تا عملکرد را به میزان قابل توجهی بهبود بخشد و ترافیک را کاهش دهد. Cache‌ها سخت کار می‌کنند تا اطمینان حاصل کنند که محتوای مفید در Cache باقی می‌ماند.

Byte Hit Rate

با این حال، Document Hit Rate، کل داستان را بیان نمی‌کند، زیرا اسناد همگی یک اندازه نیستند. برخی از اشیاء بزرگ ممکن است کمتر مورد دسترسی قرار گیرند، اما به دلیل اندازه آن‌ها، بیشتر به ترافیک کلی داده کمک می‌کنند. به همین دلیل، برخی از افراد معیار Byte Hit Rate را ترجیح می‌دهند (به ویژه آن دسته از افرادی که برای هر بایت ترافیک صورتحساب دریافت می‌کنند!).

Byte Hit Rate نشان دهنده کسری از تمام بایت‌های منتقل شده است که از Cache ارائه شده است.

این متريک میزان صرفه جویی در ترافیک را نشان می‌دهد. ۱۰۰ درصدی به اين معني است که هر بایت از Cache آمده است و هیچ ترافیکی در اينترنت وجود ندارد.

Byte Hit Rate و Document Hit Rate هر دو معیارهای مفیدی برای عملکرد Cache هستند. Document Hit Rate توصیف می‌کند که چه تعداد از تراکنش‌های وب از شبکه خروجی خارج می‌شوند. از آنجایی که تراکنش‌ها دارای یک جزء زمان ثابت هستند که اغلب می‌تواند بزرگ باشد (برای مثال، راهاندازی اتصال TCP به یک سرور)، بهبود Document Hit Rate برای کاهش تأخیر کلی (تأخير) بهینه‌سازی می‌شود. نرخ ضربه بایت توصیف می‌کند که چند بایت از اینترنت خارج می‌شود. بهبود Byte Hit Rate باعث صرفه جویی در پهنه‌ای باند می‌شود.



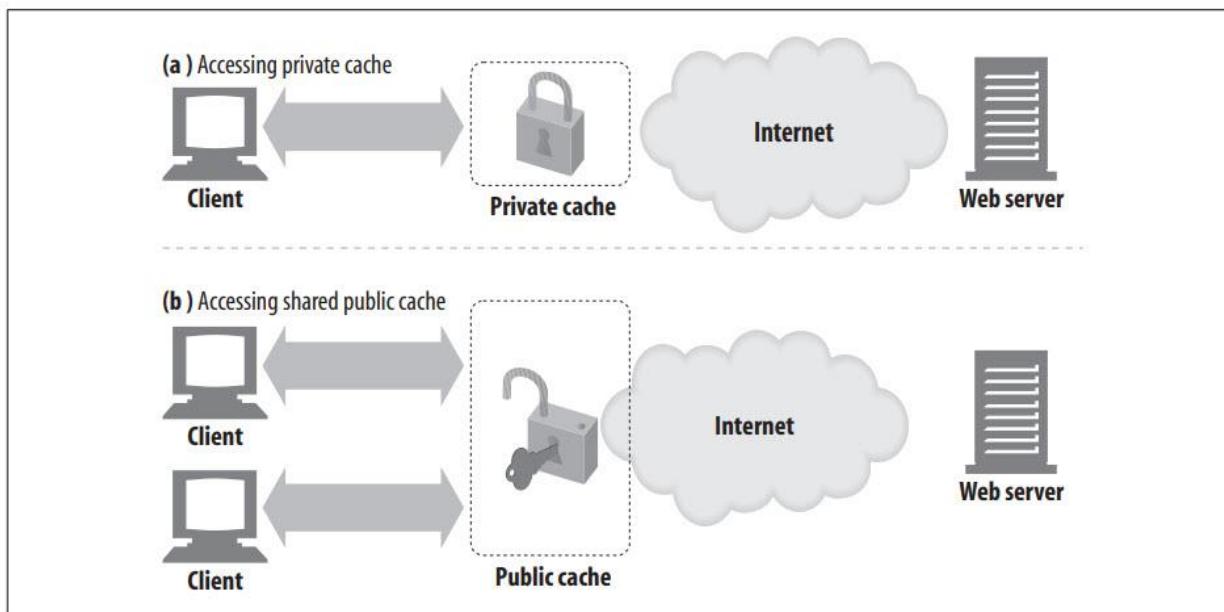
Distinguishing Hits and Misses

متأسفانه، HTTP هیچ راهی برای کلاینت فراهم نمی‌کند که بگوید آیا پاسخ Cache Hit بوده یا دسترسی به سرور مبدا. در هر دو حالت، کد پاسخ 200 OK خواهد بود که نشان می‌دهد پاسخ دارای بدنه است. برخی از Cache‌های پروکسی تجاری، اطلاعات اضافی را به هدرهای Via متصل می‌کنند تا آنچه را که در اتفاق افتاده است، توصیف کنند.

یکی از راههایی که یک کلاینت معمولاً می‌تواند تشخیص دهد که آیا پاسخ از یک Cache است یا خیر، استفاده از هدر Date است. با مقایسه مقدار هدر Date در پاسخ با زمان فعلی، یک کلاینت اغلب می‌تواند یک پاسخ ذخیره شده در Cache را با مقدار تاریخ قدیمی آن تشخیص دهد. روش دیگری که کلاینت می‌تواند پاسخ ذخیره شده را تشخیص دهد، هدر Age است که نشان دهنده سن پاسخ است.

Cache Topologies

ها را می‌توان به یک کاربر اختصاص داد یا بین هزاران کاربر به اشتراک گذاشت. Cache‌های اختصاصی Cache را Private Cache می‌نامند. Private Cache‌های شخصی هستند که حاوی صفحات محبوب Public Cache می‌نامند. برای یک کاربر هستند (بخش a شکل زیر). Public Cache را مشترک را می‌نامند. Public Cache‌ها حاوی صفحاتی هستند که در جامعه کاربران محبوب هستند (بخش b از شکل زیر).





Private Caches

ها به فضای ذخیره سازی زیادی نیاز ندارند، بنابراین می‌توان آن‌ها را کوچک و ارزان ساخت. مرورگرهای وب دارای Private Cache هستند، اغلب مرورگرها اسناد رایج را در دیسک و حافظه رایانه شخصی شما ذخیره می‌کنند و به شما امکان می‌دهند اندازه و تنظیمات Cache را پیکربندی کنید. همچنین می‌توانید به درون Cache مرورگر نگاهی بیاندازید تا ببینید چه چیزی در آن‌ها وجود دارد.

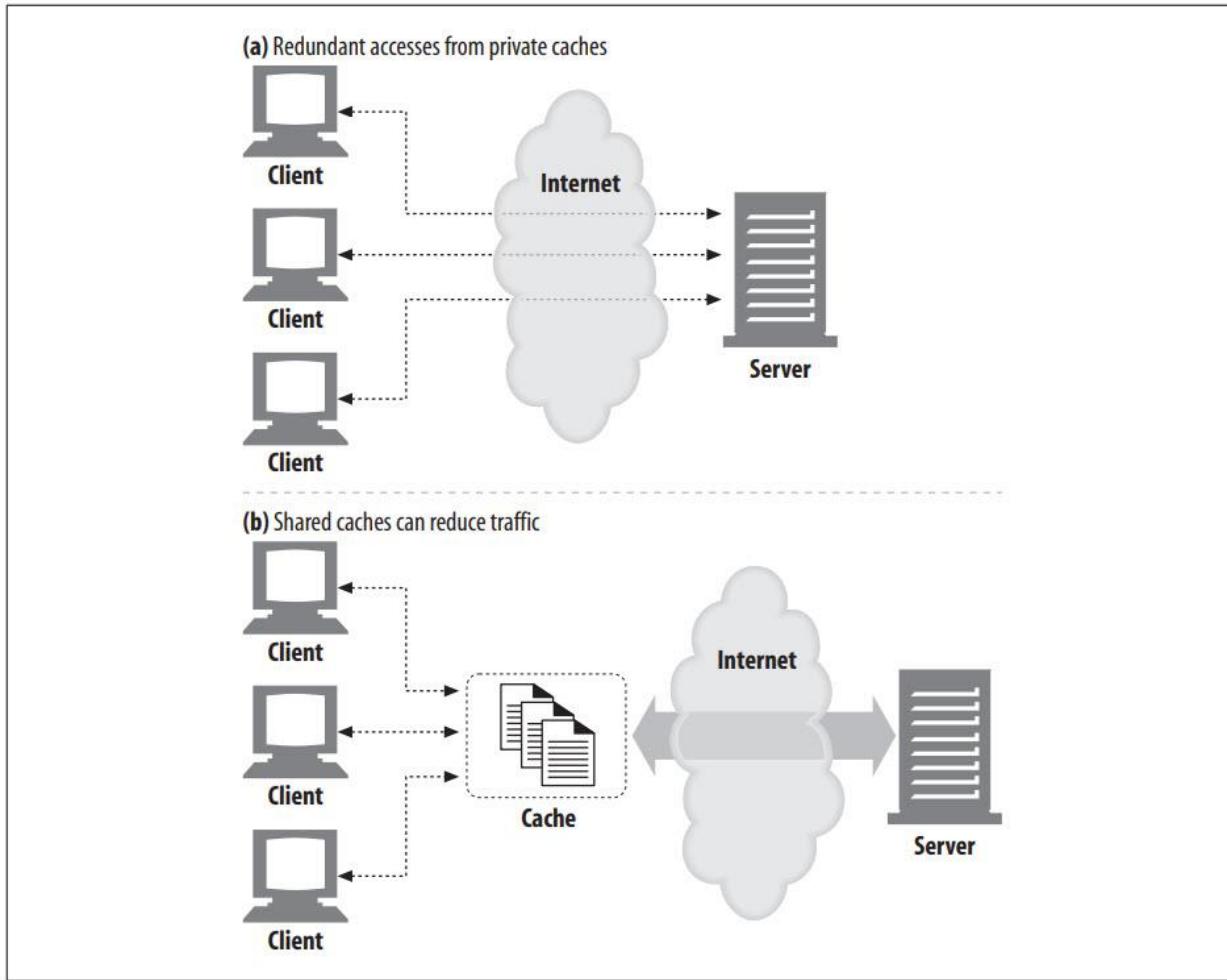
به عنوان مثال در IE، شما می‌توانید محتويات Cache را از کادر محاوره ای Cache دریافت کنید. IE اسناد Cache شده را "Temporary Files" می‌نامد و آن‌ها را به همراه URL‌های مرتبط و زمان انقضای اسناد در یک نمایش فایل فهرست می‌کند. می‌توانید محتويات Cache مربوط به Navigator را از طریق URL «Disk Cache statistics» ویژه about:cache مشاهده کنید که صفحه Cache را نشان می‌دهد. به شما بر می‌گرداند که محتويات Cache را نشان می‌دهد.

Public Proxy Caches

Public Cache ها، سرورهای پرآکسی خاص و مشترکی هستند که به آن‌ها Caching Proxy Server یا Proxy Cache معمولاً می‌گویند. Local Cache ها اسناد را از ارائه می‌دهند یا از طرف کاربر با سرور تماس می‌گیرند. از آنجا که یک Public Cache از چندین کاربر دسترسی دریافت می‌کند، فرصت بیشتری برای حذف ترافیک اضافی دارد.

در بخش a شکل زیر، هر کلاینت به طور مضاعف به یک سند جدید و «داع» (هنوز در Cache دسترسی پیدا می‌کند. هر Private Cache یک سند را fetch می‌کند و چندین بار از شبکه عبور می‌کند. با یک Cache عمومی و مشترک، مانند بخش b از شکل زیر، Cache تنها یک بار باید شی محبوب را fetch کند و از کپی مشترک برای سرویس دهی به همه درخواست‌ها استفاده می‌کند و ترافیک شبکه را کاهش می‌دهد.



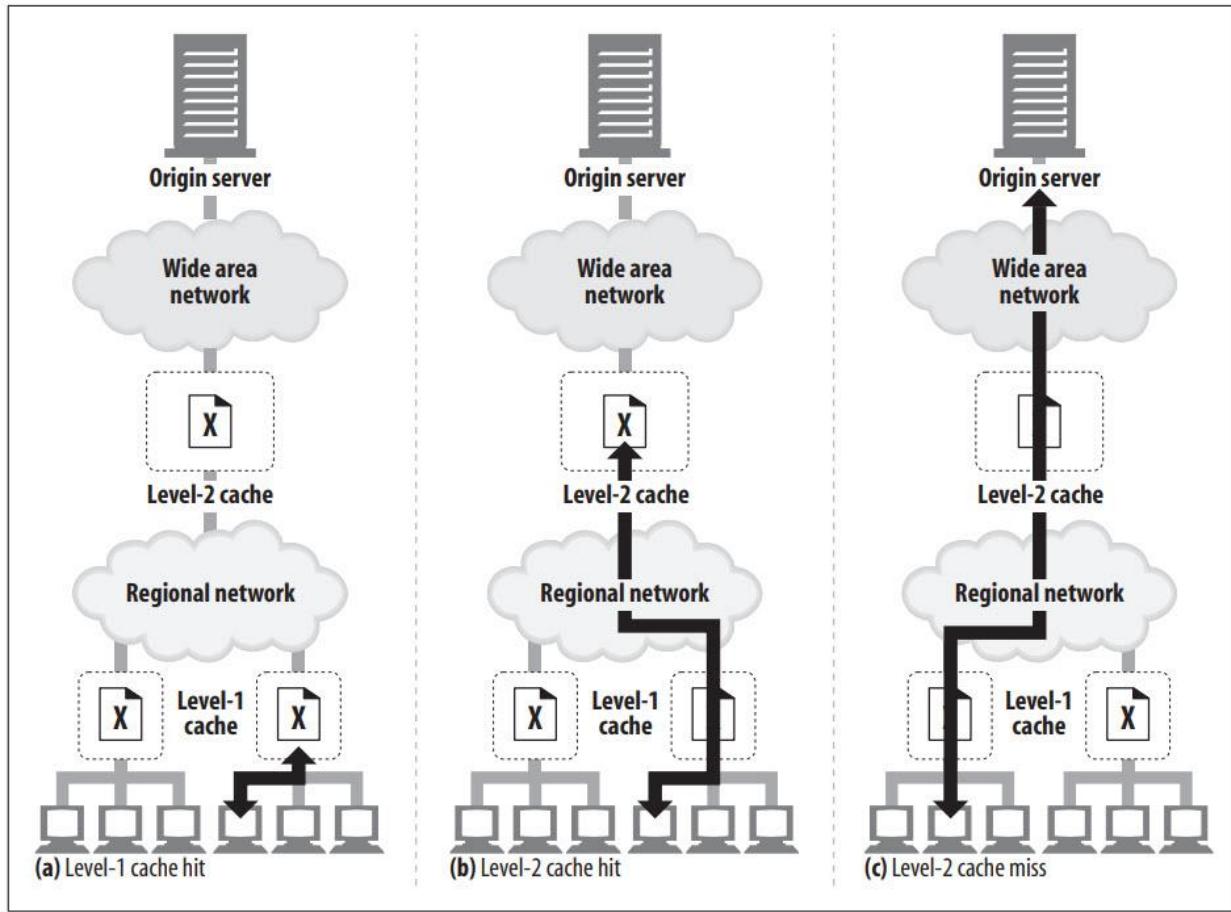


ها از قوانین مربوط به پروکسی‌ها پیروی می‌کنند که در فصل ۶ توضیح داده شده است. می‌توانید مرورگر خود را برای استفاده از Proxy Cache با تعیین یک پروکسی دستی یا با پیکربندی فایل پیکربندی خودکار پروکسی پیکربندی کنید. همچنین می‌توانید درخواست‌های HTTP را از طریق Cache بدون پیکربندی مرورگر خود با استفاده از پروکسی‌های رهگیری Force کنید.

Proxy Cache Hierarchies

در عمل، اغلب منطقی است که سلسله‌مراتب Cache‌ها را مستقر کنیم، جایی که از دست دادن در Cache‌های کوچک‌تر به Cache‌های والد بزرگ‌تر هدایت می‌شوند که به ترافیک باقی‌مانده «distilled» Cache‌سرویس می‌دهند. شکل زیر سلسله‌مراتب Cache دو سطحی را نشان می‌دهد. ایده این است که از Cache‌های کوچک و ارزان قیمت در نزدیکی کلاینت‌ها و به تدریج بزرگ‌تر و قدرتمندتر در سلسله‌مراتب برای نگهداری اسناد به اشتراک گذاشته شده توسعه بسیاری از کاربران استفاده شود.





امیدواریم که اکثر کاربران در Level-1 Cache نزدیک (همانطور که در بخش a شکل نشان داده شده است) از بازدید کنند. در غیر این صورت، Cache والدین بزرگتر ممکن است قادر به رسیدگی به درخواستهای آن‌ها باشند (بخش a شکل).

برای سلسله مراتب Cache عمیق، می‌توان از زنجیره‌های طولانی Cache عبور کرد، اما هر پروکسی مداخله گر جریمه عملکردی را اعمال می‌کند که با طولانی شدن زنجیره پروکسی قابل توجه است.

Cache Meshes, Content Routing, and Peering

برخی از معماران شبکه به جای سلسله مراتب‌های Cache ساده، Cache Meshes های پیچیده می‌سازند. Proxy Cache ها در شبکه‌های Cache به روش‌های پیچیده‌تری با یکدیگر صحبت می‌کنند و تصمیمات ارتباطی Cache پویا را می‌گیرند، تصمیم می‌گیرند که با کدام Cache صحبت کنند، یا تصمیم می‌گیرند که Proxy Cache ها به طور کامل دور بزنند و خود را به سمت سرور مبدا هدایت کنند. چنین Content Routers (Content Routers) توصیف کرد، زیرا آن‌ها تصمیمات مسیریابی در مورد نحوه دسترسی، مدیریت و ارائه محتوا را می‌گیرند.

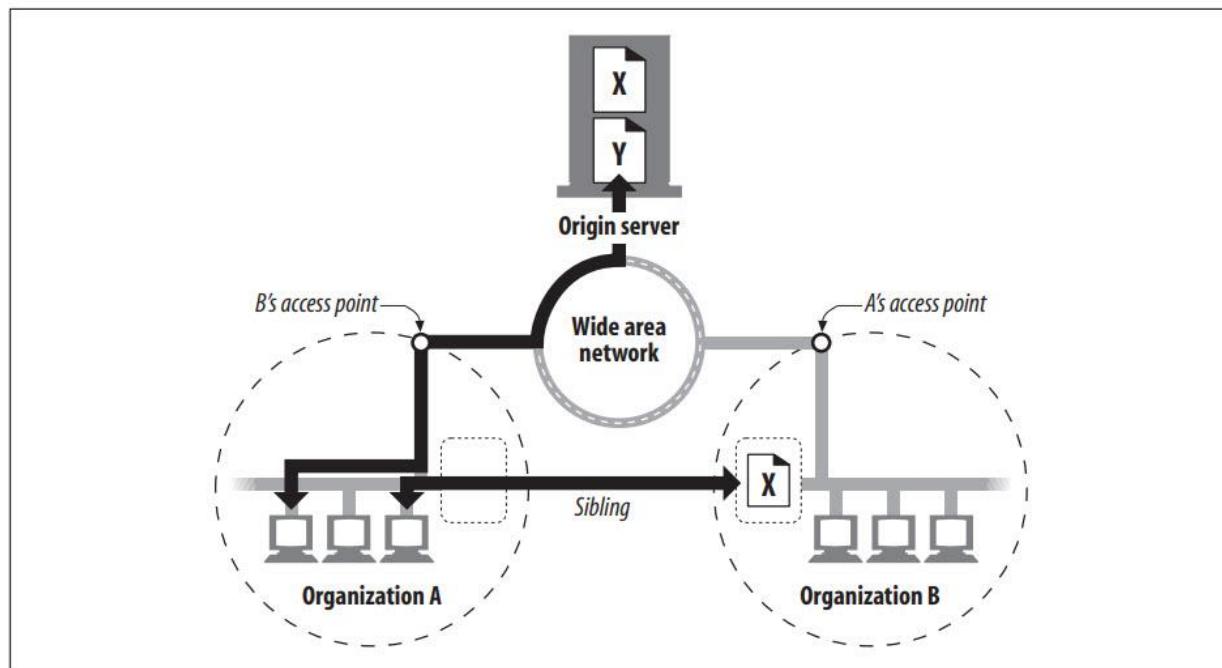




های طراحی شده برای مسیریابی محتوا در شبکهای Cache ممکن است تمام کارهای زیر را انجام دهند:

- بر اساس URL، بین Parent Cache یا سرور مبدأ به صورت پویا انتخاب را انجام دهند.
- یک Parent Cache خاص را به صورت پویا و بر اساس URL انتخاب کنند.
- قبل از رفتن به Cache، Parent Cache را در منطقه محلی برای یک کپی ذخیره شده جستجو کنند.
- به سایر Cache ها اجازه دسترسی به بخش‌هایی از محتوای ذخیره شده خود را بدهند، اما اجازه ترانزیت اینترنت از طریق Cache خود را ندهند.

این روابط پیچیده‌تر بین Cache ها به سازمان‌های مختلف اجازه می‌دهد تا با یکدیگر همتا شوند و Cache خود را برای منافع متقابل به هم متصل کنند. Cache هایی که پشتیبانی همتا انتخابی را فراهم می‌کنند، Sibling Caches نامیده می‌شوند. از آنجایی که HTTP پشتیبانی از Sibling Caches را فراهم نمی‌کند، HTTP را با HTCP HyperText Caching Protocol یا ICP Internet Cache Protocol یا گسترش داده اند. ما در مورد این پروتکل‌ها در فصل ۲۰ صحبت خواهیم کرد.



Cache Processing Steps

های تجاری مدرن بسیار پیچیده هستند. آن‌ها به گونه‌ای ساخته شده‌اند که عملکرد بسیار بالایی داشته باشند و از ویژگی‌های پیشرفته HTTP و سایر فناوری‌ها پشتیبانی کنند. اما، علیرغم



برخی جزئیات ظریف، کار اساسی یک Cache وب عمده ساده است. یک دنباله اصلی پردازش Cache برای یک پیام HTTP GET از هفت مرحله تشکیل شده است:

پیام درخواست ورودی را از شبکه می‌خواند.

پیام را تجزیه نموده، URL و هدرها را استخراج می‌کند.

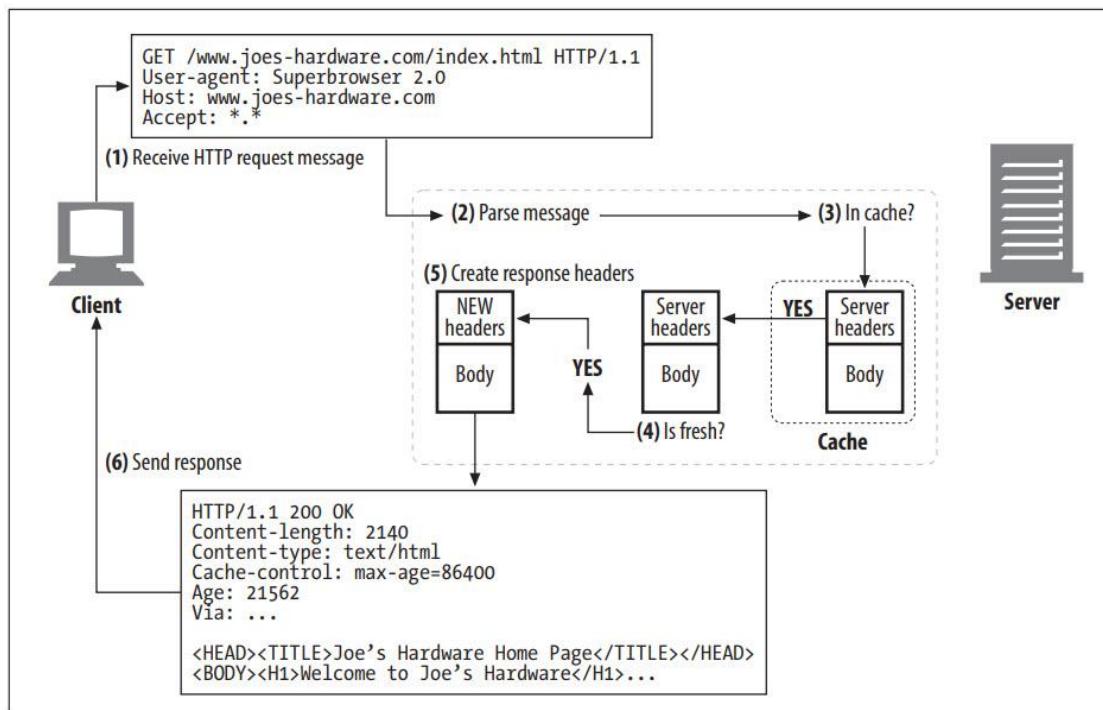
بررسی می‌کند که آیا یک نسخه محلی در دسترس است یا نه و در غیر این صورت، یک کپی را Fetch می‌کند (و آن را به صورت محلی ذخیره می‌نماید).

بررسی می‌کند که آیا نسخه Cache شده به اندازه کافی تازه است یا خیر و در غیر این صورت، از سرور به روز رسانی لازم را درخواست می‌کند.

یک پیام پاسخ با هدرهای جدید و بدن ذخیره شده در Cache ایجاد می‌کند.

پاسخ را از طریق شبکه به کلاینت ارسال می‌کند.

به صورت اختیاری، Cache یک ورودی فایل گزارش را ایجاد می‌کند که تراکنش را توصیف می‌نماید.





Step 1: Receiving

در مرحله ۱، Cache فعالیت در اتصال شبکه را شناسایی کرده و داده‌های دریافتی را می‌خواند. Cache های با کارایی بالا، داده‌ها را به طور همزمان از چندین اتصال ورودی می‌خوانند و قبل از رسیدن کل پیام، پردازش تراکنش را آغاز می‌کنند.

Step 2: Parsing

سپس، Cache پیام درخواست را به قطعات تجزیه می‌کند و قسمتهای هدر را در ساختارهای داده‌ای قرار می‌دهد که به راحتی قابل دستکاری هستند. این کار باعث می‌شود که نرمافزار ذخیره‌سازی بتواند فیلدهای هدر را آسان‌تر پردازش کند و با آن‌ها کار کند.

Step 3: Lookup

در مرحله ۳، Cache URL را می‌گیرد و یک نسخه محلی را بررسی می‌کند. کپی محلی ممکن است در حافظه، روی یک دیسک محلی یا حتی در رایانه دیگری در نزدیکی ذخیره شود. Cache های درجه حرfe ای از الگوریتم‌های سریع برای تعیین اینکه آیا یک شی در Cache محلی موجود است یا خیر استفاده می‌کنند. اگر سند به صورت محلی در دسترس نباشد، می‌توان آن را از سرور مبدا یا یک پروکسی والد دریافت کرد یا بر اساس وضعیت و پیکربندی، یک خطرا برگرداند.

شی ذخیره شده حاوی بدن پاسخ سرور و هدرهای پاسخ سرور اصلی است، بنابراین هدرهای صحیح سرور را می‌توان در طول ضربه حافظه پنهان یا Cache Hit برگرداند. شیء ذخیره شده همچنین شامل برخی متادادا است که برای حسابداری مدت زمان استفاده از شی در Cache، تعداد دفعات استفاده از آن و غیره استفاده می‌شود.

Step 4: Freshness Check

اجازه می‌دهد تا کپی‌هایی از اسناد سرور را برای مدتی نگه دارد. در طول این مدت، سند "fresh" در نظر گرفته می‌شود و حافظه پنهان می‌تواند سند را بدون تماس با سرور ارائه دهد. اما هنگامی که کپی ذخیره شده برای مدت طولانی در اطراف باقی بماند و از حد تازه بودن سند گذشته باشد، شیء "stale" یا کهنه در نظر گرفته می‌شود و Cache باید مجدداً با سرور اعتبارسنجی شود تا هرگونه تغییر سند را قبل از ارائه بررسی کند.





HTTP دارای مجموعه‌ای از قوانین بسیار پیچیده برای بررسی تازگی است که به دلیل تعداد زیادی از گزینه‌های پیکربندی پشتیبانی از محصولات Cache و نیاز به همکاری با استانداردهای تازگی غیر HTTP بدتر شده است. ما بیشتر بقیه این فصل را به توضیح محاسبات تازگی اختصاص خواهیم داد.

Step 5: Response Creation

از آنجایی که می‌خواهیم پاسخ ذخیره شده در Cache به نظر برسد که از سرور مبدا آمده است، Cache از هدرهای پاسخ سرور ذخیره شده به عنوان نقطه شروع برای هدرهای پاسخ استفاده می‌کند. سپس این هدرهای پایه توسط Cache اصلاح و تقویت می‌شوند.

حافظه پنهان وظیفه تطبیق هدرها برای مطابقت با کلاینت را بر عهده دارد. به عنوان مثال، سرور ممکن است یک پاسخ HTTP/1.0 (یا حتی یک پاسخ HTTP/0.9) را برگرداند، در حالی که کلاینت انتظار پاسخ HTTP/1.1 را دارد، در این صورت Cache باید هدرها را بر اساس آن ترجمه کند. Cache‌ها همچنین اطلاعات تازه بودن Cache (هدرهای Cache-Control، Age، Expires و Cache-Control) را درج می‌کنند و اغلب شامل یک هدر Via می‌شوند تا توجه داشته باشند که یک Cache پروکسی درخواست را ارائه می‌دهد.

توجه داشته باشید که یک Date هدر Date را تنظیم کند. هدر Date نشان دهنده تاریخ شیء است که در ابتدا در سرور مبدا ایجاد شده است.

Step 6: Sending

پس از آماده شدن هدرهای پاسخ، Cache پاسخ را به کلاینت ارسال می‌کند. مانند تمام سرورهای پروکسی، یک Cache پروکسی نیاز به مدیریت ارتباط با کلاینت دارد. Cache‌های با کارایی بالا برای ارسال کارآمد داده‌ها سخت کار می‌کنند و اغلب از کپی محتوای سند بین حافظه محلی و بافرهای I/O شبکه اجتناب می‌کنند.

Step 7: Logging

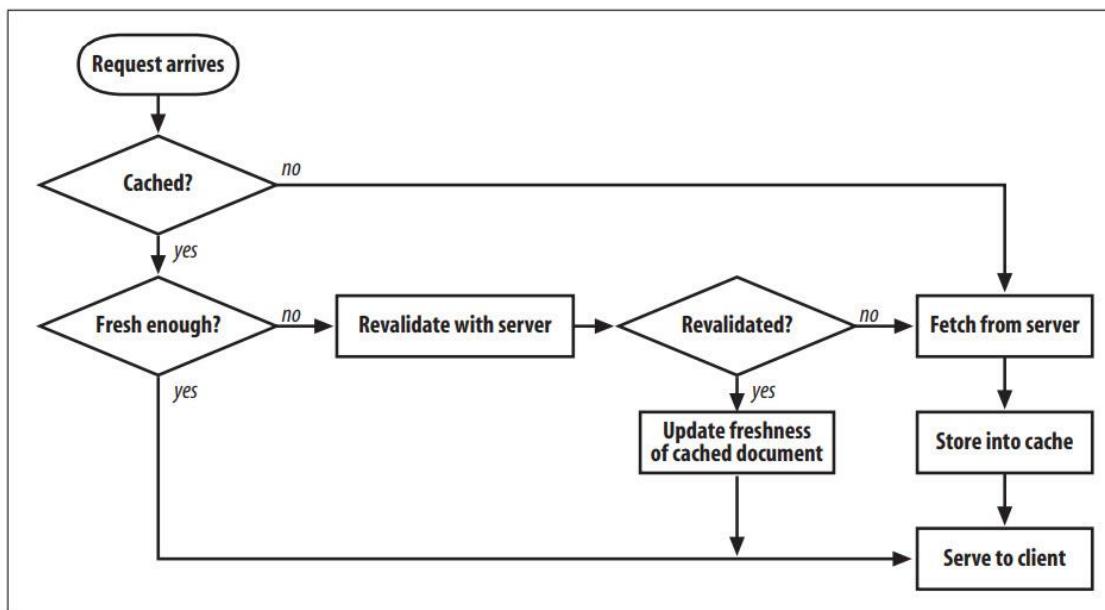
اکثر Cache‌ها فایل‌های گزارش و آمار مربوط به استفاده از Cache را نگه می‌دارند. پس از تکمیل هر تراکنش Cache، حافظه نهان آماری را با شمارش تعداد بازدید و از دست رفتن Cache (و سایر معیارهای مربوطه) به روزرسانی می‌کند و ورودی را در یک فایل گزارش وارد می‌کند که نوع درخواست، URL و آنچه اتفاق افتاده را نشان می‌دهد.

محبوب‌ترین فرمتهای گزارش Cache، فرمت log Squid و فرمت Cache توسعه یافته گزارش مشترک هستند، اما بسیاری از محصولات Cache به شما امکان می‌دهند فایل‌های ثبت سفارشی ایجاد کنید. ما در فصل ۲۱ به تفصیل در مورد فرمتهای فایل log بحث می‌کنیم.



Cache Processing Flowchart

شکل زیر به صورت ساده نشان می‌دهد که چگونه یک Cache یک درخواست برای دریافت URL را پردازش می‌کند.



Keeping Copies Fresh

ممکن است همه کپی‌های ذخیره شده با اسناد روی سرور سازگار نباشند. از این گذشته، اسناد در طول زمان تغییر می‌کنند. گزارش‌ها ممکن است ماهانه تغییر کند. روزنامه‌های آنلاین هر روز تغییر می‌کنند. داده‌های مالی ممکن است هر چند ثانیه یکبار تغییر کند. اگر Cache‌ها همیشه داده‌های قدیمی را ارائه می‌کردند بی فایده بودند. داده‌های ذخیره شده باید تا حدی با داده‌های سرور سازگاری داشته باشند.

HTTP شامل مکانیسم‌های ساده‌ای است که داده‌های Cache را به اندازه کافی با سرورها سازگار نگه می‌دارد، بدون اینکه سرورها مجبور شوند به خاطر بسیارند کدام Cache‌ها کپی اسناد خود را دارند. HTTP این مکانیسم‌های ساده را Server Revalidation و Document Expiration می‌نامد.

Document Expiration

HTTP به سرور مبدأ اجازه می‌دهد تا با استفاده از هدرهای HTTP Cache-Control و Expires یک "Expiration Date" را به هر سند متصل کند. مانند تاریخ انقضای یک ماده غذایی، این هدرها تعیین می‌کنند که چه مدت باید محتوای تازه را مشاهده کنید.





```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Expires: Fri, 05 Jul 2002, 05:00:00 GMT
Independence Day sale at Joe's Hardware
Come shop with us today!
```

(a) Expires header

```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Cache-Control: max-age=484200
Independence Day sale at Joe's Hardware
Come shop with us today!
```

(b) Cache-Control:max-age header

تا زمانی که یک سند Cache منقضی شود، Cache می‌تواند هر چند وقت یکبار که می‌خواهد، بدون تماس با سرور، کپی را ارائه کند—مگر اینکه، درخواست کلاینت شامل هدرهایی باشد که از ارائه یک منبع ذخیره‌شده یا نامعتبر جلوگیری می‌کند. اما، هنگامی که سند ذخیره شده منقضی شد، Cache باید با سرور بررسی کند که آیا سند تغییر کرده است یا خیر، و اگر چنین است، یک نسخه جدید (با تاریخ انقضا جدید) دریافت کند.

Expiration Dates and Ages

سرورها تاریخ انقضا را با استفاده از HTTP/1.1 Cache-Control:max-age یا هدرهای HTTP/1.0+ Expires Cache-Control: max-age و هدرهای Expires ارائه می‌کنند. هدرهای Expires اساساً همین کار را انجام می‌دهند، اما هدر جدیدتر Cache-Control ترجیح داده می‌شود، زیرا از یک زمان نسبی به جای تاریخ مطلق استفاده می‌کند. تاریخ‌های مطلق به تنظیم صحیح ساعت‌های رایانه بستگی دارد. جدول زیر هدرهای پاسخ انقضا را فهرست می‌کند.

Header	Description
Cache-Control: max-age	The max-age value defines the maximum age of the document—the maximum legal elapsed time (in seconds) from when a document is first generated to when it can no longer be considered fresh enough to serve. Cache-Control: max-age=484200
Expires	Specifies an absolute expiration date. If the expiration date is in the past, the document is no longer fresh. Expires: Fri, 05 Jul 2002, 05:00:00 GMT

فرض کنید امروز ۲۹ ژوئن ۲۰۰۲ ساعت ۹:۳۰ صبح به وقت استاندارد شرقی (EST) است و فروشگاه سخت افزار Joe در حال آماده شدن برای فروش چهارم جولای است (فقط پنج روز دیگر). جو می‌خواهد یک صفحه وب ویژه را روی وب سرور خود قرار دهد و آن را تنظیم کند تا در نیمه شب EST در شب ۵ ژوئن ۲۰۰۲ منقضی شود. اگر سرور جو از هدرهای Expires به سبک قدیمی‌تر استفاده کند، پیام پاسخ سرور (بخش a تصویر زیر) ممکن است شامل این هدر باشد:

Expires: Fri, 05 Jul 2002, 05:00:00 GMT





```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Expires: Fri, 05 Jul 2002, 05:00:00 GMT
Independence Day sale at Joe's Hardware
Come shop with us today!
```

(a) Expires header

```
HTTP/1.0 200 OK
Date: Sat, 29 Jun 2002, 14:30:00 GMT
Content-type: text/plain
Content-length: 67
Cache-Control: max-age=484200
Independence Day sale at Joe's Hardware
Come shop with us today!
```

(b) Cache-Control:max-age header

اگر سرور Joe از هدرهای جدیدتر Cache-Control: max-age استفاده کند، پیام پاسخ سرور (بخش b تصویر بالا) ممکن است حاوی این هدر باشد:

Cache-Control: max-age=484200

در صورتی که بلافاصله مشخص نبود، ۴۸۴۲۰۰ ثانیه تعداد ثانیه‌های بین تاریخ فعلی، ۲۹ ژوئن ۲۰۰۲ در ساعت ۹:۳۰ صبح به وقت شرقی و تاریخ پایان فروش، ۵ ژوئیه ۲۰۰۲ در نیمه شب است. ۱۳۴.۵ ساعت (حدود ۵ روز) تا پایان فروش باقی مانده است. با ۳۶۰۰ ثانیه در هر ساعت، ۴۸۴۲۰۰ ثانیه تا پایان فروش باقی می‌ماند.

Server Revalidation

فقط به این دلیل که یک سند Cache شده منقضی شده است به این معنی نیست که در واقع با آنچه در سرور مبدا قرار دارد، متفاوت است. این فقط به این معنی است که زمان بررسی است. به این موضوع "تأیید مجدد سرور" می‌گویند، به این معنی که Cache باید از سرور مبدا بپرسد که آیا سند تغییر کرده است یا خیر:

- اگر اعتبار مجدد نشان دهد که محتوا تغییر کرده است، Cache یک کپی جدید از سند دریافت می‌کند، آن را به جای داده‌های قدیمی ذخیره می‌کند و سند را برای کلاینت ارسال می‌کند.
- اگر اعتبار مجدد نشان دهد که محتوا تغییر نکرده است، Cache فقط هدرهای جدید، از جمله تاریخ انقضا جدید، را دریافت نموده و هدرهای موجود در Cache را به روز می‌کند.

این یک سیستم خوب است. Cache لازم نیست برای هر درخواستی تازه بودن سند را تأیید کند - فقط زمانی که سند منقضی شود باید دوباره بوسیله سرور تأیید شود. این موضوع باعث صرفه جویی در ترافیک سرور می‌شود و زمان پاسخگویی بهتر کاربر را بدون ارائه محتوای قدیمی فراهم می‌کند.

پروتکل HTTP برای برگرداندن یکی از موارد زیر به یک Cache به رفتار صحیح نیاز دارد:

- یک کپی Cache شده که "به اندازه کافی تازه" است.





- یک نسخه Cache که برای اطمینان از تازه بودن آن مجدداً با سرور تأیید شده است.
- یک پیغام خطا، اگر سرور مبدا برای تأیید مجدد خراب باشد.
- یک کپی Cache شده با یک هشدار پیوست شده مبنی بر اینکه ممکن است نادرست باشد.

Revalidation with Conditional Methods

متدهای شرطی HTTP اعتبار سنجی مجدد را کارآمد می‌کند. HTTP به اجازه می‌دهد تا یک «GET مشروط» را به سرور مبدا ارسال کند، و از سرور می‌خواهد که بدنہ شی را تنها در صورتی که سند با نسخه‌ای که در حال حاضر در Cache است متفاوت باشد، ارسال کند. به این ترتیب، بررسی تازگی و واکشی شی در یک GET شرطی واحد ترکیب می‌شوند. GET های شرطی با افزودن هدرهای شرطی ویژه به پیام‌های درخواستی آغاز می‌شوند. وب سرور فقط در صورتی شی را برمی‌گرداند که شرط درست باشد.

HTTP پنج هدر درخواست شرطی را تعریف می‌کند. دو موردی که برای اعتبار سنجی مجدد Cache مفید هستند، If-None-Match و If-Modified-Since هستند. همه هدرهای شرطی با پیشوند "If" شروع می‌شوند. جدول زیر هدرهای پاسخ شرطی مورد استفاده در اعتبار سنجی مجدد Cache را فهرست می‌کند.

Header	Description
If-Modified-Since: <date>	Perform the requested method if the document has been modified since the specified date. This is used in conjunction with the Last-Modified server response header, to fetch content only if the content has been modified from the cached version.
If-None-Match: <tags>	Instead of matching on last-modified date, the server may provide special tags (see "ETag" in Appendix C) on the document that act like serial numbers. The If-None-Match header performs the requested method if the cached tags differ from the tags in the server's document.

If-Modified-Since: Date Revalidation

raigترین هدر اعتبار سنجی مجدد If-Modified-Since Cache است. درخواست‌های اعتبار سنجی مجدد If-Modified-Since اغلب درخواست‌های "IMS" نامیده می‌شوند. درخواست‌های IMS به سرور دستور می‌دهد که درخواست را تنها در صورتی انجام دهد که منبع از تاریخ خاصی تغییر کرده باشد:

اگر سند از تاریخ مشخص شده اصلاح شده باشد، شرط If-Modified-Since درست است و GET به طور معمول موفق می‌شود. سند جدید به همراه هدرهای جدید حاوی تاریخ انقضای جدید به Cache بازگردانده می‌شود.

اگر سند از تاریخ مشخص شده اصلاح نشده باشد، شرط نادرست است و یک پیام پاسخ کوچک 304 Not Modified به کلاینت، بدون بدنہ سند، برگردانده می‌شود. هدرها در پاسخ برگردانده می‌شوند و





با این حال، تنها هدرهایی که نیاز به بروزرسانی از نسخه اصلی دارند باید برگردانده شوند. برای مثال، هدر Content-Type معمولاً نیازی به ارسال ندارد، زیرا معمولاً تغییری نکرده است. معمولاً یک تاریخ انقضا جدید است که ارسال می‌شود.

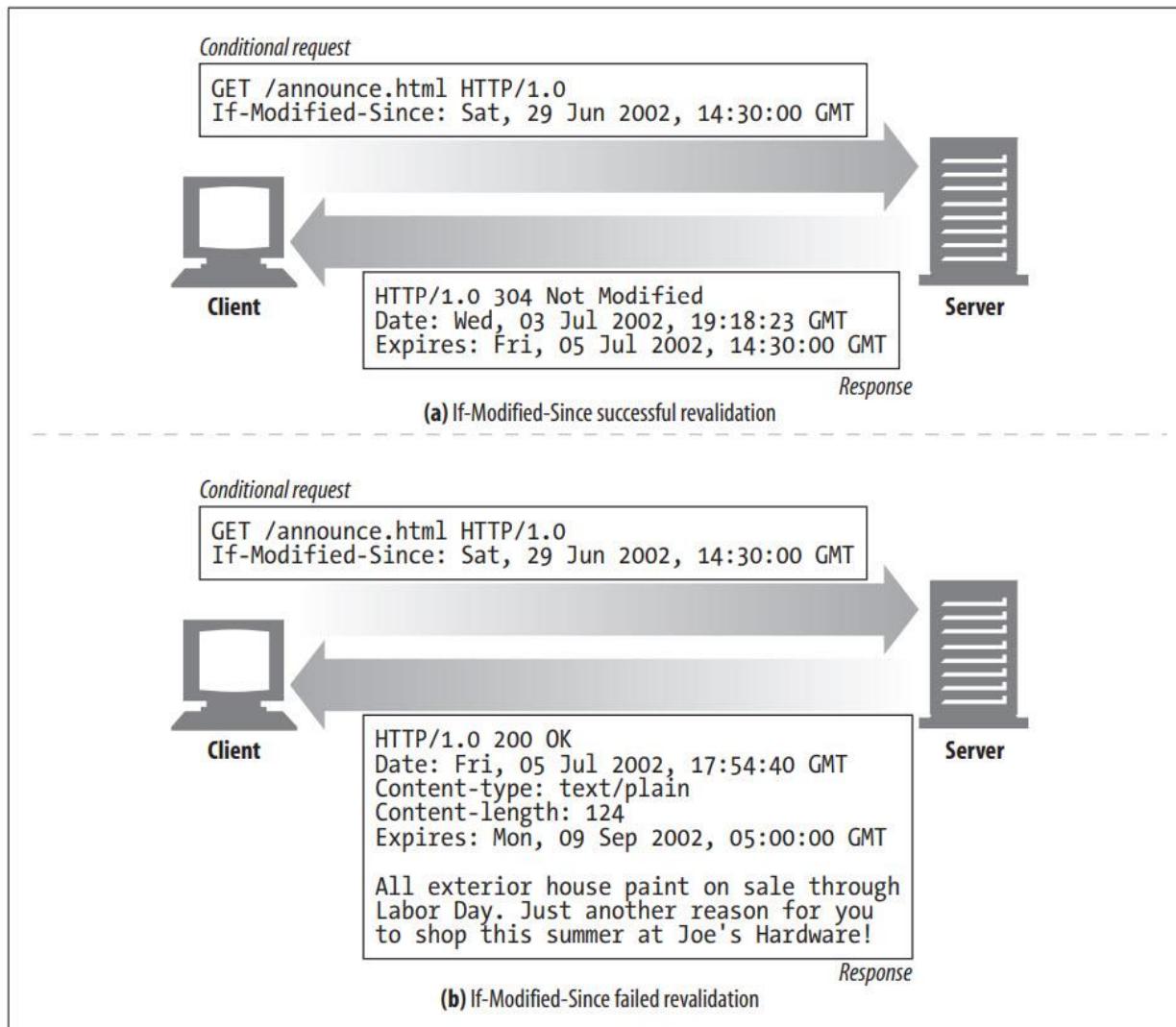
هدر If-Modified-Since در ارتباط با هدر پاسخ سرور Last-Modified کار می‌کند. سرور مبدأ آخرین تاریخ اصلاح را به اسناد ارائه شده پیوست می‌کند. هنگامی که یک Cache می‌خواهد یک سند ذخیره شده را مجددًا اعتبار سنجی کند، شامل یک هدر If-Modified-Since با تاریخ آخرین ویرایش نسخه ذخیره شده در Cache می‌شود:

If-Modified-Since: <cached last-modified date>

اگر در این مدت محتوا تغییر کرده باشد، آخرین تاریخ اصلاح متفاوت خواهد بود و سرور مبدأ سند جدید را پس می‌فرستد. در غیر این صورت، سرور متوجه خواهد شد که تاریخ آخرین اصلاح Cache با تاریخ آخرین ویرایش فعلی سند سرور مطابقت دارد و یک پاسخ ۳۰۴ بدون تغییر را برمی‌گرداند.

به عنوان مثال، همانطور که در شکل زیر نشان داده شده است، اگر Cache شما اعلامیه فروش چهارم ژوئیه Joe's Hardware را در ۳ ژوئیه مجددًا تأیید کند، یک پاسخ Not Modified را دریافت خواهید کرد (بخش Cache a) اما اگر Cache شما سند را پس از پایان فروش در نیمه شب ۵ جولای مجددًا تأیید کند، سند جدیدی دریافت می‌کند، زیرا محتوای سرور تغییر کرده است (بخش b از شکل).





توجه داشته باشید که برخی از وب سرورها If-Modified-Since را به عنوان مقایسه تاریخ واقعی پیاده سازی نمی‌کنند. در عوض، آن‌ها یک تطابق رشته‌ای بین تاریخ IMS و تاریخ آخرین اصلاح انجام می‌دهند. به این ترتیب، در لغت «اگر آخرین بار در این تاریخ اصلاح نشده است» بهجای «اگر از این تاریخ اصلاح شده است» رفتار می‌کند. این معنایی جایگزین برای انقضای Cache، زمانی که از تاریخ آخرین اصلاح به عنوان نوعی شماره سریال استفاده می‌کنید، به خوبی کار می‌کند، اما از استفاده کلاینت‌ها از هدر If-Modified-Since برای اهداف مبتنی بر زمان واقعی جلوگیری می‌کند.

If-None-Match: Entity Tag Revalidation

شرایطی وجود دارد که اعتبار مجدد آخرین تاریخ اصلاح شده کافی نیست:





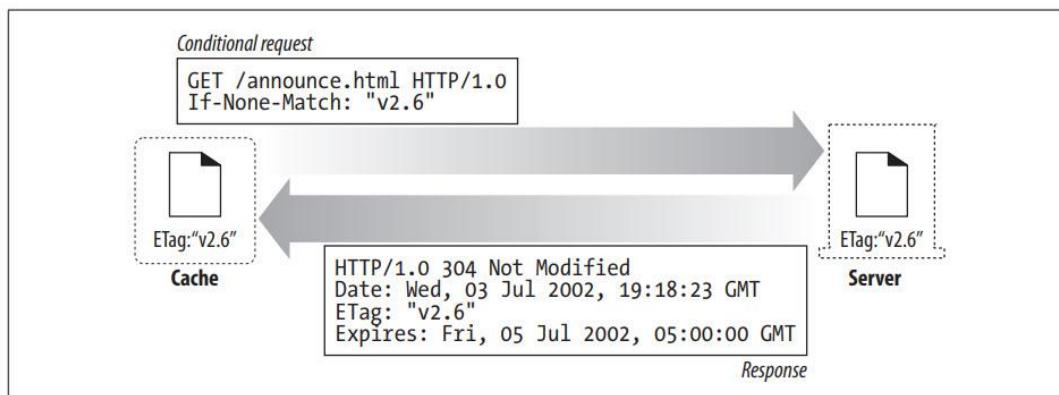
- برخی از اسناد ممکن است به صورت دوره‌ای بازنویسی شوند (به عنوان مثال، از یک پروسس پس زمینه) اما در واقع اغلب حاوی همان داده‌ها هستند. تاریخ‌های اصلاح تغییر خواهد کرد، حتی اگر محتوا تغییر نکرده باشد.

- ممکن است برخی از اسناد تغییر کرده باشند، اما فقط به روش‌هایی که به اندازه کافی مهم نیستند که در سراسر جهان را برای بارگیری مجدد داده‌ها تضمین کند (مانند تغییرات املایی یا کامنت).
- برخی از سرورها نمی‌توانند آخرین تاریخ اصلاح صفحات خود را به درستی تعیین کنند.
- برای سرورهایی که اسنادی را ارائه می‌دهند که در بازه‌های زمانی زیر یک ثانیه تغییر می‌کنند (مثلاً برای جزئیات یک ثانیه‌ای تاریخ‌های اصلاح ممکن است کافی نباشد).

برای دور زدن این مشکلات، HTTP به شما این امکان را می‌دهد که «شناسه‌های نسخه» اسناد به نام Entity Tag یا ETags را با هم مقایسه کنید. Entity Tag ها برچسب‌های دلخواه (رشته‌های نقل قول) هستند که به سند متصل می‌شوند. آن‌ها ممکن است حاوی شماره سریال یا نام نسخه برای سند، یا یک Fingerprint یا Checksum دیگری از محتوای سند باشند.

وقتی ناشر تغییری در سند ایجاد می‌کند، می‌تواند Entity Tag سند را برای نمایش این نسخه جدید تغییر دهد. در صورتی که Entity Tag ها تغییر کرده باشند، Cache ها می‌توانند از هدر شرطی If-None-Match برای دریافت نسخه جدیدی از سند استفاده کنند.

در شکل زیر، Cache دارای سندی با "v2.6" Entity Tag است. فقط در صورتی که تگ "v2.6" دیگر مطابقت نداشته باشد، با سرور مبدا که یک شی جدید را درخواست می‌کند، دوباره اعتبار می‌یابد. در شکل زیر، تگ همچنان مطابقت دارد، بنابراین یک پاسخ 304 Not Modified برگردانده می‌شود.



اگر سرور تغییر کرده بود (شاید به "v3.0")، سرور محتوای جدید را در یک پاسخ 200 OK به همراه محتوا و ETag جدید برمی‌گرداند.





چندین Entity Tag را می‌توان در یک هدر If-None-Match گنجاند تا به سرور بگوید که Cache قبل از اشیاء با آن Entity Tag ها دارد:

If-None-Match: "v2.6"

If-None-Match: "v2.4","v2.5","v2.6"

If-None-Match: "foobar","A34FAC0095","Profiles in Courage"

Weak and Strong Validators

Entity Tag ها از Cache برای تعیین به روز بودن نسخه ذخیره شده با توجه به سرور استفاده می‌کنند (مثل اینکه از تاریخ‌های آخرین ویرایش استفاده می‌کنند). به این ترتیب، Entity Tag ها و آخرین تاریخ‌های اصلاح شده، هر دو اعتبار سنجی Cache هستند.

سرورها ممکن است گاهی بخواهند تغییرات زیبایی یا ناچیز را در اسناد بدون باطل کردن تمام کپی‌های ذخیره شده در Cache مجاز کنند. از HTTP/1.1 «weak validators» پشتیبانی می‌کند، که به سرور اجازه می‌دهد حتی اگر محتویات کمی تغییر کرده باشند، معادل «good enough» را ادعا کنند.

Strong Validator ها هر زمان که محتوا تغییر کند تغییر می‌کنند. Weak Validator اجازه می‌دهد برخی از محتوا تغییر کند، اما به طور کلی زمانی که معنای مهم محتوا تغییر می‌کند، تغییر می‌کند. برخی از عملیات را نمی‌توان با استفاده از Weak Validator انجام داد (مانند واکشی با محدوده جزئی شرطی)، بنابراین سرورها Weak Validator هایی را که هستند با پیشوند "W/" شناسایی می‌کنند:

ETag: W/"v2.6"

If-None-Match: W/"v2.6"

هر زمان که مقدار Entity مرتبه به هر نحوی تغییر کرد، یک Strong Entity Tag باید تغییر کند. یک Weak Entity Tag باید هر زمان که Entity مرتبه به شکل معنی‌داری تغییر کند، تغییر نماید.

توجه داشته باشید که یک سرور مبدأ باید از استفاده مجدد از یک مقدار Strong Entity Tag خاص برای دو موجودیت مختلف یا استفاده مجدد از یک مقدار Weak Entity Tag خاص برای دو موجودیت معنایی متفاوت اجتناب کند. ورودی‌های حافظه پنهان ممکن است برای دوره‌های دلخواه طولانی، بدون توجه به زمان‌های انقضا، باقی بمانند، بنابراین ممکن است انتظار این که یک Cache دیگر هرگز تلاشی برای





تایید یک ورودی با استفاده از اعتبارسنجی‌ای که در نقطه‌ای در گذشته به دست آورده است، نامناسب باشد.

When to Use Entity Tags and Last-Modified Dates

اگر سرور یک Entity Tag را ارسال کند، کلاینت‌های HTTP/1.1 باید از اعتبارسنجی Entity Tag استفاده کنند. اگر سرور فقط یک مقدار Last-Modified را ارسال کند، کلاینت می‌تواند از اعتبارسنجی If-Modified-Since استفاده نماید. اگر هم Entity Tag و تاریخ آخرین اصلاح موجود باشد، کلاینت باید از هر دو طرح اعتبارسنجی مجدد استفاده کند و به Cache‌های HTTP/1.0 و HTTP/1.1 اجازه پاسخ مناسب بدهد.

سرورهای مبدأ HTTP/1.1 باید یک اعتبارسنجی Entity Tag را ارسال کنند، مگر اینکه ایجاد آن امکان پذیر نباشد و اگر اعتبار سنجی ضعیف مزایایی داشته باشد ممکن است به جای Strong Entity Tag، یک Weak Entity Tag مورد استفاده قرار گیرد. همچنین، ترجیح داده می‌شود که یک مقدار last-modified نیز ارسال شود.

اگر یک Entity Tag یا سرور Cache و If-Modified-Since HTTP/1.1 درخواستی با هر دو هدر شرطی دریافت کند، نباید پاسخ 304 Not Modified را برگرداند مگر اینکه این کار با تمام فیلدهای هدر شرطی در درخواست مطابقت داشته باشد.

Controlling Cachability

چندین راه را برای سرور تعریف می‌کند تا مشخص کند یک سند تا چه مدت می‌تواند قبل از انقضا در Cache بماند. به ترتیب اولویت، سرور می‌تواند:

هدر Cache-Control: no-store را به پاسخ ضمیمه کند.

هدر Cache-Control: no-cache را به پاسخ ضمیمه کند.

هدر Cache-Control: must-revalidate را ضمیمه کند.

هدر Cache-Control: max-age را به پاسخ ضمیمه کند.

هدر Expire Date را به پاسخ پیوست کند.

هیچ اطلاعات انقضایی را ضمیمه نکند و به Cache اجازه دهد تاریخ انقضای اکتشافی خود را تعیین کند.





No-Cache and No-Store Response Headers

HTTP/1.1 چندین راه برای محدود کردن Cache کردن اشیاء یا سرویس دادن به اشیاء Cache شده برای حفظ تازگی ارائه می‌دهد. هدرهای no-cache و no-store از سرویس Cache به اشیاء ذخیره نشده تأیید نشده جلوگیری می‌کنند:

`Cache-Control: no-store`

`Cache-Control: no-cache`

`Pragma: no-cache`

پاسخی که با علامت "no-store" مشخص شده است، Cache را از کپی کردن پاسخ منع می‌کند. یک Cache معمولاً یک پاسخ no-store را به کلاینت ارسال می‌کند و سپس شی را حذف می‌کند، همانطور که یک سرور پراکسی غیر Cache انجام می‌دهد.

پاسخی که "no-cache" علامت گذاری شده است، در واقع می‌تواند در Cache محلی ذخیره شود. فقط نمی‌توان آن را از Cache به کلاینت بدون تأیید مجدد تازگی با سرور مبدا ارائه کرد. نام بهتر برای این هدر ممکن است "do-notserve-from-cache-without-revalidation" باشد.

هدر `Pragma: no-cache` در HTTP/1.1 برای سازگاری با HTTP/1.0 + گنجانده شده است. برنامه‌های HTTP 1.0 باید از Cache-Control: no-cache به جز زمانی که با برنامه‌های سروکار دارند که فقط Pragma: no-cache را می‌فهمند.

Max-Age Response Headers

هدر Cache-Control: max-age ت Shan دهنده تعداد ثانیه‌هایی است که از سروری که یک سند را می‌توان تازه در نظر گرفت. همچنین یک هدر s-maxage وجود دارد (به عدم وجود خط فاصله در "maxage" توجه کنید) که مانند max-age عمل می‌کند اما فقط برای Cache های مشترک (عمومی) اعمال می‌شود:

`Cache-Control: max-age=3600`

`Cache-Control: s-maxage=3600`

سرورها می‌توانند درخواست کنند که Cache ها یا سندی را در Cache ذخیره نکنند یا در هر دسترسی با تنظیم maximum age روی صفر، آن را به روز کنند:

`Cache-Control: max-age=0`





Cache-Control: s-maxage=0

Expires Response Headers

هدر منسخ شده Expires به جای زمان در ثانیه، تاریخ انقضای واقعی را مشخص می‌کند. طراحان HTTP بعداً به این نتیجه رسیدند که، چون بسیاری از سرورها ساعتهاي ناهمگام يا نادرست دارند، بهتر است انقضا را در ثانیه‌های سپری شده نشان دهند تا زمان مطلق. با محاسبه تعداد ثانیه‌های اختلاف بین مقدار انقضا و مقدار تاریخ، می‌توان طول عمر تازه بودن مشابهی را محاسبه کرد:

Expires: Fri, 05 Jul 2002, 05:00:00 GMT

برخی از سرورها نیز یک هدر پاسخ 0: Expires را ارسال می‌کنند تا سعی کنند اسناد همیشه منقضی شوند، اما این نحو غیرقانونی است و می‌تواند برای برخی از نرم افزارها مشکلاتی ایجاد کند. شما باید سعی کنید از این ساختار به عنوان ورودی پشتیبانی کنید، اما نباید آن را تولید کنید.

Must-Revalidate Response Headers

ممکن است برای خدمت به اشیاء قدیمی (منقضی شده) به منظور بهبود عملکرد پیکربندی شود. اگر سرور مبدأ بخواهد Cache ها به طور دقیق به اطلاعات انقضا پایبند باشند، می‌تواند یک Cache-Control را ضمیمه کند:

Cache-Control: must-revalidate

هدر پاسخ Cache ها می‌گوید که نمی‌تواند یک کپی قدیمی از این شی را بدون تأیید مجدد با سرور مبدأ ارائه کنند. Cache ها هنوز برای ارائه نسخه‌های جدید آزاد هستند. اگر سرور مبدأ در دسترس نباشد و قتی Cache تلاش می‌کند بررسی تازه‌بودن را مجدد تأیید کند، Cache باید خطای 504 Gateway Timeout را برگرداند.

Heuristic Expiration

اگر پاسخ شامل هدر Cache-Control: max-age یا هدر Expires ممکن است Maximum Age اکتشافی را محاسبه کند. ممکن است از هر الگوریتمی استفاده شود، اما اگر حداقل سن به دست آمده بیشتر از ۲۴ ساعت باشد، یک هدر Warning 13 (Heuristic Expiration Warning) باید به هدرهای پاسخ اضافه شود. تا آنجا که می‌دانیم، تعداد کمی از مرورگرها این اطلاعات هشدار را در اختیار کاربران قرار می‌دهند.





یک الگوریتم انقضای اکتشافی محبوب، الگوریتم LM-Factor است که اگر سند حاوی تاریخ last-modified باشد، می‌تواند مورد استفاده قرار گیرد. الگوریتم last-modified از تاریخ LM-Factor به عنوان تخمین میزان فرار یک سند استفاده می‌کند. این منطق است:

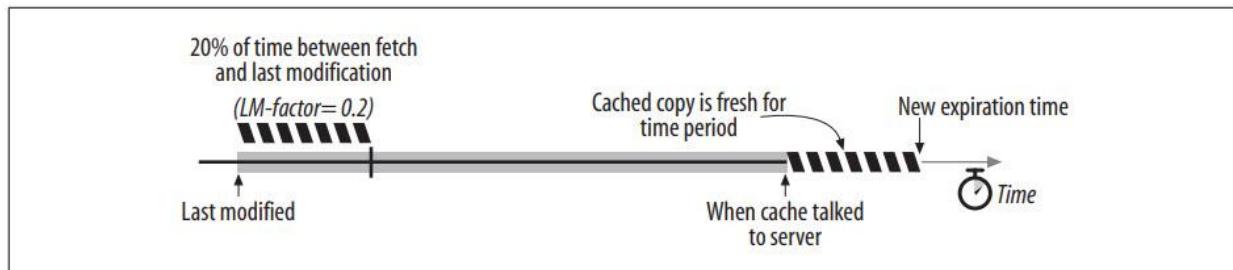
- اگر یک سند ذخیره شده آخرین بار در گذشته‌های دور تغییر کرده باشد، ممکن است سندی پایدار باشد و احتمال تغییر ناگهانی آن کمتر باشد، بنابراین نگهداری آن برای مدت طولانی در Cache امن‌تر است.
- اگر سند ذخیره شده اخیراً اصلاح شده باشد، احتمالاً مرتباً تغییر می‌کند، بنابراین باید فقط مدت کوتاهی قبل از تأیید مجدد با سرور، آن را در Cache ذخیره کنیم.

الگوریتم واقعی LM-Factor زمان بین زمانی که Cache با سرور صحبت می‌کند تا زمانی که سرور می‌گوید سند برای آخرین بار اصلاح شده است را محاسبه می‌کند، کسری از این زمان میانی را می‌گیرد و از این کسری به عنوان مدت زمان تازه‌سازی در Cache استفاده می‌کند. در اینجا تعدادی شبه کد پرل برای الگوریتم عامل LM آمده است:

```
$time_since_modify = max(0, $server_Date - $server_Last_Modified);
```

```
$server_freshness_limit = int($time_since_modify * $lm_factor);
```

شکل زیر دوره تازگی LM-factor را به صورت گرافیکی نشان می‌دهد. خط متقطع نشان دهنده دوره تازگی با استفاده از ضریب 0.2 است.



به طور معمول، افراد محدوده‌های بالایی را در دوره‌های تازگی اکتشافی قرار می‌دهند تا نتوانند بیش از حد بزرگ شوند. یک هفته معمولی است، اگرچه سایتها محفظه کارتر از یک روز استفاده می‌کنند.

در نهایت، اگر تاریخ last-modified Cache را نیز ندارید، اطلاعات زیادی برای ادامه ندارد. Cache معمولاً یک دوره پیش‌فرض تازگی (یک ساعت یا یک روز معمولی است) را برای اسناد بدون هیچ سرنخ تازه‌سازی اختصاص می‌دهد. Cache محفظه کارانه‌تر گاهی اوقات طول عمر تازگی صفر را برای این اسناد اکتشافی





انتخاب می‌کنند، و Cache را مجبور می‌کند تا قبل از هر بار ارائه به کلاینت، تأیید کند که داده‌ها هنوز تازه هستند.

بسیاری از سرورهای مبدا هنوز هدرهای Expire و Max-age را تولید نمی‌کنند. پیش‌فرضهای انقضای Cache خود را با دقت انتخاب کنید!

Client Freshness Constraints

مرورگرهای وب دارای یک دکمه Refresh یا Reload برای به روزرسانی اجباری محتوا هستند که ممکن است در Cache مرورگر یا پراکسی باشد. دکمه Refresh یک درخواست GET را با هدرهای درخواست کنترل اضافی صادر می‌کند که اعتبار مجدد یا واکشی بدون قید و شرط از سرور را فراهم می‌کند. رفتار دقیق Refresh به مرورگر خاص، سند و پیکربندی‌های Cache بستگی دارد.

کلاینت‌ها از هدرهای درخواست Cache-Control برای ثبتیت یا کاهش محدودیت‌های انقضا استفاده می‌کنند. برای برنامه‌هایی که به جدیدترین اسناد (مانند دکمه Refresh دستی) نیاز دارند، کلاینت‌ها می‌توانند از هدرهای Cache-Control استفاده کنند تا انقضا را سخت‌تر کنند. از سوی دیگر، کلاینت‌ها ممکن است بخواهند نیازهای تازگی را به عنوان مصالحه‌ای برای بهبود عملکرد، قابلیت اطمینان یا هزینه‌ها کاهش دهند. جدول زیر دستورالعمل‌های درخواست Cache-Control را به صورت خلاصه نشان می‌دهد.

Directive	Purpose
Cache-Control: max-stale	The cache is free to serve a stale document. If the <s> parameter is specified, the document must not be stale by more than this amount of time. This relaxes the caching rules.
Cache-Control: max-stale = <s>	
Cache-Control: min-fresh = <s>	The document must still be fresh for at least <s> seconds in the future. This makes the caching rules more strict.
Cache-Control: max-age = <s>	The cache cannot return a document that has been cached for longer than <s> seconds. This directive makes the caching rules more strict, unless the max-stale directive also is set, in which case the age can exceed its expiration time.
Cache-Control: no-cache Pragma: no-cache	This client won't accept a cached resource unless it has been revalidated.
Cache-Control: no-store	The cache should delete every trace of the document from storage as soon as possible, because it might contain sensitive information.
Cache-Control: only-if-cached	The client wants a copy only if it is in the cache.





Cautions

انقضای سند یک سیستم کامل نیست. اگر ناشر به طور تصادفی تاریخ انقضا را در آینده خیلی دور تعیین کند، هر گونه تغییر سندی که باید انجام دهد، لزوماً تا زمانی که سند منقضی نشده باشد، در تمام Cache ها نمایش داده نمی‌شود. به همین دلیل بسیاری از ناشران از تاریخ انقضای دور استفاده نمی‌کنند. همچنین، بسیاری از ناشران حتی از تاریخ انقضا نیز استفاده نمی‌کنند و این امر باعث می‌شود که Cache ها بدانند تا چه زمانی سند تازه است.

Setting Cache Controls

وب سرورهای مختلف مکانیسم‌های متفاوتی را برای تنظیم هدرهای HTTP Cache-Control و انقضا ارائه می‌دهند. در این بخش، به طور خلاصه در مورد نحوه پشتیبانی وب سرور محبوب آپاچی از کنترل‌های Cache صحبت خواهیم کرد.

Controlling HTTP Headers with Apache

وب سرور آپاچی مکانیزم‌های مختلفی را برای تنظیم هدرهای کنترل حافظه پنهان HTTP فراهم می‌کند. بسیاری از این مکانیسم‌ها به طور پیش‌فرض فعال نیستند—شما باید آن‌ها را فعال کنید (در برخی موارد ابتدا باید Apache Extension Modules می‌پردازیم):

mod_headers

ماژول mod_headers به شما امکان می‌دهد هدرهای جداگانه را تنظیم کنید. هنگامی که این ماژول بارگذاری شد، می‌توانید فایل‌های پیکربندی آپاچی را با دستور العمل‌هایی برای تنظیم هدرهای تکی HTTP تقویت کنید. همچنین می‌توانید از این تنظیمات در ترکیب با عبارات منظم و فیلترهای آپاچی برای مرتبط کردن هدرها با محتوای فردی استفاده کنید. در اینجا نمونه‌ای از پیکربندی نشان داده شده است که می‌تواند تمام فایل‌های HTML موجود در یک دایرکتوری را به عنوان uncacheable علامت گذاری کند:

```
<Files *.html>
```

```
Header set Cache-control no-cache
```

```
</Files>
```

mod_expires





ماژول mod_expires منطق برنامه را برای تولید خودکار هدرهای Expires با تاریخ انقضا صحیح ارائه می‌دهد. این ماژول به شما امکان می‌دهد تاریخ انقضا را برای مدتی پس از آخرین دسترسی به سند یا پس از آخرین تاریخ اصلاح آن تنظیم کنید. این ماژول همچنین به شما امکان می‌دهد تاریخ‌های انقضای مختلفی را به انواع فایل‌های مختلف اختصاص دهید و از توصیف‌های پرمخاطب راحت، مانند Access Plus 1 Month یا دسترسی به اضافه ۱ ماه برای توصیف قابلیت ذخیره‌سازی در Cache استفاده کنید. در اینجا چند نمونه هستند:

ExpiresDefault A3600

ExpiresDefault M86400

ExpiresDefault "access plus 1 week"

ExpiresByType text/html "modification plus 2 days 6 hours 12 minutes"

mod_cern_meta

ماژول mod_cern_meta به شما اجازه می‌دهد تا فایلی از هدرهای HTTP را با اشیاء خاصی مرتبط کنید. وقتی این ماژول را فعال می‌کنید، مجموعه‌ای از Metafile‌ها ایجاد می‌کنید، یکی برای هر سندی که می‌خواهد کنترل کنید انتخاب نموده و هدرهای دلخواه را به هر Metafile اضافه می‌کنید.

Controlling HTML Caching Through HTTP-EQUIV

هدرهای پاسخ سرور HTTP برای انتقال اطلاعات انقضای سند و کنترل Cache استفاده می‌شود. وب سرورها با فایل‌های پیکربندی تعامل دارند تا هدرهای cache-control صحیح را به استناد ارائه شده اختصاص دهند.

برای آسان‌تر کردن تخصیص اطلاعات هدر HTTP به استناد HTML بدون تعامل با فایل‌های پیکربندی وب سرور، HTML 2.0 تگ <META HTTP-EQUIV> را تعریف کرد. این تگ اختیاری در بالای یک سند HTML قرار می‌گیرد و هدرهای HTTP را که باید با سند مرتبط باشند تعریف می‌کند. در اینجا نمونه‌ای از تگ <META> برای علامت گذاری سند HTML غیر قابل ذخیره سازی وجود دارد:

<HTML>

<HEAD>

<TITLE>My Document</TITLE>

<META HTTP-EQUIV="Cache-control" CONTENT="no-cache">

</HEAD>





.....

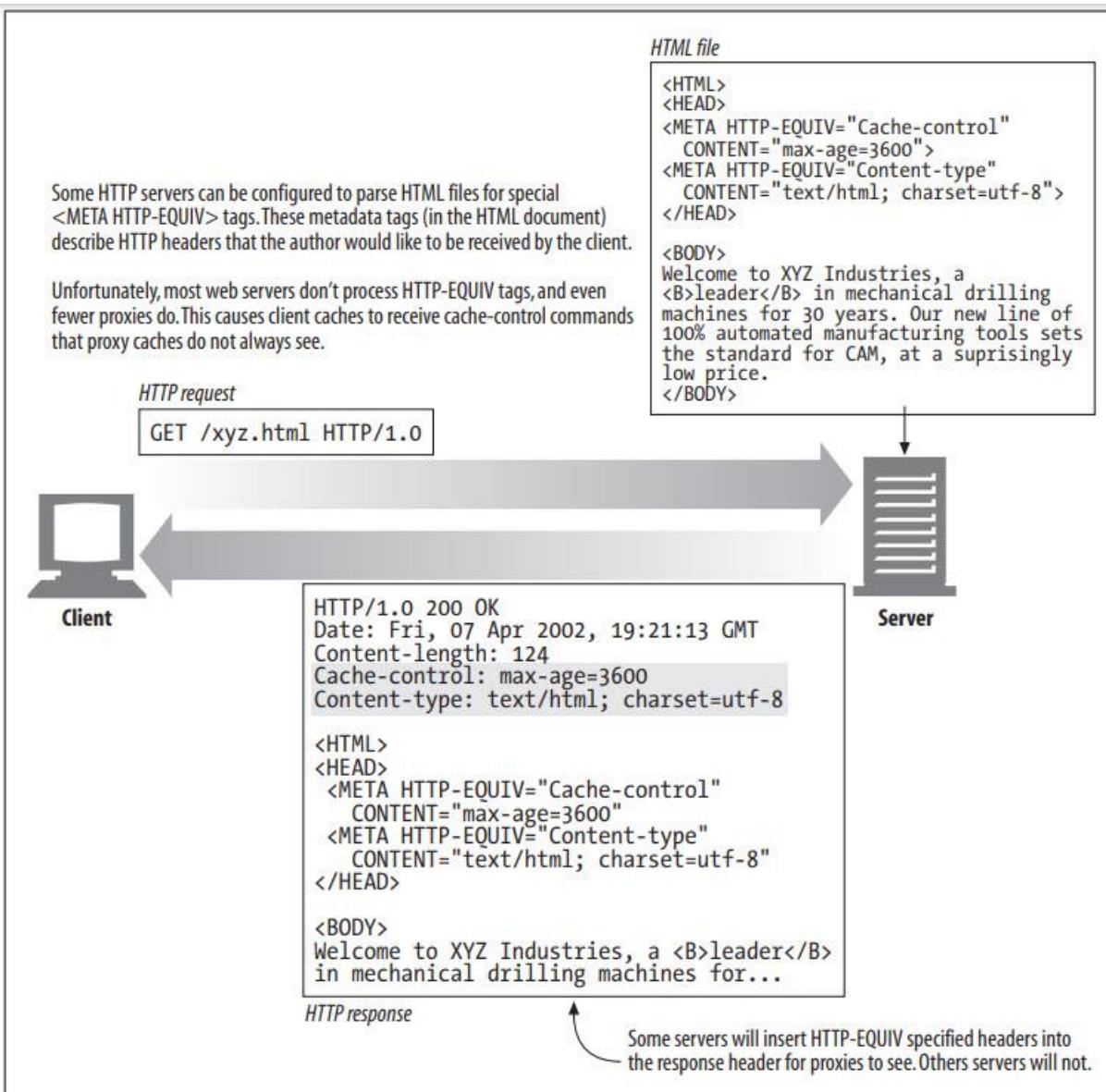
این تگ HTTP-EQUIV در ابتدا برای استفاده توسط سرورهای وب در نظر گرفته شده بود. قرار بود سرورهای HTTP را برای تگهای <META HTTP-EQUIV> تجزیه کنند و هدرهای تجویز شده را در پاسخ HTML مستند شده است: وارد کنند، همانطور که در HTML RFC 1866 مستند شده است:

یک سرور HTTP ممکن است از این اطلاعات برای پردازش سند استفاده کند. به طور خاص، ممکن است شامل یک فیلد هدر در پاسخ به درخواست‌های این سند باشد: نام هدر از مقدار ویژگی HTTP-EQUIV و مقدار هدر از مقدار ویژگی CONTENT گرفته شده است.

متأسفانه، تعداد کمی از وب سرورها و پروکسی‌ها از این ویژگی اختیاری پشتیبانی می‌کنند. ایجاد بار اضافی سرور، مقادیر ثابت و پشتیبانی تنها از HTML برخی از دلایل این موضوع است.

با این حال، برخی از مرورگرها تگهای HTML را در محتوای HTTP-EQUIV تجزیه می‌کنند و به آن‌ها پاییند هستند و با هدرهای تعبیه‌شده مانند هدرهای HTTP واقعی رفتار می‌کنند (شکل زیر). این مایه تاسف است، زیرا مرورگرهای HTML که از HTTP-EQUIV پشتیبانی می‌کنند ممکن است قوانین cache-control متفاوتی را نسبت به Intervening Proxy Cache می‌شود.





به طور کلی، تگ‌های **<META HTTP-EQUIV>** روش ضعیفی برای کنترل قابلیت ذخیره‌سازی اسناد هستند. تنها راه مطمئن برای برقراری ارتباط با درخواست‌های **cache-control** برای اسناد، از طریق هدرهای HTTP است که توسط یک سرور به درستی پیکربندی شده ارسال می‌شود.

Detailed Algorithms

مشخصات HTTP یک الگوریتم دقیق، اما کمی مبهم و اغلب گیج کننده برای محاسبه قدیمی بودن اسناد و تازگی Cache ارائه می‌دهد. در این بخش، الگوریتم‌های محاسبه تازگی HTTP را مورد بحث قرار می‌دهیم و انگیزه پشت آن‌ها را توضیح می‌دهیم.





این بخش برای خوانندگانی که با حافظه‌های داخلی کار می‌کنند بسیار مفید خواهد بود. برای کمک به توضیح عبارت در مشخصات HTTP، از شبهه کد پرل استفاده خواهیم کرد. اگر به جزئیات ناخوشایند فرمول‌های انقضای علاقه‌ای ندارید، از این بخش صرفنظر کنید.

Age and Freshness Lifetime

برای تشخیص اینکه آیا یک سند ذخیره شده در Cache به اندازه کافی تازه است یا خیر، یک Cache تنها باید دو مقدار را محاسبه کند: سن کپی ذخیره شده و طول عمر تازه بودن نسخه ذخیره شده در Cache. اگر سن یک کپی ذخیره شده کمتر از طول عمر آن باشد، کپی به اندازه کافی تازه است که بتواند ارائه شود. در پرل:

```
$is_fresh_enough = ($age < $freshness_lifetime);
```

سن سند کل زمانی است که سند از زمانی که از سرور ارسال شده است (یا آخرین بار توسط سرور اعتبارسنجی شده است) "پیر" شده است. یک سرور، نمی‌تواند فرض کند که سند کاملاً جدید است. باید سن سند از طریق هدر Age صریح (ترجیح داده شده) یا با پردازش هدر Date تولید شده توسط سرور مشخص شود.

برخی از کلاینت‌ها ممکن است مایل به پذیرش اسناد کمی قدیمی باشند (با استفاده از هدر Cache-Control: max-stale). سایر کلاینت‌ها ممکن است اسنادی را که در آینده نزدیک کهنه می‌شوند (با استفاده از Cache-Control: min-fresh header) نپذیرند. اطلاعات انقضای سرور را با نیازهای تازه بودن کلاینت ترکیب می‌کند تا حداکثر طول عمر تازگی را تعیین کند.

Age Computation

سن پاسخ، کل زمان از زمان صدور پاسخ از سرور (یا تأیید مجدد از سرور) است. این سن شامل زمانی است که پاسخ در مسیریاب‌ها و دروازه‌های اینترنت، زمان ذخیره شده در Cache میانی و زمانی که پاسخ در شما باقی مانده است. مثال زیر شبهه کدی را برای محاسبه سن ارائه می‌دهد.





```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$response_delay_estimate = ($time_got_response - $time_issued_request);
$age_when_document_arrived_at_our_cache =
    $corrected_apparent_age + $response_delay_estimate;
$how_long_copy_has_been_in_our_cache = $current_time - $time_got_response;

$age = $age_when_document_arrived_at_our_cache +
    $how_long_copy_has_been_in_our_cache;
```

جزئیات محاسبه سن HTTP کمی پیچیده است، اما مفهوم اصلی ساده است. Cache ها می‌توانند با بررسی هدرهای Date یا Age نشان دهند که پاسخ در زمان رسیدن به Cache چند ساله بوده است. Cache ها همچنین می‌توانند مدت زمانی را که سند در Cache محلی باقی مانده است، یادداشت کنند. در مجموع، این مقادیر کل سن پاسخ هستند. HTTP برای جبران انحراف ساعت و تأخیرهای شبکه موارد خاصی را ایجاد می‌کند، اما محاسبات اولیه به اندازه کافی ساده است:

```
$age = $age_when_document_arrived_at_our_cache +
    $how_long_copy_has_been_in_our_cache;
```

یک Cache می‌تواند به راحتی تعیین کند که یک کپی ذخیره شده در Cache چه مدت به صورت محلی ذخیره شده است (یک موضوع ساده حسابداری)، اما تعیین سن پاسخ زمانی که به Cache می‌رسد دشوارتر است، زیرا همه سرورها ساعتهای همگام سازی شده ندارند و همچنین ما نمی‌دانیم پاسخ کجا بوده است. الگوریتم کامل محاسبه سن سعی در رفع این مشکل دارد.

Apparent age is based on the Date header

اگر همه رایانه‌ها ساعت یکسان و دقیقاً درست را به اشتراک بگذارند، سن یک سند ذخیره شده در Cache به سادگی «سن ظاهری» سند خواهد بود – زمان فعلی منهای زمانی که سرور سند را ارسال کرده است. زمان ارسال سرور به سادگی مقدار هدر Date است. ساده‌ترین محاسبه سن اولیه فقط از سن ظاهری استفاده می‌کند:

```
$apparent_age = $time_got_response - $Date_header_value;
$age_when_document_arrived_at_our_cache = $apparent_age;
```

متسفانه، همه ساعتها به خوبی هماهنگ نیستند. ساعتهای کلاینت و سرور ممکن است چندین دقیقه یا ساعتها باهم متفاوت بوده و یا روزهایی آن‌ها نیز به درستی تنظیم نشده باشد.





برنامه‌های کاربردی وب، به ویژه Cache پراکسی‌ها، باید برای تعامل با سرورهایی با مقادیر ساعت بسیار متفاوت آماده شوند. این مشکل انحراف ساعت نامیده می‌شود که تفاوت بین تنظیمات ساعت دو رایانه است. به دلیل انحراف ساعت، سن ظاهربهی گاهی نادرست و گاهی منفی است.

اگر سن همیشه منفی باشد، آن را روی صفر قرار می‌دهیم. همچنین می‌توانیم بررسی کنیم که سن ظاهربهی به‌طور خاصی بزرگ نیست، اما سن‌های ظاهربهی بزرگ ممکن است در واقع درست باشند. ممکن است با یک Parent Date Cache صحبت کنیم که سند را برای مدت طولانی در Cache ذخیره کرده است (هدر اصلی Cache را نیز ذخیره می‌کند):

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$age_when_document_arrived_at_our_cache = $apparent_age;
```

توجه داشته باشید که هدر Date تاریخ اصلی سرور مبدأ را توصیف می‌کند. پراکسی‌ها و Cache ها نباید این تاریخ را تغییر دهند!

Hop-by-hop age calculations

بنابراین، می‌توانیم سن‌های منفی ناشی از انحراف ساعت را حذف کنیم، اما نمی‌توانیم در مورد کاهش دقت کلی به دلیل انحراف ساعت کاری انجام دهیم. HTTP/1.1 با درخواست از هر دستگاه برای انباستن پیری نسبی در هدر Age، زمانی که یک سند از میان پراکسی‌ها و Cache ها عبور می‌کند، تلاش می‌کند تا کمبود ساعت‌های همگام جهانی را برطرف کند. به این ترتیب، نیازی به مقایسه بین سرور و ساعت end-to-end نیست.

با عبور سند از پروکسی‌ها، مقدار هدر Age افزایش می‌یابد. برنامه‌های کاربردی HTTP/1.1 باید مقدار هدر Age را تا زمانی که سند در هر برنامه و در انتقال شبکه قرار می‌گیرد، افزایش دهند. هر برنامه میانی می‌تواند به راحتی زمان اقامت سند را با استفاده از ساعت محلی خود محاسبه کند.

با این حال، هر دستگاه غیر HTTP/1.1 در زنجیره پاسخ، هدر Age را تشخیص نمی‌دهد و هدر را بدون تغییر پروکسی نموده یا آن را حذف می‌کند. بنابراین، تا زمانی که HTTP/1.1 به طور جهانی پذیرفته نشود، هدر Age نسبت به سن نسبی دست کم گرفته می‌شود.

مقادیر سن نسبی علاوه بر محاسبه سن مبتنی بر تاریخ استفاده می‌شود و محافظه‌کارانه‌ترین تخمین سنی انتخاب می‌شود، زیرا ممکن است مقدار تاریخ متقاطع یا مقدار محاسبه شده با سن کمتر برآورد شود (محافظه‌کارانه‌ترین آن‌ها مسن ترین سن است). به این ترتیب، HTTP خطاهای Age را در هدرهای HTTP نیز تحمل می‌کند، در حالی که در سمت محتوای تازه‌تر اشتباه می‌کند:





```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$age_when_document_arrived_at_our_cache = $corrected_apparent_age;
```

Compensating for network delays

تراکنش‌ها ممکن است کند باشند. این انگیزه اصلی برای ذخیره سازی است. اما برای شبکه‌های بسیار کند یا سرورهای پربار، اگر اسناد برای مدت طولانی در ترافیک شبکه یا سرور گیر کرده باشند، ممکن است محاسبه سن نسبی به طور قابل توجهی سن اسناد را دست کم بگیرد.

هدر Date نشان می‌دهد که سند چه زمانی از سرور مبدأ خارج شده است، اما نمی‌گوید چه مدت زمانی را که سند در مسیر انتقال به Cache سپری کرده است. اگر سند از طریق یک زنجیره طولانی از پراکسی‌ها و Cache‌ها ارائه شود، تاخیر شبکه ممکن است قابل توجه باشد.

هیچ راه آسانی برای اندازه گیری تاخیر شبکه یک طرفه از سرور به Cache وجود ندارد، اما اندازه گیری تاخیر رفت و برگشت آسان‌تر است. یک Cache می‌داند که چه زمانی سند را درخواست کرده و چه زمانی وارد شده است. HTTP/1.1 این تاخیرهای شبکه را با اضافه کردن کل تاخیر رفت و برگشت به طور محافظه کارانه تصحیح می‌کند. این تأخیر Cache به سرور، تخمینی بیش از حد از تأخیر سرور به Cache است، اما محافظه کارانه است. اگر اشتباه باشد، فقط اسناد را قدیمی‌تر از آنچه هستند نشان می‌دهد و باعث اعتبار مجدد غیرضروری می‌شود. در اینجا نحوه محاسبه قرار داده شده است:

```
$apparent_age = max(0, $time_got_response - $Date_header_value);
$corrected_apparent_age = max($apparent_age, $Age_header_value);
$response_delay_estimate = ($time_got_response - $time_issued_request);
$age_when_document_arrived_at_our_cache =
    $corrected_apparent_age + $response_delay_estimate;
```

Complete Age-Calculation Algorithm

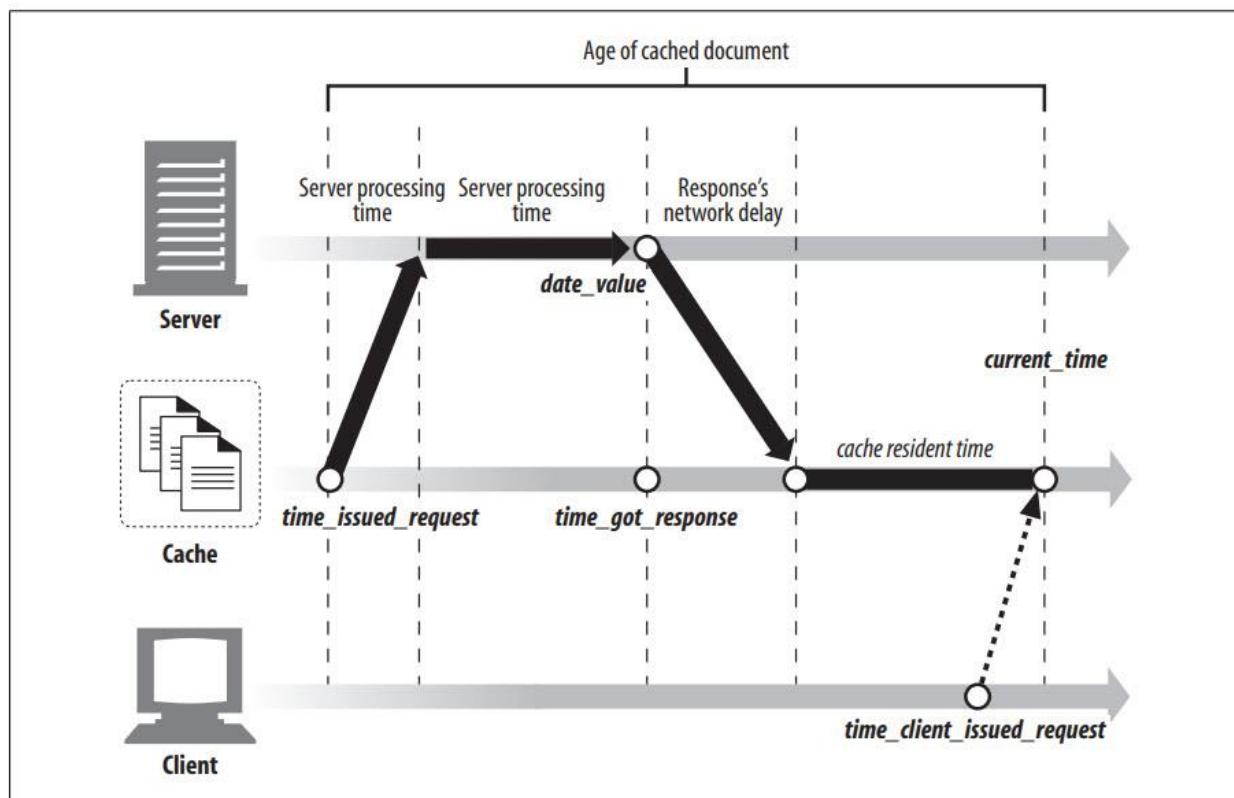
بخش آخر نشان داد که چگونه می‌توان سن یک سند حامل HTTP را هنگام رسیدن به Cache محاسبه کرد. هنگامی که این پاسخ در Cache ذخیره می‌شود، پیتر می‌شود. هنگامی که درخواستی برای سند در Cache می‌رسد، باید بدانیم که سند چه مدت در Cache است، بنابراین می‌توانیم سن سند فعلی را محاسبه کنیم:





```
$age = $age_when_document_arrived_at_our_cache +
      $how_long_copy_has_been_in_our_cache;
```

این الگوریتم کامل محاسبه سنی HTTP/1.1 را که در مثال اول ارائه کردیم به ما می‌دهد. این یک موضوع حسابداری ساده است - ما می‌دانیم چه زمانی سند به Cache رسیده است (`$time_got_response`) و می‌دانیم که درخواست فعلی چه زمانی رسیده است (در حال حاضر)، بنابراین زمان اقامت فقط تفاوت است. همه این‌ها به صورت گرافیکی در شکل زیر نشان داده شده است.



Freshness Lifetime Computation

به یاد داشته باشید که ما در تلاشیم تا بفهمیم که آیا یک سند ذخیره شده به اندازه کافی تازه است که به یک کلاینت ارائه شود یا خیر. برای پاسخ به این سوال، باید سن سند ذخیره شده را تعیین کنیم و طول عمر تازگی را بر اساس محدودیت‌های سرور و کلاینت محاسبه کنیم. ما فقط نحوه محاسبه سن را توضیح دادیم. حالا بباید به عمر تازگی برویم.

طول عمر یک سند نشان می‌دهد که سند مجاز است قبل از اینکه دیگر به اندازه کافی تازه نباشد تا به کلاینت خاصی ارائه شود، چقدر قدیمی شود. طول عمر تازگی به محدودیت‌های سرور و کلاینت بستگی دارد. سرور ممکن است اطلاعاتی در مورد نرخ تغییر انتشار سند داشته باشد. گزارش‌های بسیار پایدار،





ممکن است سال‌ها تازه بمانند. ممکن است نشریات ادواری فقط برای مدت زمان باقیمانده تا انتشار برنامه ریزی شده بعدی (هفته آینده یا ساعت ۶ صبح فردا) به روز باشند.

کلاینت‌ها ممکن است دستورالعمل‌های خاصی داشته باشند. آن‌ها ممکن است مایل به پذیرش محتوای کمی قدیمی باشند، اگر سریعتر باشد، یا ممکن است به بروزترین محتوای ممکن نیاز داشته باشند. Cache‌ها در خدمت کاربران هستند. ما باید به خواسته‌های آن‌ها پایبند باشیم.

Complete Server-Freshness Algorithm

مثال زیر یک الگوریتم پرل را برای محاسبه محدودیت‌های تازگی سرور نشان می‌دهد. این کد حداقل سنی که یک سند می‌تواند به آن برسد و همچنان توسط سرور ارائه شود را برمی‌گرداند.





```
sub server_freshness_limit
{
    local($heuristic,$server_freshness_limit,$time_since_last_modify);

    $heuristic = 0;

    if ($Max_Age_value_set)
    {
        $server_freshness_limit = $Max_Age_value;
    }
    elsif ($Expires_value_set)
    {
        $server_freshness_limit = $Expires_value - $Date_value;
    }
    elsif ($Last_Modified_value_set)
    {
        $time_since_last_modify = max(0, $Date_value - $Last_Modified_value);
        $server_freshness_limit = int($time_since_last_modify * $lm_factor);
        $heuristic = 1;
    }
    else
    {
        $server_freshness_limit = $default_cache_min_lifetime;
        $heuristic = 1;
    }

    if ($heuristic)
    {
        if ($server_freshness_limit > $default_cache_max_lifetime)
        { $server_freshness_limit = $default_cache_max_lifetime; }
        if ($server_freshness_limit < $default_cache_min_lifetime)
        { $server_freshness_limit = $default_cache_min_lifetime; }
    }
}

return($server_freshness_limit);
}
```

اکنون بباید ببینیم که چگونه کلاینت می‌تواند محدودیت سنی تعیین شده توسط سرور سند را لغو کند. مثال زیر یک الگوریتم پرل را نشان می‌دهد که محدودیت تازگی سرور را در نظر می‌گیرد و آن را با محدودیتهای کلاینت تغییر می‌دهد. حداکثر سنی را که یک سند می‌تواند به آن برسد و همچنان توسط Cache بدون اعتبار سنجی مجدد ارائه شود، را برمی‌گرداند.





```
sub client_modified_freshness_limit
{
    $age_limit = server_freshness_limit();    ## From Example 7-2

    if ($Max_Stale_value_set)
    {
        if ($Max_Stale_value == $INT_MAX)
        { $age_limit = $INT_MAX; }
        else
        { $age_limit = server_freshness_limit() + $Max_Stale_value; }
    }

    if ($Min_Fresh_value_set)
    {
        $age_limit = min($age_limit, server_freshness_limit() - $Min_Fresh_value_set);
    }

    if ($Max_Age_value_set)
    {
        $age_limit = min($age_limit, $Max_Age_value);
    }
}
```

کل فرآیند شامل دو متغیر است: سن سند و حد تازگی آن.

اگر سن کمتر از حد تازگی باشد، سند "fresh enough" یا به اندازه کافی تازه است. الگوریتم موجود در مثال بالا فقط محدودیت تازگی سرور را می‌گیرد و آن را بر اساس محدودیتهای کلاینت اضافی به اطراف می‌کشاند. امیدواریم این بخش الگوریتم‌های انقضایی ظریفی را که در مشخصات HTTP توضیح داده شده اند کمی واضح‌تر کنند.

Caches and Advertising

اگر تا اینجا که پیش رفته‌اید، متوجه شدید که Cahce ها عملکرد را ببینند و ترافیک را کاهش می‌دهند. می‌دانید که Cahce می‌تواند به کاربران کمک کند و تجربه بهتری به آن‌ها بدهد، و می‌دانید که Cahce می‌تواند به اپراتورهای شبکه کمک کند تا ترافیک خود را کاهش دهند.

The Advertiser's Dilemma

همچنین ممکن است انتظار داشته باشید که ارائه دهنده‌گان محتوا Cahce را دوست داشته باشند. به هر حال، اگر Cahce همه جا وجود داشت، ارائه دهنده‌گان محتوا مجبور نبودند سرورهای وب چند پردازنده‌ای بزرگ بخوبند تا با تقاضا مطابقت داشته باشند و مجبور نخواهند بود هزینه‌های سنگین خدمات شبکه را پردازنند تا داده‌های یکسانی را بارها و بارها به بینندگان خود ارائه دهند. و بهتر از آن، Cahce باعث





می‌شود که مقالات و تبلیغات پر زرق و برق سریع‌تر نشان داده شوند و در صفحه‌نمایش بیننده بهتر دیده شوند و آن‌ها را تشویق به مصرف محتوای بیشتر و دیدن تبلیغات بیشتر می‌کند. این همان چیزی است که ارائه دهنگان محتوا می‌خواهند! چشم‌های بیشتر و تبلیغات بیشتر!

بسیاری از ارائه‌دهنگان محتوا از طریق تبلیغات پول می‌گیرند – به ویژه، هر بار که تبلیغی به کاربر نشان داده می‌شود، پول دریافت می‌کنند (شاید فقط کسری از یک یا دو پنی، اما اگر یک میلیون تبلیغ در روز نشان دهید چه!). و این مشکل با Cache است – آن‌ها می‌توانند تعداد دسترسی واقعی را از سرور مبدا پنهان کنند. اگر Cache کردن کامل بود، سرور مبدا ممکن است هیچ گونه دسترسی HTTP را دریافت نکند، زیرا آن‌ها توسط Cache‌های اینترنتی جذب می‌شوند. اما، اگر از نظر تعداد دسترسی به شما پولی پرداخت شود، جشن نخواهد گرفت.

The Publisher's Response

امروزه، تبلیغ‌کنندگان از انواع تکنیک‌های cache-busting استفاده می‌کنند تا اطمینان حاصل کنند که Cache‌ها جریان بازدیدشان را نمی‌رزند. آن‌ها هدرهای بدون Cache را روی محتوای خود می‌زنند. آن‌ها تبلیغات را از طریق دروازه‌های CGI ارائه می‌دهند. آن‌ها URL‌های تبلیغات را در هر دسترسی بازنویسی می‌کنند.

و این تکنیک‌های حذف Cache فقط برای Proxy Cache‌ها نیستند. در واقع، امروزه آن‌ها عمدتاً در Cache هستند که در هر مرورگر وب فعال است. متاسفانه، در حالی که بیش از حد تلاش می‌کنند تا جریان بازدید خود را حفظ کنند، برخی از ارائه‌دهنگان محتوا در حال کاهش اثرات مثبت Cache کردن در سایت خود هستند.

در دنیای ایده‌آل، ارائه‌دهنگان محتوا به Cache‌ها اجازه می‌دهند ترافیک آن‌ها را جذب کنند و Cache‌ها به آن‌ها می‌گویند که چه تعداد بازدید داشته‌اند. امروزه چند راه وجود دارد که Cache‌ها می‌توانند این کار را انجام دهند.

یک راه حل این است که Cache‌ها برای تأیید مجدد با سرور مبدا در هر دسترسی پیکربندی کنند. این یک ضربه به سرور مبدا برای هر دسترسی وارد می‌کند، اما معمولاً هیچ داده در Body را منتقل نمی‌کند. البته این باعث کندی تراکنش می‌شود.

Log Migration

یک راه حل ایده‌آل نیازی به ارسال بازدیدها از طریق سرور ندارد. پس از همه، Cache می‌تواند گزارشی از تمام بازدیدها نگه دارد. Cache‌ها فقط می‌توانند گزارش‌های بازدید را در سرورها توزیع کنند. در





واقع، برخی از ارائه‌دهندگان بزرگ Cache می‌دانند که گزارش‌های Cache را به صورت دستی پردازش و به ارائه‌دهندگان محتوای با نفوذ تحویل می‌دهند تا ارائه‌دهندگان محتوا را راضی نگه دارند.

متأسفانه، لاغ‌های مربوط به این بخش بزرگ هستند، که حرکت آن‌ها را سخت می‌کند و گزارش‌های Cache استاندارد یا سازمان‌دهی نشده‌اند تا لاغ‌ها را برای ارائه‌دهندگان محتوای جداگانه جدا کنند. همچنین، مسائل مربوط به احراز هویت و حفظ حریم خصوصی نیز وجود دارد.

پیشنهادهایی برای طرح‌های توزیع مجدد گزارش کارآمد (و کمتر کارآمد) ارائه شده است. هیچ کدام به اندازه کافی توسعه نیافتدند که توسط فروشنندگان نرم افزار وب مورد استفاده قرار گیرد. بسیاری از آن‌ها بسیار پیچیده هستند و برای موفقیت نیاز به مشارکت تجاری دارند. چندین شرکت سرمایه گذاری برای توسعه زیرساخت‌های حمایتی برای احیای درآمد تبلیغاتی راه اندازی شده‌اند.

Hit Metering and Usage Limiting

طرح بسیار ساده‌تری را تعریف Simple Hit-Metering and Usage-Limiting for HTTP، RFC 2227 می‌کند. این پروتکل یک هدر جدید به HTTP به نام Meter اضافه می‌کند که به صورت دوره‌ای تعداد بازدیدها را برای URL‌های خاص به سرورها بر می‌گرداند. به این ترتیب، سرورها بروزرسانی‌های دوره‌ای را از Cache‌ها در مورد تعداد دفعاتی که اسناد ذخیره شده در Cache وارد شده است، دریافت می‌کنند.

علاوه بر این، سرور می‌تواند کنترل کند که چند بار اسناد می‌توانند از Cache ارائه شوند، قبل از اینکه باید به سرور گزارش دهد. این موضوع usage limiting نامیده می‌شود. این ویژگی به سرورها اجازه می‌دهد تا قبل از نیاز به گزارش دادن به سرور مبدأ، میزان استفاده از یک منبع Cache را کنترل کنند.

For More Information

<http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

<http://search.ietf.org/rfc/rfc3040>.

<http://search.ietf.org/rfc/rfc3143>.

<http://www.squid-cache.org>





فصل هشتم – Integration Points: Gateways, Tunnels, and Relays

وب ثابت کرده است که ابزاری باورنگردنی برای انتشار محتوا است. با گذشت زمان، تمایل افراد در ایجاد محتوا از قرار دادن اسناد ثابت آنلاین به تمایل به اشتراک گذاری منابع پیچیده‌تر، مانند محتوای پایگاه داده یا صفحات HTML به صورت پویا تغییر کرده است. برنامه‌های کاربردی HTTP، مانند مرورگرهای وب، ابزار یکپارچه‌ای را برای دسترسی به محتوا از طریق اینترنت در اختیار کاربران قرار داده اند.

HTTP همچنین به یک بلوک اساسی برای توسعه دهنده‌گان برنامه تبدیل شده است، که پروتکل‌های دیگر را در بالای HTTP پشت سر می‌گذارند (به عنوان مثال، استفاده از HTTP برای تونل کردن یا انتقال ترافیک پروتکل‌های دیگر از طریق فایروال‌ها، با قرار دادن آن ترافیک در HTTP). HTTP به عنوان یک پروتکل برای تمام منابع وب استفاده می‌شود و همچنین پروتکلی است که سایر برنامه‌ها و پروتکل‌های اپلیکیشن از آن برای انجام کارهای خود استفاده می‌کنند.

این فصل نگاهی کلی به برخی از روش‌هایی دارد که توسعه‌دهنده‌گان برای استفاده از HTTP برای دسترسی به منابع مختلف ارائه کرده‌اند و بررسی می‌کند که چگونه توسعه‌دهنده‌گان از HTTP به عنوان چارچوبی برای فعال کردن سایر پروتکل‌ها و ارتباطات برنامه‌ها استفاده می‌کنند.

در این فصل به بحث زیر می‌پردازیم:

- **Gateway** هایی که HTTP را با سایر پروتکل‌ها و برنامه‌ها مرتبط می‌کنند.
- **Interface** های برنامه، که به انواع مختلف برنامه‌های کاربردی وب اجازه می‌دهد تا با یکدیگر ارتباط برقرار کنند.
- تونل‌هایی که به شما امکان می‌دهند ترافیک غیر HTTP را از طریق اتصالات HTTP ارسال کنید.
- **Relay** یا رله‌ها، که یک نوع پراکسی HTTP ساده‌شده هستند و برای ارسال داده‌ها در یک بار پرش (Hop) استفاده می‌شوند.

Gateways

تاریخچه پشت Extention ها و رابطه‌ای HTTP بر اساس نیازهای افراد ایجاد شده است. هنگامی که تمایل به قرار دادن منابع پیچیده‌تر در وب پدیدار شد، به سرعت مشخص شد که هیچ برنامه کاربردی واحدی نمی‌تواند تمام منابع قابل تصور را مدیریت کند.

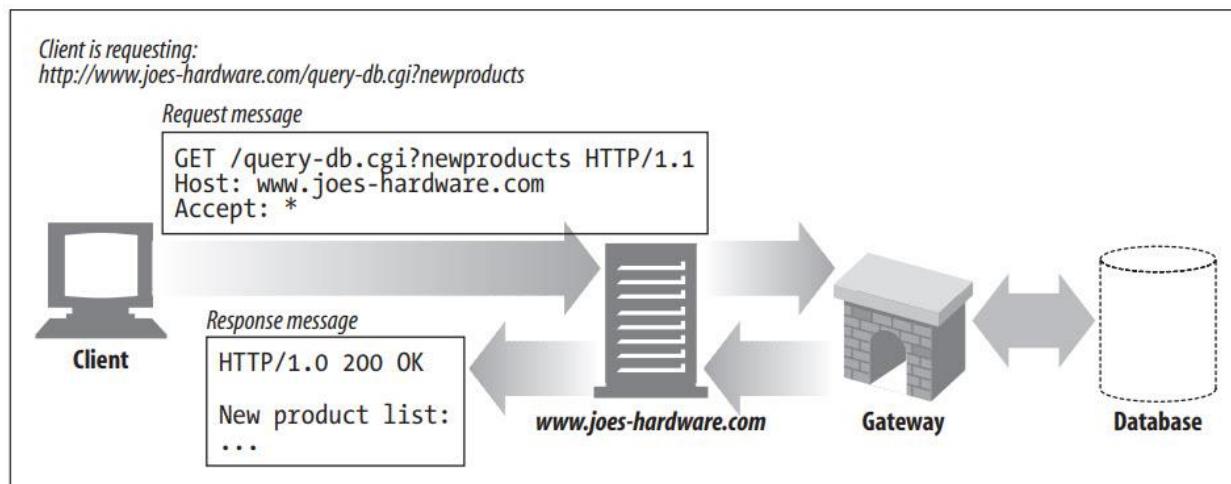
برای دور زدن این مشکل، توسعه‌دهنده‌گان به مفهوم **Gateway** رسیدند که می‌تواند به عنوان نوعی مفسر عمل کند و راهی را برای دستیابی به منبع ارائه نماید. **Gateway** در واقع یک چسب بین منابع





و برنامه‌ها است. یک برنامه کاربردی می‌تواند (از طریق HTTP یا یک Interface) از یک برای رسیدگی به درخواست، بپرسد و Gateway می‌تواند پاسخی ارائه دهد. Gateway می‌تواند با زبان پرس و جو به پایگاه داده صحبت کند یا محتوای پویا را تولید کند و مانند یک پورتال عمل کند: یک درخواست وارد می‌شود و یک پاسخ بیرون می‌آید.

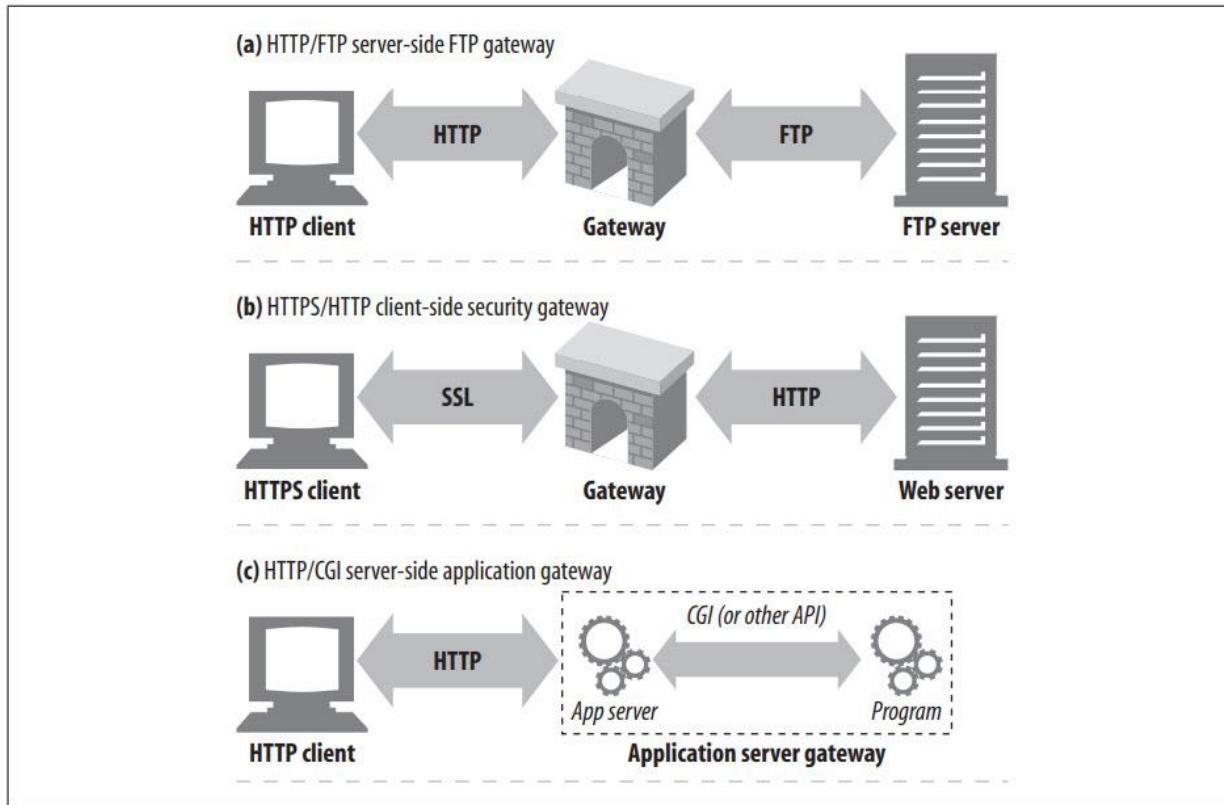
شکل زیر نوعی Resource Gateway را نشان می‌دهد. در اینجا، سرور سخت افزار Joe به عنوان دروازه‌ای برای محتوای پایگاه داده عمل می‌کند - توجه داشته باشید که کلاینت به سادگی از طریق HTTP منبعی را درخواست می‌کند و سرور سخت افزار Joe با یک Gateway برای دسترسی به منبع ارتباط برقرار می‌کند.



برخی از Gateway‌ها به‌طور خودکار ترافیک HTTP را به پروتکل‌های دیگر ترجمه می‌کنند، بنابراین کلاینت‌های HTTP می‌توانند با برنامه‌های کاربردی دیگر ارتباط برقرار کنند بدون اینکه کلاینت‌ها نیازی به دانستن پروتکل‌های دیگر داشته باشند.



شکل زیر سه نمونه از Gateway ها را نشان می دهد:



در بخش a از شکل بالا، **Gateway** درخواست‌های **HTTP** های **FTP** URL را برای **HTTP** دریافت می‌کند. سپس دروازه اتصالات **FTP** را باز می‌کند و دستورات مناسب را به سرور **FTP** صادر می‌کند. سند از طریق **HTTP** به همراه هدراهای صحیح **HTTP** بازگردانده می‌شود.

در بخش b از شکل بالا، **Gateway** یک درخواست وب رمزگذاری شده را از طریق **SSL** دریافت می‌کند، درخواست را رمزگشایی می‌کند و یک درخواست **HTTP** معمولی را به سرور مقصد ارسال می‌کند. این شتاب دهنده‌های امنیتی را می‌توان مستقیماً در مقابل سرورهای وب (معمولًاً در همان محل) قرار داد تا رمزگذاری با کارایی بالا را برای سرورهای مبدأ فراهم کند.

در بخش c شکل بالا، **Application Server Gateway** API **HTTP** را از طریق یک **HTTP** مشتریان **Application Server Gateway** به برنامه‌های کاربردی سمت سرور متصل می‌کند. وقتی از فروشگاه‌های تجارت الکترونیک در وب خرید می‌کنید، پیش‌بینی آب و هوا را بررسی می‌کنید یا قیمت‌های سهام را دریافت می‌کنید، در حال بازدید از **Application Server Gateway** هاستید.





Client-Side and Server-Side Gateways

Gateway های وب از یک طرف HTTP و در طرف دیگر با پروتکل متفاوتی صحبت می‌کنند.

<client-protocol>/<server-protocol>

بنابراین ای که کلاینت‌های HTTP را به سرورهای خبری NNTP پیوند می‌دهد، یک Client-Side HTTP/NNTP Gateway است. ما از اصطلاحات "Server-Side Gateway" و "Gateway" برای توضیح اینکه تبدیل برای کدام سمت Gateway انجام می‌شود استفاده می‌کنیم:

- Server-Side Gateway ها بوسیله HTTP با کلاینت‌ها و یک پروتکل خارجی با سرورها (*HTTP/*)

صحبت می‌کنند.

- Client-Side Gateway ها بوسیله پروتکل‌های خارجی با کلاینت‌ها و HTTP با سرورها (*/*HTTP)

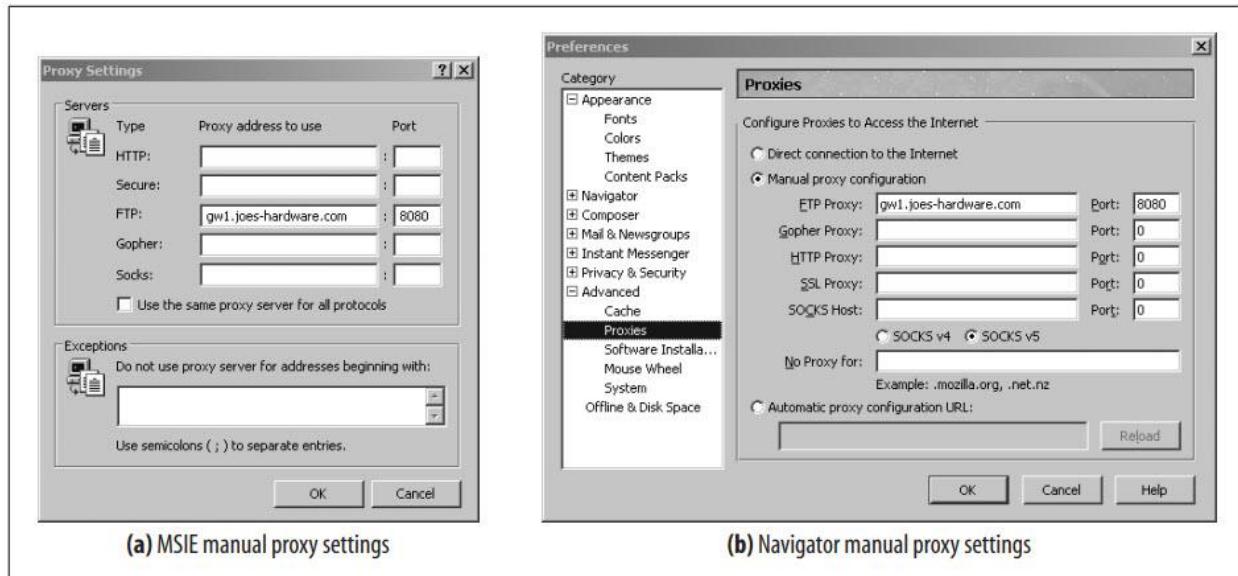
صحبت می‌کنند.

Protocol Gateways

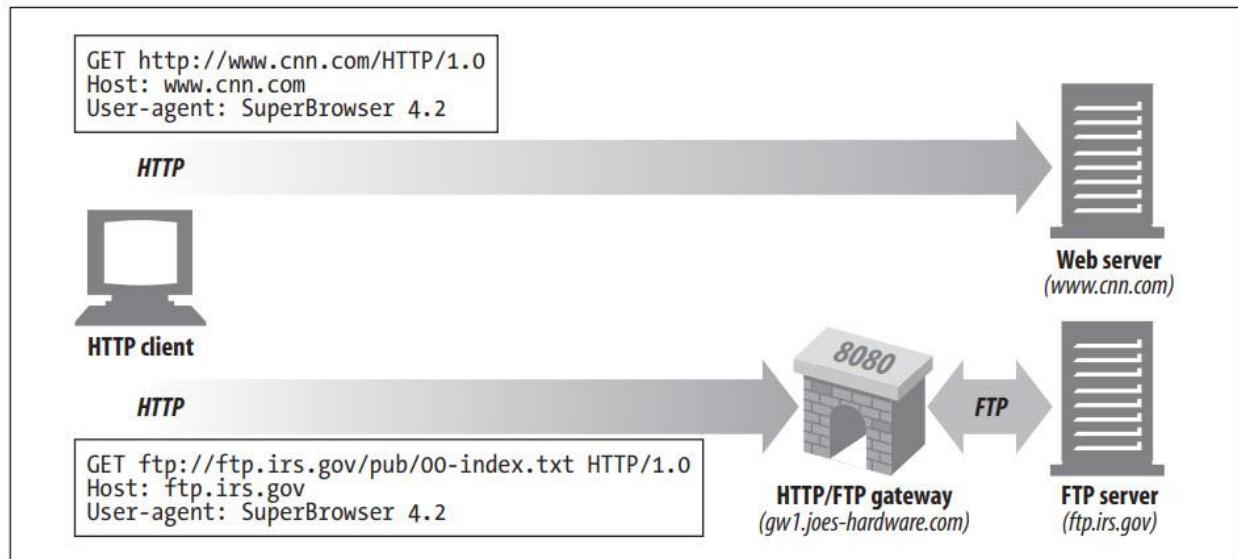
شما می‌توانید ترافیک HTTP را به همان روشی که ترافیک را به پرائسی‌ها هدایت می‌کنید به Gateway ها هدایت کنید. معمولاً، شما صراحتاً مرورگرها را برای استفاده از Gateway ها، رهگیری ترافیک شفاف، یا پیکربندی Gateway ها به عنوان جانشین (پرائسی‌های معکوس) پیکربندی می‌کنید.

شکل زیر کادرهای محاوره ای مورد استفاده برای پیکربندی مرورگر برای استفاده از FTP های سمت Server را نشان می‌دهد. در پیکربندی نشان داده شده، مرورگر برای استفاده از gw1.joeshardware.com به عنوان دروازه HTTP/FTP برای همه URL های FTP پیکربندی شده است. مرورگر به جای ارسال دستورات FTP به سرور gw1.joeshardware.com را به دروازه HTTP/FTP gw1.joeshardware.com در پورت ۸۰۸۰ ارسال می‌کند.





نتیجه این پیکربندی Gateway در شکل زیر نشان داده شده است.



ترافیک HTTP معمولی تحت تأثیر قرار نمی‌گیرد و به طور مستقیم به سرورهای مبدا جریان می‌یابد. اما درخواست‌ها برای URL‌های FTP به Gateway مربوط به gw1.joes-hardware.com در درخواست‌های HTTP ارسال می‌شوند. تراکنش‌های FTP را از طرف کلاینت انجام می‌دهد و نتایج را توسط HTTP به کلاینت باز می‌گرداند.

در ادامه به انواع رایج Gateway‌ها را می‌پردازیم.



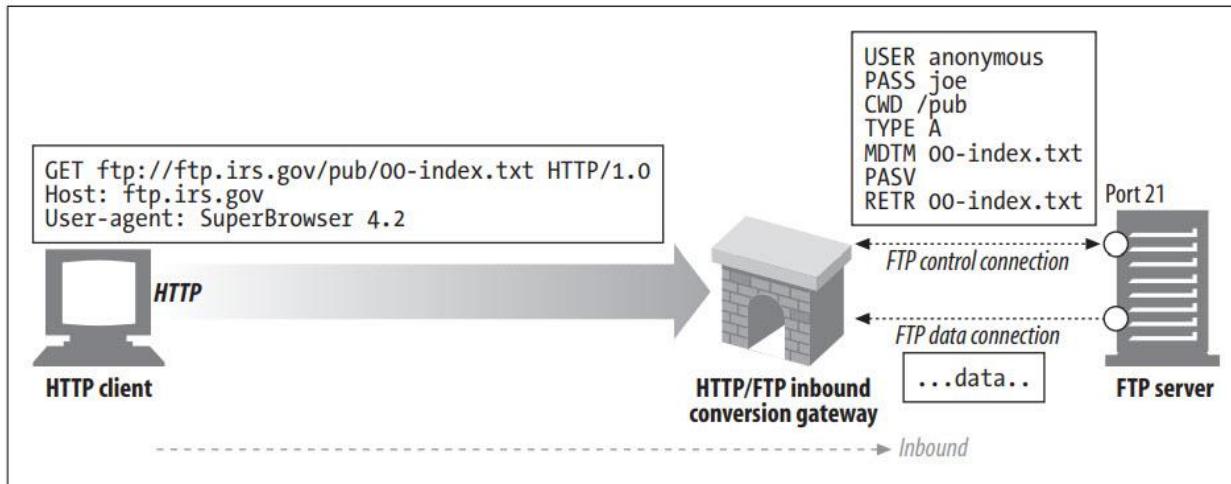


HTTP/*: Server-Side Web Gateways

درخواست‌های HTTP سمت کلاینت را به یک پروتکل خارجی تبدیل می‌کنند، زیرا درخواست‌ها به سمت سرور مبدأ وارد می‌شوند.

در شکل زیر، یک درخواست HTTP برای یک منبع FTP دریافت می‌کند:

`ftp://ftp.irs.gov/pub/00-index.txt`



شروع به باز کردن اتصال FTP به پورت FTP در سرور مبدأ (پورت ۲۱) می‌کند و با پروتکل FTP برای واکشی شی صحبت می‌کند. **Gateway** کارهای زیر را انجام می‌دهد:

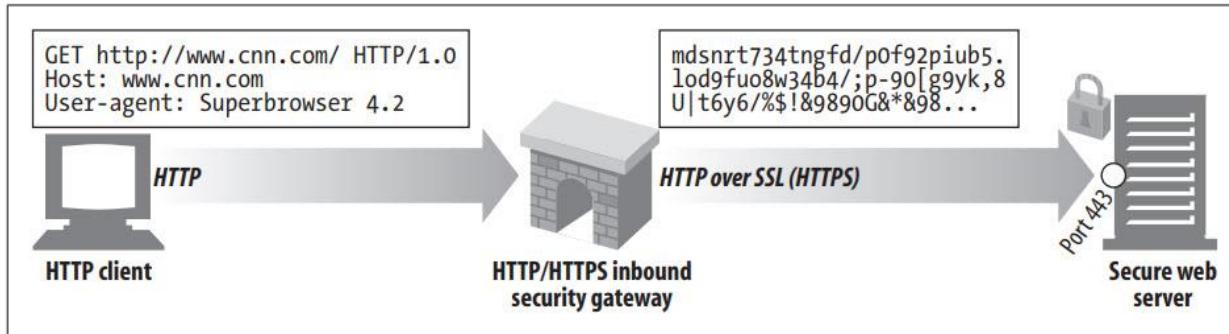
- دستورات USER و PASS را برای ورود به سرور ارسال می‌کند.
- دستور CWD را برای تغییر به دایرکتوری مناسب روی سرور صادر می‌کند.
- نوع دانلود را روی ASCII تنظیم می‌کند.
- آخرین زمان اصلاح سند را با MDTM واکشی می‌کند.
- به سرور می‌گوید که با استفاده از PASV انتظار بازیابی اطلاعات غیرفعال را داشته باشد.
- درخواست بازیابی شی با استفاده از RETR
- اتصال داده‌ای را به سرور FTP در پورت بازگردانده شده در کانال کنترل باز می‌کند. به محض باز شدن کانال داده، محتوای شیء به **Gateway** باز می‌گردد.

هنگامی که بازیابی کامل شد، شی در یک پاسخ HTTP برای کلاینت ارسال می‌شود.



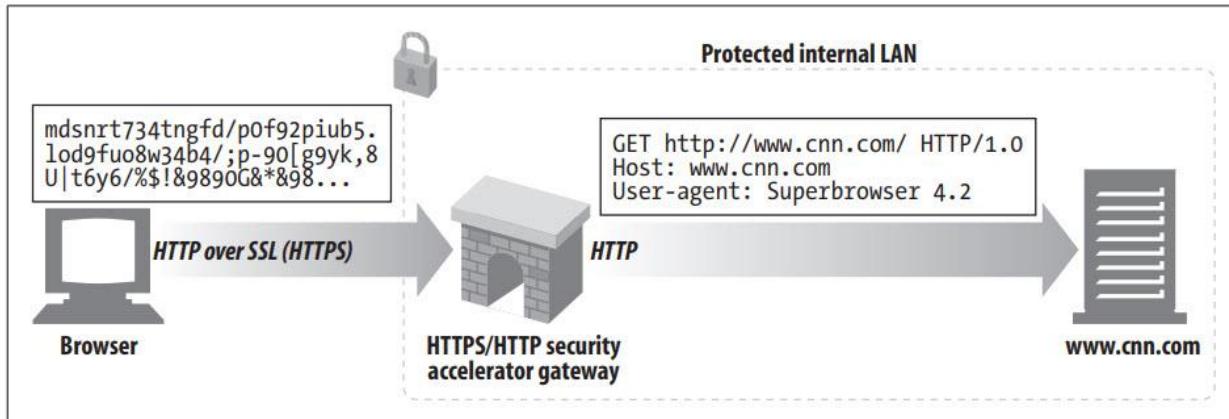
HTTP/HTTPS: Server-Side Security Gateways

از Gateway ها می‌توان برای ایجاد حریم خصوصی و امنیت بیشتر برای یک سازمان، با رمزگذاری تمام درخواست‌های وب ورودی استفاده کرد. کلاینت‌ها می‌توانند با استفاده از HTTP معمولی، وب را مرور کنند، اما به طور خودکار جلسات کاربر را رمزگذاری می‌کند (شکل زیر).



HTTPS/HTTP: Client-Side Security Accelerator Gateways

اخیراً HTTPS/HTTP Gateway های به عنوان شتاب دهنده‌های امنیتی محبوب شده‌اند. این Gateway های HTTPS/HTTP که در مقابل سرور وب قرار می‌گیرند، معمولاً به عنوان یک Gateway رهگیری نامрئی یا یک پروکسی معکوس عمل می‌کنند. آن‌ها ترافیک امن HTTPS را دریافت می‌کنند، ترافیک امن را رمزگشایی می‌کنند و درخواست‌های HTTP معمولی را به سرور وب می‌دهند (شکل زیر).



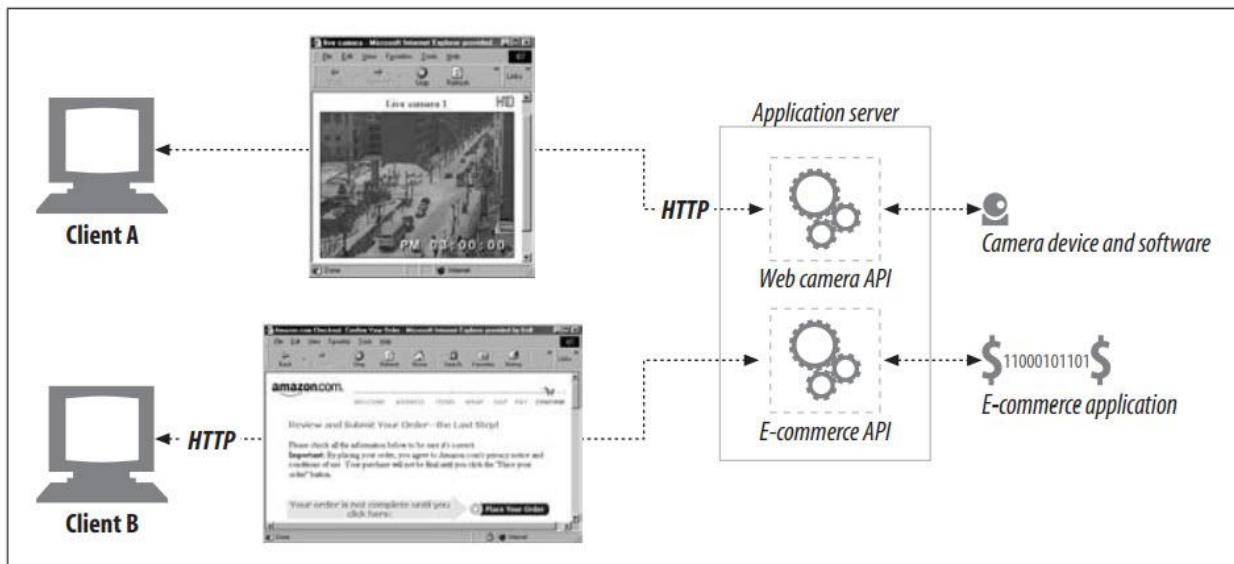
این Gateway ها اغلب شامل سخت‌افزار رمزگشایی ویژه برای رمزگشایی ترافیک امن بسیار کارآمدتر از سرور مبدا هستند و بار را از سرور مبدا حذف می‌کنند. از آنجا که این Gateway ها ترافیک رمزگذاری نشده را بین Gateway و سرور مبدا ارسال می‌کنند، باید احتیاط کنید تا مطمئن شوید شبکه بین Gateway و سرور مبدا ایمن است.





Resource Gateways

تا کنون، ما در مورد **Gateway**هایی صحبت کرده‌ایم که کلاینت‌ها و سرورها را در یک شبکه به هم متصل می‌کنند. با این حال، رایج ترین شکل **Gateway**، سرور برنامه، سرور مقصد و **Gateway** را در یک سرور واحد ترکیب می‌کند. سرورهای برنامه **Gateway**های سمت سرور هستند که با سرویس گیرنده HTTP صحبت می‌کنند و به یک برنامه کاربردی در سمت سرور متصل می‌شوند (شکل زیر را ببینید).



در شکل بالا، دو کلاینت با استفاده از HTTP به یک سرور برنامه متصل می‌شوند. اما، به جای ارسال فایل‌ها از سرور، سرور برنامه درخواست‌ها را از طریق یک رابط برنامه‌نویسی برنامه‌نویسی دروازه (API) به برنامه‌های در حال اجرا روی سرور ارسال می‌کند:

- درخواست کلاینت A دریافت شده و بر اساس URI از طریق یک API به یک برنامه دوربین دیجیتال ارسال می‌شود. تصویر دوربین به دست آمده در یک پیام پاسخ HTTP قرار می‌گیرد و برای نمایش در مرورگر کلاینت به وی ارسال می‌شود.
- کلاینت B برای یک برنامه تجارت الکترونیکی است. درخواست‌های کلاینت B از طریق API دروازه URI سرور به نرم افزار تجارت الکترونیک ارسال می‌شود و نتایج به مرورگر بازگردانده می‌شود. نرم افزار تجارت الکترونیک با کلاینت تعامل دارد و کاربر را از طریق دنباله ای از صفحات HTML برای تکمیل خرید راهنمایی می‌کند.

CGI یا Common Gateway Interface API اولین API محبوب برای Application Gateways بود. CGI مجموعه استاندارد شده‌ای از اینترفیس‌ها است که سرورهای وب برای راه اندازی برنامه‌ها در پاسخ به درخواست‌های URL برای HTTP URL های خاص، جمع آوری خروجی برنامه و ارسال آن در



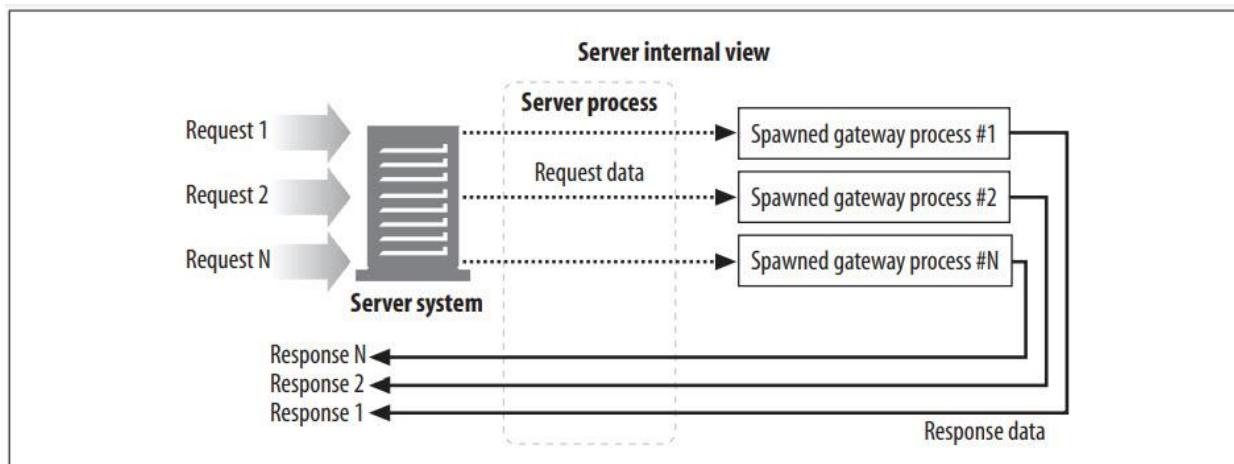


پاسخهای HTTP استفاده می‌کنند. در چند سال گذشته، وب سرورهای تجاری، اینترفیس‌های پیچیده‌تری را برای اتصال سرورهای وب به برنامه‌ها ارائه کرده‌اند.

وب سرورهای اولیه ساخته‌های نسبتاً ساده‌ای بودند و رویکرد ساده‌ای که برای پیاده‌سازی یک اینترفیس برای **Gateway**‌ها در نظر گرفته شد تا به امروز باقی مانده است.

هنگامی که یک درخواست برای منبعی وارد می‌شود که به یک **Gateway** نیاز دارد، سرور برنامه کمکی را برای رسیدگی به درخواست ارسال می‌کند. برنامه کمکی داده‌های مورد نیاز خود را ارسال می‌کند. اغلب این فقط کل درخواست یا چیزی شبیه پرس و جوی است که کاربر می‌خواهد در پایگاه داده اجرا کند.

سپس یک پاسخ یا داده پاسخ را به سرور برمی‌گرداند که آن را به کلاینت منتقل می‌کند. سرور و **Gateway** برنامه‌های جداگانه‌ای هستند، بنابراین خطوط مسئولیت روشن نگه داشته می‌شوند. شکل زیر مکانیزم اساسی در پشت تعاملات برنامه کاربردی سرور و **Gateway** را نشان می‌دهد.



این پروتکل ساده (درخواست وارد کردن، تحویل دادن و پاسخ دادن) جوهره پشت قدیمی‌ترین و یکی از رایج‌ترین اینترفیس‌ها یعنی CGI است.

Common Gateway Interface (CGI)

Common Gateway Interface اولین و احتمالاً هنوز هم پرکاربردترین افزونه سرور است و در سرتاسر وب برای مواردی مانند HTML پویا، پردازش کارت اعتباری و جستجو در پایگاه داده استفاده می‌شود.

از آنجایی که برنامه‌های CGI جدا از سرور هستند، تقریباً می‌توان آن‌ها را در هر زبانی از جمله Perl، C، Tcl و زبان‌های مختلف Shell پیاده‌سازی کرد. از آنجایی که CGI ساده است، تقریباً تمام سرورهای HTTP از آن پشتیبانی می‌کنند. مکانیزم اصلی مدل CGI در شکل بالا نشان داده شده است.





پردازش CGI برای کاربران نامرئی است. از دیدگاه کلاینت، این فقط یک درخواست عادی بوده و وی کاملاً از روند دستیابی بین سرور و برنامه CGI بی اطلاع است. تنها اشاره ای در کلاینت که نشان دهنده استفاده یک برنامه از CGI است، وجود حروف "cgi" و شاید "? در URL است.

بنابراین CGI فوق العاده است، درست است؟ خوب، بله و نه. این یک شکل ساده و کاربردی از چسب بین سرورها و تقریباً هر نوع منبعی را فراهم نموده و هر ترجمه‌ای را که نیاز به انجام دارد مدیریت می‌کند. این رابط همچنین برای محافظت از سرور در برابر برنامه‌های افزودنی دارای باگ ظریف است (اگر برنامه افزودنی روی خود سرور قرار گیرد، ممکن است باعث خطا شود که ممکن است سرور را از کار بیندازد).

با این حال، این جداسازی هزینه در عملکرد دارد. هزینه سربار ایجاد یک فرآیند جدید برای هر درخواست CGI بسیار زیاد است و عملکرد سرورهایی را که از CGI استفاده می‌کنند محدود می‌کند و منابع ماشین سرور را هدر می‌دهد. برای تلاش برای دور زدن این مشکل، شکل جدیدی از CGI - که به درستی Fast CGI نامیده می‌شود - ایجاد شده است. این رابط از CGI تقلید می‌کند، اما به عنوان یک شیخ پایدار اجرا می‌شود و جریمه‌ی عملکردی را برای راهاندازی و از بین بردن یک فرآیند جدید برای هر درخواست حذف می‌کند.

Server Extension APIs

پروتکل CGI روشی تمیز برای اتصال مفسرهای خارجی با سرورهای HTTP موجود ارائه می‌کند، اما اگر بخواهید رفتار خود سرور را تغییر دهید، یا فقط می‌خواهید آخرین قطره عملکردی را که می‌توانید از سرور خود خارج کنید، چه باید کرد؟ برای این دو نیاز، توسعه دهندگان Server Extension APIs را ارائه کرده اند که یک رابط قدرتمند برای توسعه دهندگان وب فراهم می‌کند تا مازولهای خود را مستقیماً با یک سرور HTTP ارتباط برقرار کنند. Extension APIs به برنامه نویسان اجازه می‌دهند تا کد خود را به سرور پیوند بزنند یا به طور کامل یک جزء از سرور را تعویض کرده و آن را با کد خود جایگزین کنند.

اکثر سرورهای محبوب یک یا چند API افزونه را برای توسعه دهندگان ارائه می‌کنند. از آنجایی که این پسوندها اغلب با معماری خود سرور مرتبط هستند، اکثر آن‌ها مختص یک نوع سرور هستند. مایکروسافت، نت اسکیپ، آپاچی و سایر سرورها همگی دارای رابطهای API هستند که به توسعه دهندگان اجازه می‌دهد رفتار سرور را تغییر دهند یا رابطهای سفارشی را برای منابع مختلف ارائه دهند. این رابطهای سفارشی یک رابط قدرتمند برای توسعه دهندگان فراهم می‌کند.

یکی از نمونه‌های Server Extension، افزونه فرانت پیج سرور مایکروسافت (FPSE) است که از خدمات انتشار وب برای نویسندهای فرانت پیج پشتیبانی می‌کند. FPSE قادر است دستورات Remote





HTTP یا RPC ارسال شده توسط کلاینت فرانت پیج را تفسیر کند. این دستورات بر روی (به طور خاص، روی متد POST همپوشانی دارند).

Application Interfaces and Web Services

ما Gateway های منابع را به عنوان راههایی برای وب سرورها برای برقراری ارتباط با برنامه‌ها مورد بحث قرار داده‌ایم. به طور کلی‌تر، با برنامه‌های کاربردی وب که انواع بیشتری از خدمات را ارائه می‌دهند، مشخص می‌شود که HTTP می‌تواند بخشی پایه‌ای برای پیوند دادن برنامه‌ها به یکدیگر باشد. یکی از مسائل پیچیده‌تر در سیم‌کشی برنامه‌ها، مذاکره روی رابط پروتکل بین دو برنامه است تا بتوانند داده‌ها را مبادله کنند - اغلب این کار بر اساس برنامه به برنامه انجام می‌شود.

برای کار با هم، برنامه‌ها معمولاً نیاز به تبادل اطلاعات پیچیده‌تر با یکدیگر دارند که در هدرهای HTTP قابل بیان است. چند نمونه از گسترنش پروتکل‌های HTTP یا لایه‌بندی در بالای HTTP به منظور تبادل اطلاعات سفارشی شده در فصل ۱۹ توضیح داده خواهد شد.

جامعه اینترنتی مجموعه‌ای از استانداردها و پروتکل‌ها را توسعه داده است که به برنامه‌های کاربردی وب اجازه می‌دهد با یکدیگر صحبت کنند. این استانداردها به‌طور ساده تحت عنوان Web Service شناخته می‌شوند، اگرچه این اصطلاح می‌تواند به معنای خود برنامه‌های کاربردی وب (ساختمان) مستقل باشد. پیش‌فرض سرویس‌های وب جدید نیست، اما مکانیسم جدیدی برای برنامه‌های کاربردی جهت به اشتراک گذاری اطلاعات است. وب سرویس‌ها بر اساس فناوری‌های استاندارد وب مانند HTTP ساخته شده‌اند.

سرویس‌های وب با استفاده از XML از طریق SOAP اطلاعات را مبادله می‌کنند. زبان نشانه گذاری توسعه پذیر (XML) راهی برای ایجاد و تفسیر اطلاعات سفارشی شده در مورد یک شی داده فراهم می‌کند. پروتکل دسترسی ساده به اشیا (SOAP) استانداردی برای افزودن اطلاعات XML به پیام‌های HTTP است.

<http://www.w3.org/TR/2001/WD-soap12-part0-20011217> جهت کسب اطلاعات بیشتر، به مراجعه کنید. برنامه نویسی وب سرویس با SOAP، توسط داگ تیدول، جیمز اسنل، و پاول کولچنکو (O'Reilly) است. نیز منبع عالی اطلاعات در مورد پروتکل SOAP است.

Tunnels

ما روش‌های مختلفی را مورد بحث قرار داده‌ایم که از HTTP می‌توان برای فعال کردن دسترسی به انواع منابع (از طریق Gateway ها) و فعال کردن ارتباط برنامه به برنامه استفاده کرد. در این بخش، نگاهی به کاربرد





دیگری از HTTP خواهیم داشت، تونل‌های وب، دسترسی به برنامه‌هایی که از پروتکل‌های غیر HTTP استفاده می‌کنند را از طریق برنامه‌های HTTP امکان‌پذیر می‌سازد.

تونل‌های وب به شما امکان می‌دهند ترافیک غیر HTTP را از طریق اتصالات HTTP ارسال کنید و به پروتکل‌های دیگر اجازه می‌دهد در بالای HTTP به عقب بروند. رایج‌ترین دلیل استفاده از تونل‌های وب، تعییه ترافیک غیر HTTP در یک اتصال HTTP است، بنابراین می‌توان آن را از طریق فایروال‌هایی ارسال کرد که فقط به ترافیک وب اجازه وارد شدن می‌دهند.

Establishing HTTP Tunnels with CONNECT

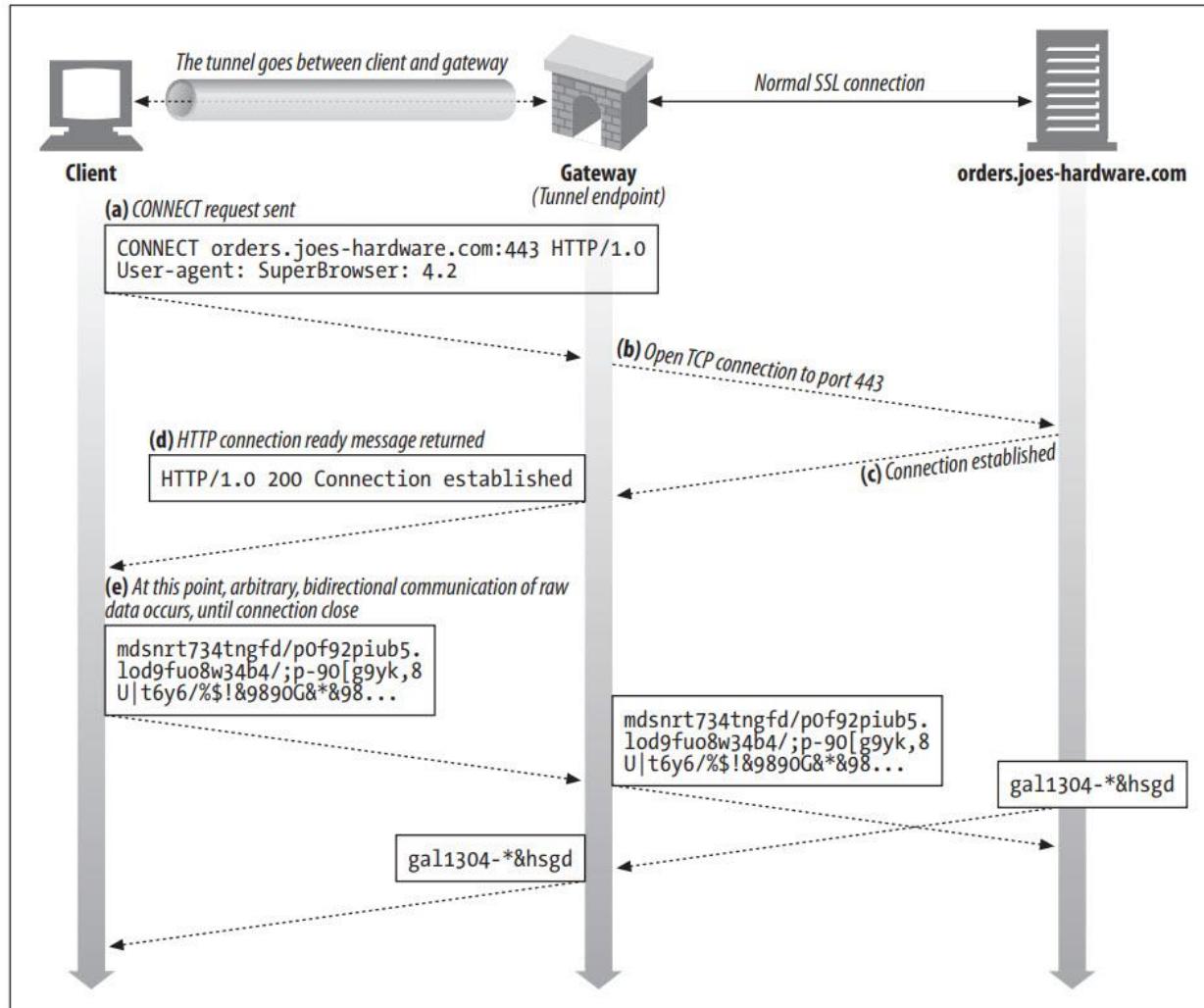
تونل‌های وب با استفاده از روش HTTP's CONNECT ایجاد می‌شوند. پروتکل CONNECT بخشی از مشخصات اصلی HTTP/1.1 نیست، اما یک برنامه افزودنی است که به طور گستردۀ پیاده سازی شده است. مشخصات فنی را می‌توان در مشخصات پیش‌نویس اینترنت منقضی شده آری لوتون، «تونل‌سازی پروتکل‌های مبتنی بر TCP از طریق سرورهای پروکسی وب» یا در کتاب سرورهای پروکسی وب او یافت که هر دو در پایان این فصل ذکر شده‌اند.

متدهای CONNECT از یک Tunnel Gateway می‌خواهد که یک اتصال TCP به سرور و پورت مقصد دلخواه ایجاد کند و داده‌های بعدی را کورکورانه بین کلاینت و سرور ارسال نماید.

شكل زیر نحوه عملکرد متدهای CONNECT برای ایجاد یک تونل به یک دروازه را **Gateway** می‌دهد:

- در بخش a شکل، کلاینت یک درخواست CONNECT را به Tunnel Gateway ارسال می‌کند. متدهای CONNECT کلاینت از Tunnel Gateway می‌خواهد که یک اتصال TCP را باز کند (در اینجا، به میزبانی به نام orders.joes-hardware.com در پورت ۴۴۳، پورت پیش‌فرض SSL).
- اتصال TCP در بخش b و بخش c شکل ایجاد شده است.
- هنگامی که اتصال TCP برقرار شد، دروازه با ارسال یک پاسخ HTTP 200 Connection به کلاینت (بخش d شکل) اطلاع می‌دهد.
- در این مرحله، تونل راه اندازی می‌شود. هر داده‌ای که توسط کلاینت از طریق تونل HTTP ارسال می‌شود، مستقیماً به اتصال TCP خروجی رله می‌شود و هر داده‌ای که توسط سرور ارسال می‌شود از طریق تونل HTTP به کلاینت منتقل می‌گردد.





مثال در موجود در شکل بالا یک تونل SSL را توصیف می‌کند که در آن ترافیک HTTP از طریق یک اتصال SSL ارسال می‌شود، اما روش CONNECT می‌تواند برای ایجاد یک اتصال TCP به هر سرور با استفاده از هر پروتکل استفاده شود.

CONNECT requests

ساختار CONNECT از نظر شکل با سایر متدهای HTTP یکسان است، به استثنای خط شروع. URI درخواست با یک نام میزبان و به دنبال آن یک دونقطه و به دنبال آن یک شماره پورت جایگزین می‌شود. هر دو میزبان و پورت باید مشخص شوند:

CONNECT home.netscape.com:443 HTTP/1.0

User-agent: Mozilla/4.0





بعد از خط شروع، مانند سایر پیامهای HTTP، فیلدهای هدر درخواست HTTP صفر یا بیشتر وجود دارد. طبق معمول، خطوط به CRLF ختم می‌شوند و فهرست هدرها با CRLF خالی به پایان می‌رسد.

CONNECT responses

پس از ارسال درخواست، کلاینت منتظر پاسخ از **Gateway** می‌شود. همانند پیامهای HTTP معمولی، کد پاسخ 200 نشان دهنده موفقیت است. طبق قرارداد، **Reason Pharse** یا عبارت دلیل در پاسخ معمولاً روی "Connection Established" تنظیم می‌شود:

HTTP/1.0 200 Connection Established

Proxy-agent: Netscape-Proxy/1.1

برخلاف پاسخهای معمولی HTTP، پاسخ نیازی به هدر Content-Type نداشته و هیچ نوع محتوایی مورد نیاز نیست. زیرا اتصال به جای حامل پیام به یک رله بایت خام تبدیل می‌شود.

Data Tunneling, Timing, and Connection Management

از آنجایی که داده‌های تونل شده نسبت به **Gateway** مات هستند(شفاف نیستند)، **Gateway** نمی‌تواند هیچ فرضی در مورد ترتیب و جریان بسته‌ها داشته باشد. پس از ایجاد تونل، داده‌ها در هر زمانی در هر جهتی جریان دارند.

به عنوان یک بهینه سازی عملکرد، کلاینت‌ها اجازه دارند پس از ارسال درخواست CONNECT، اما قبل از دریافت پاسخ، داده‌های تونل را ارسال کنند. این داده‌ها را سریع‌تر به سرور می‌رسانند، اما به این معنی است که **Gateway** باید بتواند داده‌های پس از درخواست را به درستی مدیریت کند. به طور خاص، **Gateway** نمی‌تواند فرض کند که درخواست ورودی/خروجی شبکه فقط داده‌های هدر را برمی‌گرداند و **Gateway** باید مطمئن باشد که هر داده‌ای را که با هدر خوانده شده است، زمانی که اتصال آمده است، به سرور ارسال می‌کند.

اگر پاسخ به عنوان چالش احراز هویت یا سایر وضعیت‌های غیر 200 و nonfatal بازگردد، کلاینت‌هایی که داده‌ها را پس از درخواست ارسال می‌کنند، باید آماده باشند تا داده‌های درخواست را مجدداً ارسال کنند.

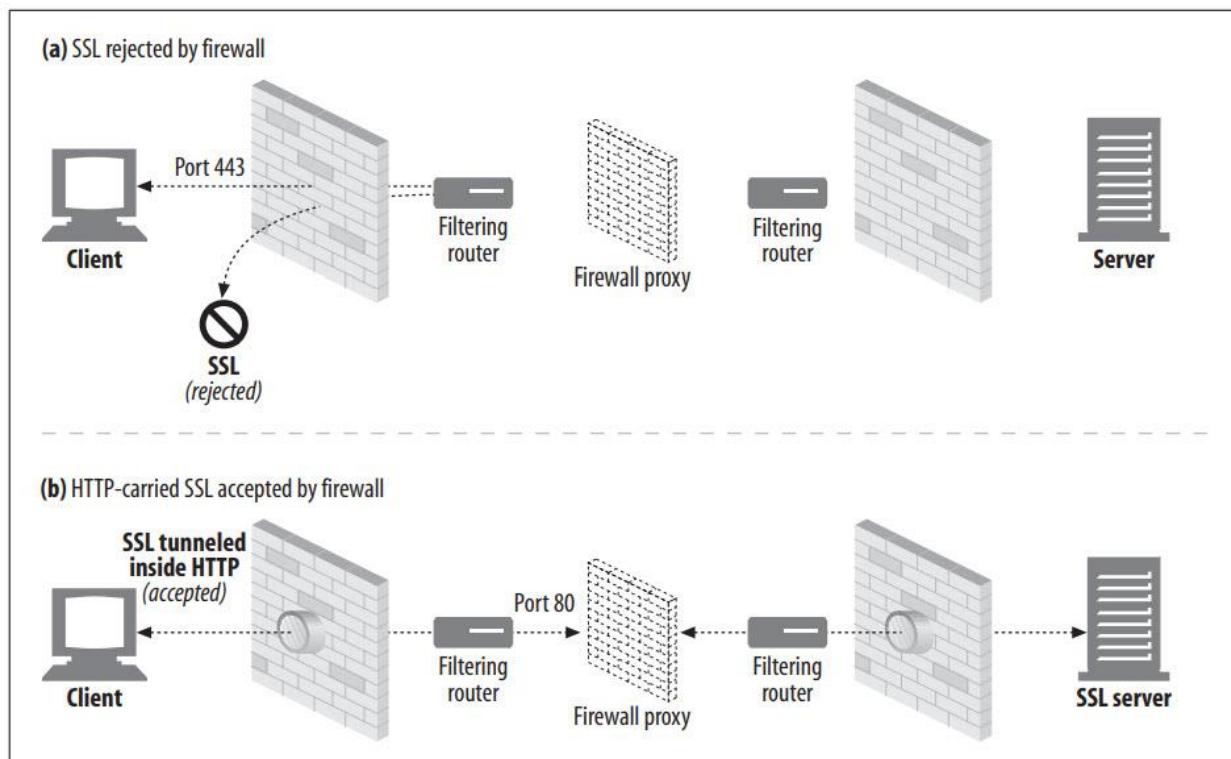
اگر در هر نقطه از یک Endpoint تونل قطع شود، هر داده باقی مانده‌ای که از آن آمده است به دیگری منتقل شده و پس از آن نیز اتصال دیگر توسط پراکسی قطع می‌شود. اگر داده‌های تحويل‌نشده برای Endpoint بسته وجود داشته باشد، آن داده‌ها کنار گذاشته می‌شوند.





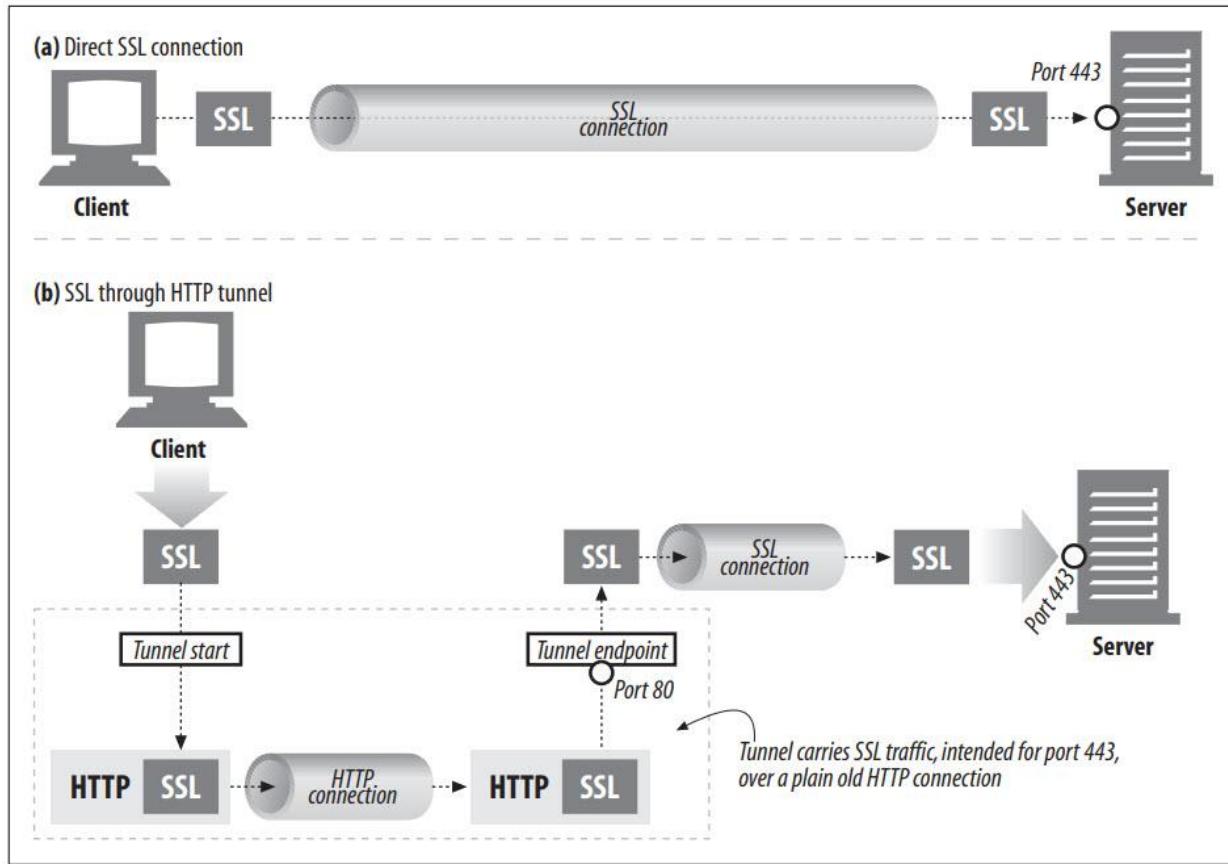
SSL Tunneling

تونل‌های وب ابتدا برای انتقال ترافیک رمزگذاری شده SSL از طریق فایروال‌ها توسعه یافتند. بسیاری از سازمان‌ها تمام ترافیک را از طریق روترهای Packet Filter و سرورهای پروکسی برای افزایش امنیت انتقال می‌دهند. اما برخی از پروتکل‌ها، مانند SSL رمزگذاری شده، توسط سرورهای پروکسی سنتی قابل پروکسی نیستند، زیرا اطلاعات رمزگذاری شده است. تونل‌ها به ترافیک SSL اجازه می‌دهند از طریق فایروال HTTP پورت ۸۰ با انتقال آن از طریق یک اتصال HTTP حمل شود.



برای اجازه دادن به ترافیک SSL از طریق فایروال‌های پراکسی موجود، یک ویژگی تونل سازی به HTTP اضافه شد که در آن داده‌های خام و رمزگذاری شده در پیام‌های HTTP قرار می‌گیرند و از طریق کانال‌های HTTP معمولی ارسال می‌شوند.





در بخش a از شکل بالا، ترافیک SSL مستقیماً به یک وب سرور امن (در پورت SSL 443) ارسال می‌شود. در بخش b، ترافیک SSL در پیام‌های HTTP کپسوله شده و از طریق اتصالات پورت HTTP 80 ارسال می‌شود، تا زمانی که دوباره به اتصالات SSL معمولی کپسوله شود.

در تونل‌ها اغلب برای عبور ترافیک غیر HTTP، از فایروال‌های فیلتر کننده پورت استفاده می‌شود. از این موضوع می‌توان به خوبی استفاده کرد، به عنوان مثال، اجازه می‌دهد تا ترافیک امن SSL از طریق فایروال‌ها جریان یابد. با این حال، این ویژگی می‌تواند مورد سوء استفاده قرار گیرد و به پروتکل‌های مخرب اجازه می‌دهد تا از طریق تونل HTTP به یک سازمان سرازیر شوند.

SSL Tunneling Versus HTTPS Gateways

پروتکل HTTPS روی HTTP (SSL روی HTTPS) می‌تواند به روشی مشابه پروتکل‌های دیگر دروازه‌سازی شود: داشتن Gateway (به جای کلاینت) جلسه SSL را با سرور HTTPS راه دور آغاز می‌کند و سپس تراکنش HTTPS را در قسمت کلاینت انجام می‌دهد. پاسخ توسط پراکسی دریافت و رمزگشایی می‌شود و از طریق HTTP (نامن) برای کلاینت ارسال می‌شود. این راهی است که FTP ها با FTP مدیریت می‌کنند.





با این حال، این روش دارای چندین معایب است:

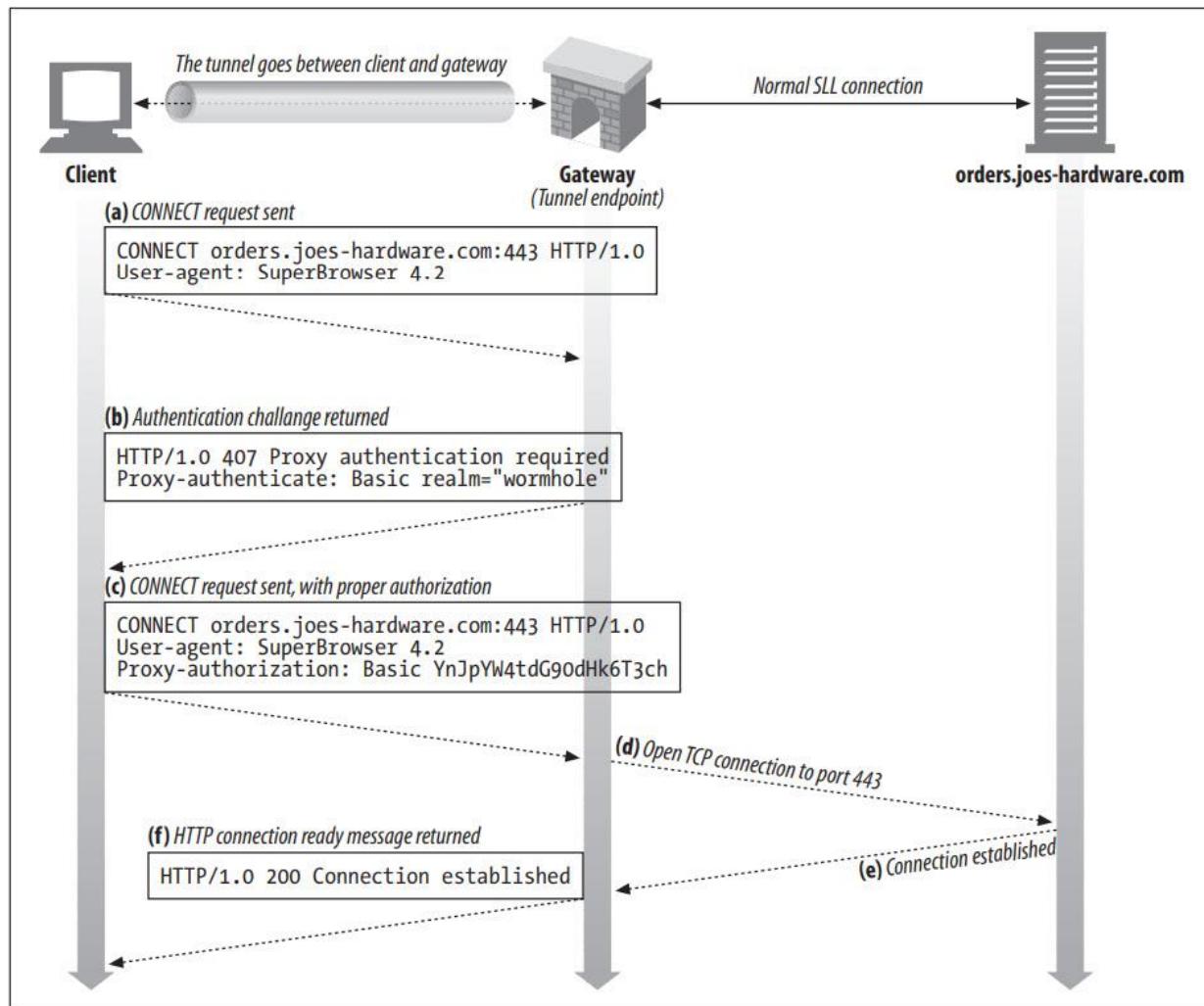
- اتصال کلاینت به دروازه HTTP عادی و نامن است.
- کلاینت نمی‌تواند احراز هویت کلاینت SSL (تأیید هویت بر اساس گواهی‌های X509) را در سرور راه دور انجام دهد، زیرا پراکسی طرف احراز هویت است.
- دروازه باید از اجرای کامل SSL پشتیبانی کند.

توجه داشته باشید که این مکانیسم، اگر برای تونل سازی SSL استفاده شود، نیازی به پیاده سازی SSL در پروکسی ندارد. جلسه SSL بین • کلاینت ایجاد کننده درخواست و وب سرور مقصد (ایمن) برقرار می‌شود. سرور پروکسی در این میان فقط داده‌های رمزگذاری شده را تونل می‌کند و هیچ بخش دیگری در تراکنش امن نمی‌گیرد.

Tunnel Authentication

ویژگی‌های دیگر HTTP نیز می‌تواند در صورت لزوم با تونل‌ها استفاده شود. به طور خاص، پشتیبانی از احراز هویت پراکسی می‌تواند با تونل‌ها برای احراز هویت حق استفاده از یک تونل استفاده شود. (client's right to (use a tunnel





Tunnel Security Considerations

به طور کلی، Tunnel Gateway نمی‌تواند تأیید کند که صحبت می‌شود واقعاً همان چیزی است که قرار است تونل کند یا خیر. بنابراین، برای مثال، کاربران مخرب ممکن است از تونل‌های در نظر گرفته شده برای SSL برای تونل کردن ترافیک بازی‌های اینترنتی از طریق فایروال شرکتی استفاده کنند، یا کاربران مخرب ممکن است از تونل‌ها برای باز کردن جلسات Telnet یا ارسال ایمیلی استفاده کنند که اسکنرهای ایمیل شرکتی را دور می‌زنند.

برای به حداقل رساندن سوء استفاده از تونل‌ها، Gateway باید تونل‌ها را فقط برای پورت‌های شناخته شده خاص، مانند ۴۴۳ برای HTTPS باز کند.

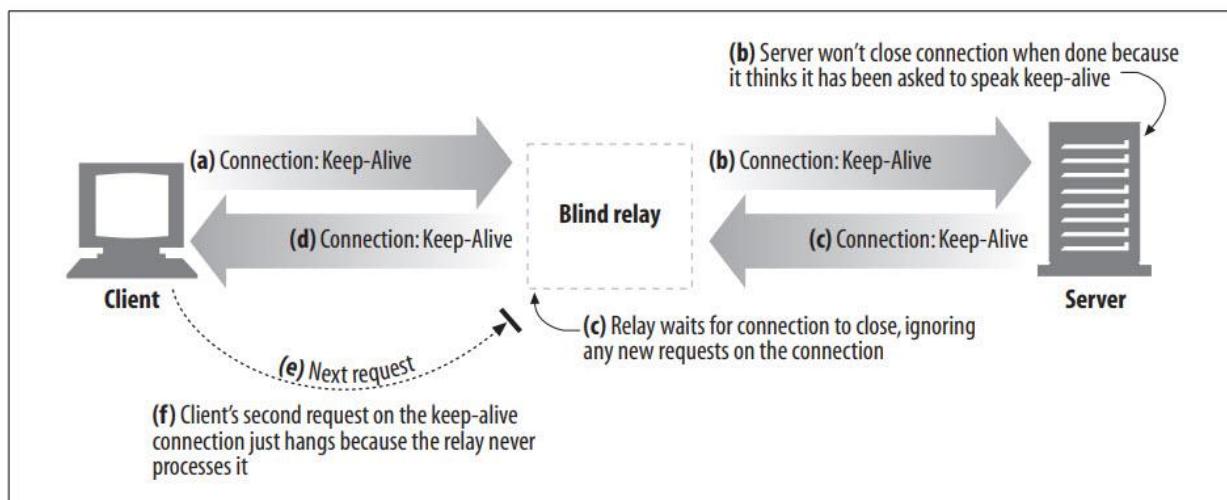


Relays

رله‌های HTTP پروکسی‌های ساده HTTP هستند که به طور کامل به مشخصات HTTP پایبند نیستند. رله‌ها کافی را برای ایجاد اتصالات پردازش می‌کنند، سپس کورکورانه بایت‌ها را به جلو ارسال می‌کنند.

از آنجایی که HTTP پیچیده است، گاهی اوقات پیاده‌سازی پروکسی‌های بی‌سابقه‌ای که فقط به‌طور کورکورانه ترافیک را جلو می‌برند، بدون انجام تمام منطق هدر و متده، مفید است. از آنجایی که رله‌های کور به راحتی قابل پیاده‌سازی هستند، گاهی اوقات از آن‌ها برای ارائه فیلتر ساده، تشخیص یا تغییر محتوا استفاده می‌شود. اما به دلیل پتانسیل جدی برای مشکلاتی که ممکن است استفاده از آن‌ها به همراه داشته باشد، باید با احتیاط فراوان به کار گرفته شوند.

یکی از مشکلات رایج (و بدnam) در برخی از پیاده‌سازی‌های رله‌های کور ساده، به پتانسیل آن‌ها برای قطع-keepalive مربوط می‌شود، زیرا هدر Connection را به درستی پردازش نمی‌کنند. این وضعیت در شکل زیر نشان داده شده است.



این چیزی است که در این شکل اتفاق می‌افتد:

در بخش a از شکل بالا، یک سرویس گیرنده وب پیامی را به رله ارسال می‌کند، از جمله هدر Connection: Keep-Alive، و در صورت امکان درخواست اتصال keep-alive می‌کند. کلاینت منتظر پاسخ می‌ماند تا بداند آیا درخواستش برای کانال keep-alive پذیرفته شده است یا خیر.

رله، درخواست HTTP را دریافت می‌کند، اما هدر Connection را نمی‌فهمد، بنابراین پیام را کلمه به hop-by- کلمه از زنجیره به سرور ارسال می‌کند (بخش b). با این حال، هدر Connection یک هدر-





hop است و فقط برای یک لینک حمل و نقل اعمال می‌شود و نباید به زنجیره منتقل شود. اتفاقات بد در شرف شروع شدن است!

در بخش b از شکل بالا، درخواست HTTP رله شده به وب سرور می‌رسد. هنگامی که وب سرور هدر Connection: Keep-Alive را دریافت می‌کند، به اشتباه نتیجه می‌گیرد که رله (که مانند هر کلاینت دیگری برای سرور به نظر می‌رسد) می‌خواهد با keep-alive صحبت کند! این برای وب سرور خوب است – می‌پذیرد که به طور keep-alive صحبت کند و یک هدر پاسخ Connection: Keep-Alive را (در بخش c) ارسال می‌کند. بنابراین، در این مرحله، وب سرور تصور می‌کند که با رله در حال صحبت با keep-alive است و به قوانین keep-alive پایبند است. اما رله چیزی در مورد keep-alive نمی‌داند.

در بخش d، رله پیام پاسخ سرور وب را به سمت کلاینت ارسال می‌کند و از هدر Connection: Keep-Alive از وب سرور عبور می‌کند. کلاینت این هدر را می‌بیند و فرض می‌کند که رله پذیرفته است که بماند. در این مرحله، هم کلاینت و هم سرور بر این باورند که دارند keep-alive صحبت می‌کنند، اما رله‌ای که با آن صحبت می‌کنند، موارد اولیه مربوط به keep-alive را نمی‌داند.

از آنجایی که رله چیزی در مورد keepalive نمی‌داند، تمام داده‌هایی را که دریافت می‌کند به کلاینت ارسال می‌کند و منظر می‌ماند تا سرور مبدعاً اتصال را بیندد. اما سرور مبدعاً اتصال را نمی‌بندد، زیرا معتقد است رله از سرور خواسته است که اتصال را باز نگه دارد! بنابراین، رله منظر بسته شدن اتصال خواهد ماند.

هنگامی که کلاینت پیام پاسخ را در (بخش d) دریافت می‌کند، مستقیماً به درخواست بعدی حرکت نموده و درخواست دیگری را به رله در اتصال keep-alive ارسال می‌کند (بخش e). رله‌های ساده معمولاً هرگز انتظار درخواست دیگری در همان اتصال را ندارند. در این حالت مرورگر فقط می‌چرخد و هیچ پیشرفتی ندارد.

راه‌هایی برای هوشمندتر کردن رله‌ها برای حذف این خطرات وجود دارد، اما هر گونه ساده‌سازی پراکسی‌ها خطر مشکلات عملکردی را به همراه دارد. اگر رله‌های HTTP ساده را برای هدف خاصی می‌سازید، مراقب نحوه استفاده از آن‌ها باشید. برای هر استقرار در مقیاس وسیع، باید قویاً به جای آن از یک سرور پروکسی واقعی و سازگار با HTTP استفاده کنید.





For More Information

<http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

Web Proxy Servers (Ari Luotonen, Prentice Hall Computer Books.)

<http://www.alternic.org/drafts/drafts-l-m/draft-luotonen-web-proxy-tunneling-01.txt>

<http://cgi-spec.golux.com>

<http://www.w3.org/TR/2001/WD-soap12-part0-20011217/>

Programming Web Services with SOAP (James Snell, Doug Tidwell, and Pavel Kulchenko, O'Reilly & Associates, Inc.)

<http://www.w3.org/TR/2002/WD-wsa-reqs-20020429>

Web Services Essentials (Ethan Cermai, O'Reilly & Associates, Inc.)





فصل نهم - Web Robots

ما تور خود را در معماری HTTP با نگاهی دقیق به Self-Animating User Agents (عوامل کاربر خود متحرک) به نام Web Robots ادامه می‌دهیم.

Web Robot ها برنامه‌های نرم افزاری هستند که مجموعه‌ای از تراکیش‌های وب را بدون تعامل انسانی خودکار می‌کنند. بسیاری از ربات‌ها از وبسایتی به وبسایت دیگر سرگردان هستند، محتوا را واکشی می‌کنند، لینک‌ها را دنبال می‌کنند و داده‌هایی را که پیدا می‌کنند پردازش می‌نمایند. به این نوع ربات‌ها نام‌های رنگارنگی مانند «spiders»، «crawlers»، «bots» و «worms» داده می‌شود، زیرا به طور خودکار وبسایت‌ها را کاوش می‌کنند.

در اینجا چند نمونه از ربات‌های وب آورده شده است:

Robots.txt ها هر چند دقیقه به سرورهای بازار سهام صادر می‌کنند و از داده‌ها برای ساخت نمودارهای روند قیمت سهام استفاده می‌کنند.

Robots-census، اطلاعات «سرشماری» را در مورد مقیاس و تکامل شبکه جهانی وب جمع‌آوری می‌کنند. آن‌ها در وب پرسه می‌زنند و تعداد صفحات را می‌شمارند و اندازه، زبان و نوع رسانه هر صفحه را ثبت می‌کنند.

Robots-search-engine، تمام اسنادی را که پیدا می‌کنند جمع‌آوری می‌کنند تا پایگاه داده‌های جستجو را ایجاد کنند.

Robots-comparison-shopping صفحات وب را از کاتالوگ فروشگاه‌های آنلاین جمع‌آوری می‌کنند تا پایگاه داده‌ای از محصولات و قیمت‌های آن‌ها بسازند.

Crawlers and Crawling

Web Crawler ها ربات‌هایی هستند که به صورت بازگشتی از وب‌ها عبور می‌کنند، ابتدا یک صفحه وب، سپس تمام صفحات وب را که آن صفحه به آن‌ها اشاره می‌کند، سپس تمام صفحات وب را که آن صفحات به آن‌ها اشاره می‌کنند، واکشی می‌کنند. هنگامی که یک ربات به صورت بازگشتی پیوندهای وب را دنبال می‌کند، به آن Spider یا Crawler می‌گویند.

Motors جستجوی اینترنتی از Crawler ها برای پرسه زدن در وب استفاده می‌کنند و تمام اسنادی را که با آن‌ها مواجه می‌شوند پس می‌کشند. سپس این اسناد برای ایجاد یک پایگاه داده قابل جستجو



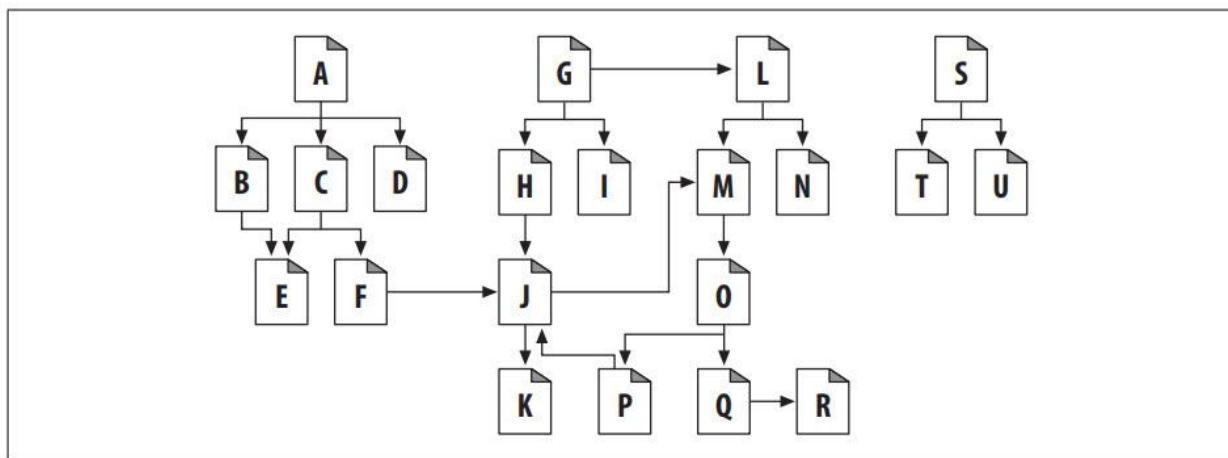


پردازش می‌شوند و به کاربران امکان می‌دهد اسنادی را که حاوی کلمات خاصی هستند پیدا کنند. با وجود میلیاردها صفحه وب برای یافتن و بازگرداندن، این Spider‌های موتور جستجو لزوماً برخی از پیچیده‌ترین ربات‌ها هستند. بیایید با جزئیات بیشتری به نحوه کار Crawler‌ها نگاه کنیم.

Where to Start: The Root Set

قبل از اینکه بتوانید Crawler گرسنه خود را آزاد کنید، باید به آن نقطه شروع بدهید. مجموعه اولیه URL‌هایی که Crawler شروع به بازدید از آن‌ها می‌کند، Root Set نامیده می‌شود. هنگام انتخاب یک Root Set، باید URL‌ها را از مکان‌های مختلف انتخاب کنید که همه پیوندها را Crawl نموده و در نهایت شما را به بیشتر صفحات وب مورد علاقه تان برساند.

یک Root Set خوب برای Crawl در وب در شکل زیر چیست؟



در وب واقعی، هیچ سند واحدی وجود ندارد که در نهایت به هر سند پیوند دهد. اگر با سند A در شکل بالا شروع کنید، می‌توانید به B، C، و D، سپس به E و F، سپس به L و سپس به K برسید. اما هیچ زنجیره‌ای از پیوندها از A به G یا از A به N وجود ندارد.

برخی از صفحات وب در این وب، مانند S، T، و U، تقریباً جدا شده اند، بدون اینکه هیچ پیوندی به آن‌ها اشاره کند. شاید این صفحات تنها جدید باشند و هنوز کسی آن‌ها را پیدا نکرده باشد. یا شاید واقعاً قدیمی یا مبهم هستند.

به طور کلی، برای پوشش دادن بخش بزرگی از وب، به صفحات زیادی در Root Set نیاز ندارید. در شکل بالا، برای دسترسی به تمام صفحات فقط به A، G و S در Root Set نیاز دارد.



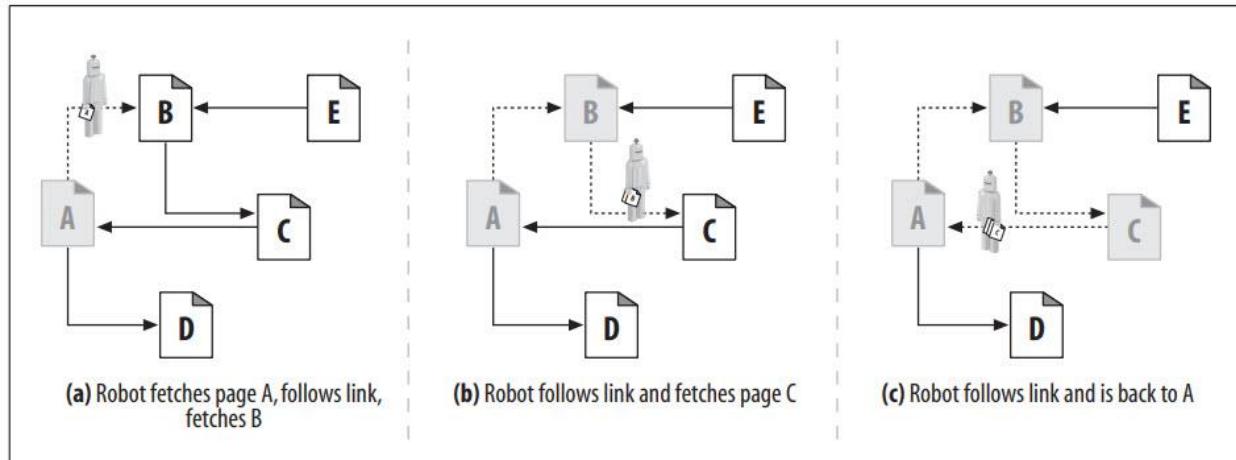
به طور معمول، یک Root Set خوب شامل وب سایتهاي بزرگ و محبوب (به عنوان مثال، <http://www.yahoo.com>)، لیستی از صفحات تازه ایجاد شده و لیستی از صفحات مبهم است که اغلب به آنها پیوند داده نمی‌شود. بسیاری از Crawler های تولیدی در مقیاس بزرگ، مانند آن‌هایی که توسط موتورهای جستجوی اینترنتی استفاده می‌شوند، راهی برای کاربران دارند تا صفحات جدید یا مبهم را در مجموعه اصلی ارسال کنند. این Root Set در طول زمان رشد می‌کند و لیست دانه (Seed List) برای هر خزیدن تازه است.

Extracting Links and Normalizing Relative Links

هنگامی که یک Crawler در وب حرکت می‌کند، دائمًا صفحات HTML را بازیابی می‌کند. باید لینک‌های URL را در هر صفحه‌ای که بازیابی می‌کند تجزیه نموده و آنها را به لیست صفحاتی که باید Crawl شوند اضافه کند. در حالی که Crawling در حال پیشرفت است، این لیست اغلب به سرعت گسترش می‌یابد، زیرا Crawler لینک‌های جدیدی را کشف می‌کند که باید کاوش شوند.

Cycle Avoidance

هنگامی که یک ربات یک وب را Crawl می‌کند، باید بسیار مراقب باشد که در یک حلقه یا چرخه گیر نکند. به در شکل زیر نگاه کنید:



در بخش a، ربات صفحه A را واکشی می‌کند، می‌بیند که A به B لینک دارد و صفحه B را واکشی می‌کند.

در بخش b، ربات صفحه B را واکشی می‌کند، می‌بیند که B به C لینک دارد و صفحه C را واکشی می‌کند.

در بخش c، ربات صفحه C را واکشی می‌کند و می‌بیند که C به A لینک دارد. اگر ربات دوباره صفحه A را واکشی کند، در یک چرخه به پایان می‌رسد و A, C, B, A, C, B, A را واکشی می‌کند.





ربات‌ها باید بدانند کجا بوده اند تا از چرخه دوری کنند. چرخه‌ها می‌توانند به تله‌های رباتی منجر شوند که می‌توانند پیشرفت ربات را متوقف یا کند نماید.

Loops and Dups

چرخه‌ها حداقل به سه دلیل برای Crawler ها بد هستند:

آن‌ها Crawler را وارد حلقه‌ای می‌کنند که می‌تواند در آن جا گیر کند. یک حلقه می‌تواند باعث شود یک Crawler با طراحی ضعیف به دور خود بچرخد و تمام وقت خود را صرف بارها و بارها واکشی صفحات مشابه کند. Crawler می‌تواند پهنانی باند شبکه زیادی را مصرف نموده و ممکن است به طور کامل نتواند هیچ صفحه دیگری را واکشی نماید.

در حالی که Crawler زنده به طور مکرر صفحات مشابه را واکشی می‌کند، سرور وب در طرف دیگر در حال ضربه خوردن است. اگر Crawler به خوبی متصل باشد، می‌تواند وب سایت را تحت تأثیر قرار دهد و از دسترسی هر کاربر واقعی به سایت جلوگیری کند. چنین انکار سرویسی می‌تواند دلیلی برای دعاوی حقوقی باشد.

حتی اگر حلقه کردن به خودی خود مشکلی نداشته باشد، Crawler تعداد زیادی صفحه تکراری (اغلب "dups" نامیده شده که با "loops" هم قافیه می‌شود) واکشی می‌کند. برنامه Crawler مملو از محتوای تکراری می‌شود که ممکن است برنامه را بی فایده کند. نمونه‌ای از این مورد یک موتور جستجوی اینترنتی است که صدها مورد مشابه را دقیقاً در همان صفحه برمی‌گرداند.

Trails of Breadcrumbs

متأسفانه، ردیابی جایی که بوده‌اید همیشه آسان نیست. در زمان نگارش این مقاله، میلیاردها صفحه وب مجزا در اینترنت وجود دارد، بدون احتساب محتوای تولید شده از Gateway های پویا.

اگر می‌خواهید بخش بزرگی از محتوای وب جهان را Crawl کنید، باید برای بازدید از میلیاردها URL آماده باشید. ردیابی آدرس‌هایی که بازدید شده‌اند می‌تواند بسیار چالش برانگیز باشد. به دلیل تعداد زیاد URL ها، باید از ساختارهای داده پیچیده استفاده کنید تا به سرعت مشخص کنید کدام URL ها را بازدید کرده‌اید. ساختارهای داده باید در سرعت و استفاده از حافظه کارآمد باشند.

سرعت مهم است زیرا صدها میلیون URL به ساختارهای جستجوی سریع نیاز دارند. جستجوی جامع در لیست‌های URL قابل بحث نیست. حداقل، یک ربات باید از درخت جستجو یا جدول هش استفاده کند تا بتواند به سرعت تشخیص دهد که آیا URL بازدید شده است یا خیر.





صدها میلیون URL نیز فضای زیادی را اشغال می‌کنند. اگر میانگین URL برابر ۴۰ کاراکتر باشد و یک ربات وب ۵۰۰ میلیون URL (فقط بخش کوچکی از وب) را جستجو کند، یک ساختار داده جستجو می‌تواند به ۲۰ گیگابایت یا بیشتر حافظه فقط برای نگهداری URL‌ها نیاز داشته باشد.

در اینجا چند تکنیک مفید وجود دارد که Crawler های وب در مقیاس بزرگ از آن‌ها برای مدیریت مکان بازدیدشان استفاده می‌کنند:

Trees and hash tables •

ربات‌های پیچیده ممکن است از درخت جستجو یا جدول هش برای پیگیری URL‌های بازدید شده استفاده کنند. این‌ها ساختارهای داده نرم افزاری هستند که جستجوی URL را بسیار سریعتر می‌کنند.

Lossy presence bit maps •

برای به حداقل رساندن فضای بزرگ از ساختارهای داده با تلفات مانند آرایه‌های بیت حضوری (Presence Bit Arrays) استفاده می‌کنند. هر URL توسط یک تابع هش به یک عدد اندازه ثابت تبدیل می‌شود و این عدد دارای یک "presence bit" مرتبط در یک آرایه است. هنگامی که یک URL فرض می‌کند که شد، بیت حضور مربوطه تنظیم می‌شود. اگر بیت حضور از قبل تنظیم شده باشد، Crawl قبلاً URL شده است.

Checkpoints •

طمئن شوید که لیست URL‌های بازدید شده را در دیسک ذخیره کرده اید. (در صورتی که برنامه ربات خراب شود)

Partitioning •

همانطور که وب رشد می‌کند، ممکن است تکمیل Crawl با یک ربات منفرد روی یک کامپیوتر غیر عملی شود. آن رایانه ممکن است حافظه، فضای دیسک، قدرت محاسباتی یا پهنای باند شبکه کافی برای تکمیل Crawl نداشته باشد. برخی از ربات‌های وب در مقیاس بزرگ از "مزرعه" ربات‌ها استفاده می‌کنند که هر کدام یک کامپیوتر جداگانه هستند و پشت سر هم کار می‌کنند. به هر ربات یک "slice" خاص از URL‌ها اختصاص داده می‌شود که مسئولیت آن را بر عهده دارد. ربات‌ها با هم برای Crawl در وب کار می‌کنند. آن‌ها ممکن است نیاز به برقراری ارتباط داشته باشند تا URL‌ها را به عقب و جلو ارسال کنند.



یک کتاب مرجع خوب برای پیاده سازی ساختارهای داده عظیم، Managing Gigabytes: Compressing and Indexing Documents and Images نوشته Witten و همکاران است. این کتاب پر از ترفندها و تکنیکهای مدیریت حجم زیاد داده است.

Aliases and Robot Cycles

حتی با ساختارهای داده‌ای مناسب، گاهی اوقات تشخیص اینکه آیا قبلاً از صفحه‌ای بازدید کرده‌اید یا خیر، دشوار است، و این به دلیل URL Aliasing است. اگر URL‌ها متفاوت به نظر برسند اما واقعاً به یک منبع اشاره کنند، آن گاه دو URL اصطلاحاً Aliases هستند.

جدول زیر چند راه ساده را نشان می‌دهد که URL‌های مختلف می‌توانند به یک منبع اشاره کنند:

First URL	Second URL	When aliased
a <i>http://www.foo.com/bar.html</i>	<i>http://www.foo.com:80/bar.html</i>	Port is 80 by default
b <i>http://www.foo.com/~fred</i>	<i>http://www.foo.com/%7Ffred</i>	%7F is same as ~
c <i>http://www.foo.com/x.html#early</i>	<i>http://www.foo.com/x.html#middle</i>	Tags don't change the page
d <i>http://www.foo.com/readme.htm</i>	<i>http://www.foo.com/README.HTM</i>	Case-insensitive server
e <i>http://www.foo.com/</i>	<i>http://www.foo.com/index.html</i>	Default page is <i>index.html</i>
f <i>http://www.foo.com/index.html</i>	<i>http://209.231.87.45/index.html</i>	<i>www.foo.com</i> has this IP address

Canonicalizing URLs

اکثر ربات‌های وب سعی می‌کنند تا نام مستعار آشکار را با «Canonicalizing» یا متعارف نمودن URL‌ها به شکل استاندارد حذف کنند. یک ربات ممکن است ابتدا هر URL را صورت زیر به یک فرم متعارف تبدیل کند:

- اگر پورت مشخص نشده باشد، "80:" را به نام میزبان اضافه کنند.
- تبدیل همه %XX کاراکترهای Escape شده به معادل کاراکترهایشان
- حذف تگ‌های #

این مراحل می‌توانند مشکلات همخوانی نشان داده شده در بخش a-c جدول بالا را حذف کنند. اما، بدون دانستن اطلاعات مربوط به وب سرور خاص، ربات هیچ راه خوبی برای جلوگیری از تکرارهای بخش‌های d-f جدول ندارد:

- ربات باید بداند که آیا وب سرور به حروف بزرگ و کوچک حساس نیست تا از نام مستعار در بخش d جدول جلوگیری کند.
- ربات باید پیکربندی صفحه فهرست وب سرور را برای این دایرکتوری بداند تا بداند آیا URL‌های بخش e جدول نام مستعار هستند یا خیر.





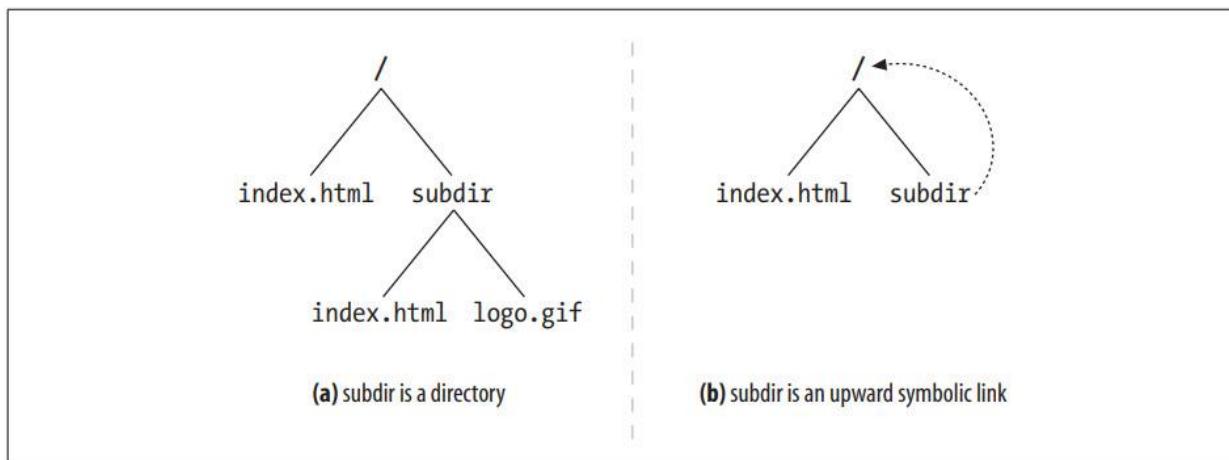
- ربات باید بداند که آیا وب سرور برای انجام Virtual Hosting پیکربندی شده است تا بداند آیا URL های بخش f جدول نام مستعار هستند، حتی اگر نام میزبان و آدرس IP مربوط به همان رایانه فیزیکی را بداند.

متعارف سازی URL می‌تواند Basic Syntactic Aliases را حذف کند، اما ربات‌ها با نام مستعار URL دیگری روبرو می‌شوند که از طریق تبدیل URL‌ها به فرم‌های استاندارد قابل حذف نیستند.

Filesystem Link Cycles

پیوندهای نمادین (Symbolic Links) در یک سیستم فایل می‌توانند نوعی چرخه به خصوص موزیانه ایجاد کنند، زیرا آن‌ها می‌توانند توهمند یک سلسله مراتب دایرکتوری عمیق بی‌نهایت را ایجاد کنند که هیچ کدام وجود ندارد. چرخه‌های پیوند نمادین معمولاً نتیجه یک خطای ناخواسته توسط مدیر سرور هستند، اما می‌توانند توسط "وب مسترهای شرور" به عنوان یک دام مخرب برای روبات‌ها ایجاد شوند.

شکل زیر دو فایل سیستم را نشان می‌دهد.



در بخش a تصویر، `subdir` یک دایرکتوری معمولی است. در بخش b تصویر، `subdir` یک پیوند نمادین است که به / اشاره دارد. در هر دو شکل، فرض کنید فایل `/index.html` حاوی یک پیوند به فایل `subdir/index.html` است.

با استفاده از سیستم فایل بخش a تصویر، یک Crawler وب ممکن است اقدامات زیر را انجام دهد:

1 .GET <http://www.foo.com/index.html>

Get `/index.html`, find link to `subdir/index.html`.





2 .GET <http://www.foo.com/subdir/index.html>

Get subdir/index.html, find link to subdir/logo.gif.

3 .GET <http://www.foo.com/subdir/logo.gif>

Get subdir/logo.gif, no more links, all done

اما در فایل سیستم بخش b تصویر، موارد زیر ممکن است رخ دهد:

1. GET <http://www.foo.com/index.html>

Get /index.html, find link to subdir/index.html.

2. GET <http://www.foo.com/subdir/index.html>

Get subdir/index.html, but get back same index.html.

3. GET <http://www.foo.com/subdir/subdir/index.html>

Get subdir/subdir/index.html.

4. GET <http://www.foo.com/subdir/subdir/subdir/index.html>

Get subdir/subdir/subdir/index.html.

مشکل بخش b تصویر این است که /subdir چرخه بازگشت به / است، اما از آنجا که URL ها متفاوت به نظر می‌رسند، ربات تنها از طریق URL نمی‌داند که اسناد یکسان هستند. ربات ناآگاه خطر ورود به یک حلقه را دارد. بدون نوعی تشخیص حلقه، این چرخه اغلب تا زمانی که طول URL از محدودیت‌های ربات یا سرور تجاوز کند ادامه خواهد داشت.

Dynamic Virtual Web Spaces

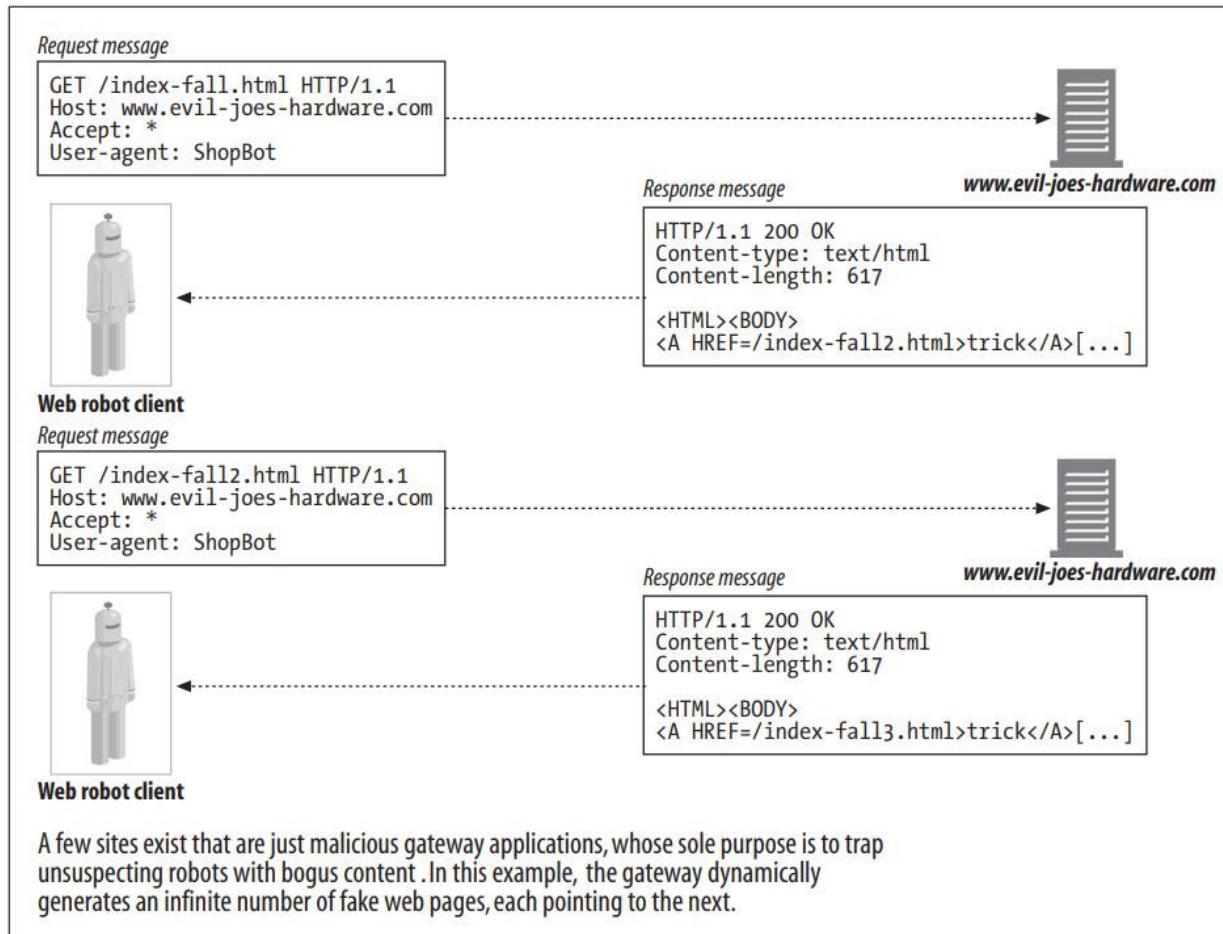
این امکان برای وب مسترهای مخرب وجود دارد که عمداً حلقه‌های Crawler پیچیده‌ای را برای به دام انداختن ربات‌های بی گناه و بی خبر ایجاد کنند. به طور خاص، انتشار URL که شبیه یک فایل معمولی بوده اما واقعاً یک برنامه Gateway می‌باشد، آسان است. این برنامه می‌تواند HTML را که حاوی لینک‌هایی به URL های خیالی در همان سرور است، به سرعت بالا ببرد. هنگامی که این URL های خیالی درخواست می‌شوند، سرور بد یک صفحه HTML جدید با URL های خیالی جدید می‌سازد.

وب سرور مخرب می‌تواند ربات ضعیف را به سفر آلیس در سرزمین عجایب در یک فضای وب مجازی بی‌نهایت ببرد، حتی اگر وب سرور واقعاً حاوی هیچ فایلی نباشد. حتی بدتر از آن، تشخیص چرخه برای





ربات بسیار دشوار است، زیرا URL ها و HTML هر بار می توانند بسیار متفاوت به نظر برسند. شکل زیر نمونه‌ای از یک وب سرور مخرب را نشان می‌دهد که محتوای جعلی تولید می‌کند.



معمولًاً، وب مسترهای خوش نیت ممکن است ناخواسته یک تله Crawler از طریق لینک‌های نمادین یا محتوای پویا ایجاد کنند. به عنوان مثال، یک برنامه تقویم مبتنی بر CGI را در نظر بگیرید که یک تقویم ماهانه و یک لینک به ماه بعد ایجاد می‌کند. یک کاربر واقعی برای همیشه به درخواست لینک ماه آینده ادامه نخواهد داد، اما روباتی که از ماهیت پویا محتوا آگاه نیست ممکن است به طور نامحدود به درخواست این منابع ادامه دهد.

Avoiding Loops and Dups

هیچ روشی برای اجتناب از تمام چرخه‌ها وجود ندارد. در عمل، ربات‌هایی که به خوبی طراحی شده‌اند، باید مجموعه‌ای از اکتشافات را برای جلوگیری از چرخه‌ها شامل شوند.

به طور کلی، هرچه یک Crawler مستقل‌تر باشد (نظرارت انسانی کمتر)، احتمال بروز مشکل بیشتر است. مقداری Trade-off وجود دارد که پیاده‌کنندگان ربات باید آن را انجام دهند - این Heuristic‌ها





می‌توانند به جلوگیری از مشکلات کمک کنند، اما تا حدودی زیان بار (*lossy*) هستند، زیرا در نهایت می‌توانید محتوای معتبری را که مشکوک به نظر می‌رسد نادیده بگیرید.

برخی از تکنیک‌هایی که ربات‌ها برای رفتار بهتر (در یک وب پر از خطرات ربات) استفاده می‌کنند عبارتند از:

Canonicalizing URLs

با تبدیل URL‌ها به فرم استاندارد از نام مستعار نحوی (*Syntactic Aliases*) اجتناب کنید.

Breadth-first crawling

URL‌ها مجموعه بزرگی از URL‌های بالقوه برای Crawler در هر زمان دارند. با برنامه ریزی Crawler‌ها برای بازدید به روی گسترده، در سراسر وب سایتها، می‌توانید تأثیر چرخه‌ها را به حداقل برسانید. حتی اگر به تله ربات برخورد کنید، باز هم می‌توانید صدها هزار صفحه را از وب سایتها دیگر واکشی کنید قبل از اینکه برای واکشی یک صفحه از چرخه بازگردد.

Throttling

تعداد صفحاتی را که ربات می‌تواند از یک وب سایت در یک دوره زمانی دریافت کند محدود کنید. اگر ربات به یک چرخه برخورد کند و به طور مداوم سعی کند به نام مستعار از یک سایت دسترسی پیدا کند، می‌توانید تعداد کل تکرارهای تولید شده و تعداد کل دسترسی‌ها به سرور را با Throttling محدود کنید.

Limit URL size

ربات ممکن است از Crawl نمودن URL‌های بیش از یک طول معین خودداری کند (۱ کیلوبایت رایج است). اگر یک چرخه باعث افزایش اندازه URL شود، محدودیت طول در نهایت چرخه را متوقف می‌کند. برخی از سرورهای وب با دادن URL طولانی از کار می‌افتد و ربات‌هایی که در چرخه افزایش URL گرفتار می‌شوند می‌توانند باعث از کار افتادن برخی از سرورهای وب شوند. این ممکن است باعث شود مدیران وب سایت ربات را به عنوان یک مهاجم انکار سرویس تعبیر کنند.

به عنوان یک احتیاط، این تکنیک مطمئناً می‌تواند منجر به از دست رفتن محتوا شود. امروزه بسیاری از سایتها از URL‌ها برای کمک به مدیریت وضعیت کاربر استفاده می‌کنند (به عنوان مثال، ذخیره شناسه‌های کاربر در URL‌های ارجاع شده در یک صفحه). اندازه URL می‌تواند راهی دشوار برای محدود کردن Crawling باشد. با این حال، می‌تواند یک Flag عالی برای کاربر فراهم کند تا آنچه را که در یک سایت خاص اتفاق می‌افتد بررسی کند که این کار با ثبت یک خطای هر زمان که URL‌های درخواستی به اندازه معینی رسیدند، انجام می‌شود.





URL/site blacklist

فهرستی از سایتها و آدرس‌های اینترنتی شناخته شده را که با چرخه‌ها و تله‌های ربات مطابقت دارند، حفظ کنید و مانند طاعون از آن‌ها دوری کنید. با پیدا شدن مشکلات جدید، آن‌ها را به لیست سیاه اضافه کنید.

این مستلزم یک اکشن انسانی است. با این حال، اکثر Crawler‌های مقیاس بزرگ که امروزه تولید می‌شوند، نوعی لیست سیاه دارند همچنین می‌توان از لیست سیاه برای جلوگیری از برخی سایتها که در مورد Crawling سر و صدا ایجاد کرده‌اند، استفاده کرد.

Pattern detection

چرخه‌های ناشی از Symlink‌های سیستم فایل و پیکربندی‌های نادرست مشابه، از برخی الگوهای پیروی می‌کنند. به عنوان مثال، URL ممکن است با اجزای تکراری رشد کند. برخی از ربات‌ها URL‌های دارای اجزای تکرار شونده را به عنوان چرخه‌های بالقوه می‌بینند و از Crawl نمودن URL‌هایی با بیش از دو یا سه جزء تکراری خودداری می‌کنند.

همه تکرارها فوری نیستند (به عنوان مثال، ".../subdir/subdir/subdir"). ممکن است چرخه‌های دوره ۲ یا فواصل دیگر مانند ".../subdir/images/subdir/images/subdir/images/" وجود داشته باشد. برخی از ربات‌ها به دنبال الگوهای تکرار شونده از چند دوره مختلف هستند.

Content fingerprinting

روشی مستقیم‌تر برای شناسایی موارد تکراری است که توسط برخی از Crawler‌های Fingerprinting وب پیچیده‌تر استفاده می‌شود. ربات‌هایی که از Fingerprinting محتوا استفاده می‌کنند، بایت‌های موجود در صفحه را می‌گیرند و یک Checksum را محاسبه می‌کنند. این Checksum نمایشی فشرده از محتوای صفحه است. اگر یک ربات صفحه‌ای را واکشی کند که Checksum آن را قبلاً دیده است، پیوندهای صفحه Crawl نمی‌شوند—اگر ربات قبلًا محتوای صفحه را دیده باشد، Crawl پیوندهای صفحه را قبلاً آغاز کرده است.

تابع Checksum باید به گونه‌ای انتخاب شود تا احتمال اینکه دو صفحه مختلف دارای Checksum یکسان باشند، کاهش پیدا کند. توابع Message Digest مانند MD5 برای Fingerprinting محبوب هستند.





از آنجایی که برخی از سرورهای وب به صورت پویا صفحات را در لحظه تغییر می‌دهند، ربات‌ها گاهی اوقات بخش‌های خاصی از محتوای صفحه وب، مانند لینک‌های تعبیه شده (Embedded Links) را از محاسبه Checksum حذف می‌کنند. با این حال، سمت سرور پویا شامل سفارشی کردن محتوای صفحه دلخواه (افزودن تاریخ، شمارنده‌های دسترسی و غیره) ممکن است از تشخیص تکراری جلوگیری کند.

Human monitoring

وب یک مکان وحشی است. ربات شجاع شما در نهایت به مشکلی برخورد خواهد کرد که هیچ یک از تکنیک‌های شما آن را حل نمی‌کند. تمام ربات‌های با کیفیت باید با تشخیص و لاغ طراحی شوند، بنابراین انسان‌ها می‌توانند به راحتی پیشرفت ربات را زیر نظر بگیرند و در صورت وقوع اتفاق غیرعادی به سرعت به آن‌ها هشدار داده شود. در برخی موارد، شهروندان شبکه خشمگین با ارسال ایمیل‌های ناخوشایند مشکل را برای شما بر جسته می‌کنند.

اکتشافات عنکبوتی خوب برای Crawling مجموعه داده‌هایی به وسعت وب همیشه در حال انجام است. قوانین در طول زمان ساخته می‌شوند و با اضافه شدن انواع جدیدی از منابع به وب سازگار می‌شوند. لازم به ذکر است که قوانین خوب همیشه در حال تغییر هستند.

بسیاری از Crawler‌های کوچک‌تر و سفارشی‌تر برخی از این مسائل را کنار می‌گذارند، زیرا منابع (سرورها، پهنانی باند شبکه، و غیره) که توسط یک Crawler خطأکار تحت تأثیر قرار می‌گیرند، قابل مدیریت هستند، یا حتی احتمالاً تحت کنترل شخصی هستند که Crawling را انجام می‌دهند (مانند یک سایت اینترنت). این Crawler‌ها برای جلوگیری از مشکلات به نظارت بیشتر انسانی متکی هستند.

Robotic HTTP

ربات‌ها هیچ تفاوتی با سایر برنامه‌های سرویس گیرنده HTTP ندارند. آن‌ها نیز باید از قوانین مشخصات HTTP پیروی کنند. رباتی که درخواست‌های HTTP را ایجاد نمود و خود را به عنوان کلاینت HTTP/1.1 تبلیغ می‌کند، باید از هدرهای درخواست HTTP مناسب استفاده کند.

بسیاری از ربات‌ها سعی می‌کنند حداقل مقدار HTTP مورد نیاز برای درخواست محتوای مورد نظر خود را پیاده سازی کنند. این می‌تواند منجر به مشکلاتی شود. با این حال، بعید است که این رفتار به این زودی‌ها تغییر کند. در نتیجه، بسیاری از ربات‌ها درخواست‌های HTTP/1.0 را ایجاد می‌کنند، زیرا آن پروتکل الزامات کمی دارد.





Identifying Request Headers

علیرغم حداقل مقدار HTTP که ربات‌ها تمایل دارند از آن پشتیبانی کنند، اکثر آن‌ها برخی از هدرهای شناسایی را پیاده‌سازی و ارسال می‌کنند – به ویژه هدر User-Agent مربوط به HTTP توصیه می‌شود که پیاده‌کننده‌های ربات برخی از اطلاعات اولیه هدر را ارسال کنند تا سایت را از قابلیت‌های ربات، هویت ربات و محل پیدایش آن مطلع کنند.

این اطلاعات مفیدی هم برای ردیابی صاحب Crawler خطاکار و هم برای دادن اطلاعاتی به سرور در مورد نوع محتوایی است که ربات می‌تواند مدیریت کند. برخی از هدرهای شناسایی اولیه که پیاده‌کنندگان ربات تشویق به پیاده‌سازی می‌شوند عبارتند از:

User-Agent: نام رباتی را که درخواست می‌کند به سرور می‌گوید.

From: آدرس ایمیل کاربر/مدیر ربات را ارائه می‌دهد.

Accept: به سرور می‌گوید که چه نوع رسانه‌ای برای ارسال مناسب است. در این بخش اطمینان حاصل می‌شود که ربات فقط محتوای مورد علاقه خود را دریافت می‌کند (متن، تصاویر و غیره).

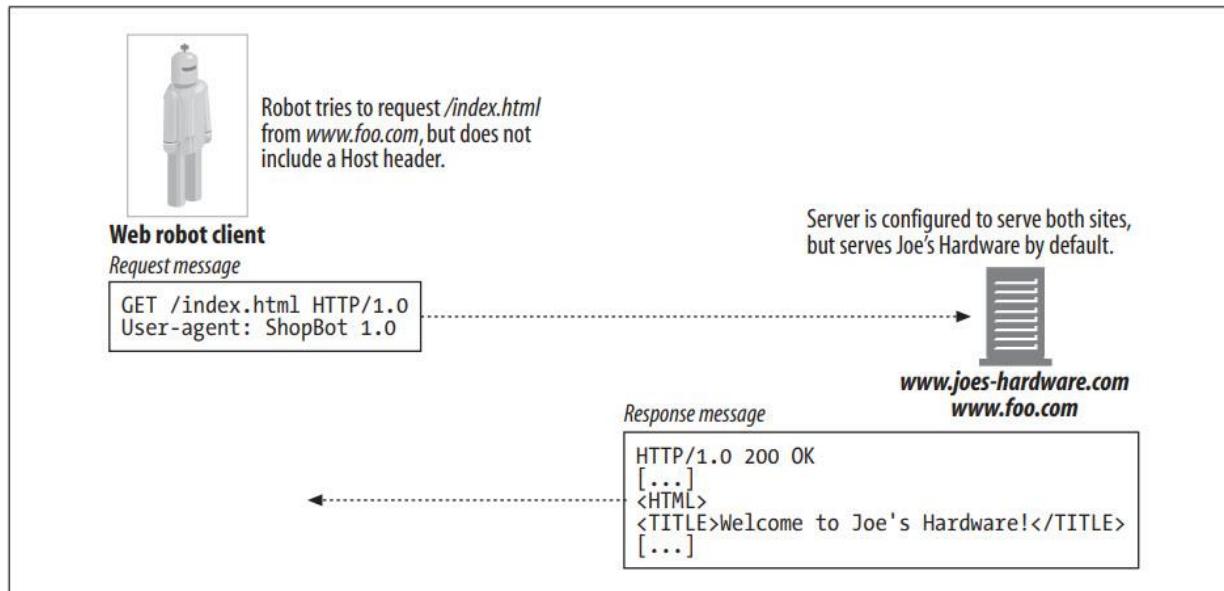
Referer: نشانی اینترنتی سندی را ارائه می‌کند که حاوی URL درخواست فعلی است.

Virtual Hosting

پیاده‌کننده‌های ربات باید از هدر Host پشتیبانی کنند. با توجه به رایج شدن میزبانی مجازی، عدم درج Host Header در درخواست‌ها می‌تواند منجر به شناسایی محتوای اشتباه توسط ربات‌ها با یک URL خاص شود. HTTP/1.1 به این دلیل نیاز به استفاده از هدر Host دارد.

اکثر سرورها به صورت پیش فرض برای سرویس دهی به یک سایت خاص پیکربندی شده اند. بنابراین، یک Crawler بدون هدر Host می‌تواند به سروری که دو سایت را ارائه می‌کند، مانند موارد موجود در شکل زیر (www.foo.com و www.joes-hardware.com) درخواست بدهد و اگر سرور به‌طور پیش‌فرض برای سرویس www.joes-hardware.com پیکربندی شده باشد (و به هدر Host نیاز ندارد)، منجر به دریافت محتوا از سایت Joe's Hardware می‌شود. بدتر از آن، Crawler در واقع فکر می‌کند که محتوای Joe's Hardware از www.foo.com است. مطمئنم اگر استنادی از دو سایت با دیدگاه‌های سیاسی قطبی یا سایر دیدگاه‌ها از یک سرور ارائه شود، می‌توانید به موقعیت‌های ناگوارتری فکر کنید.





Conditional Requests

با توجه به عظمت برخی از تلاش‌های رباتیک، اغلب معقول است که مقدار محتوایی را که یک ربات بازیابی می‌کند به حداقل برسانیم. همانطور که در مورد ربات‌های موتور جستجوی اینترنتی، با میلیاردها صفحه وب برای دانلود بالقوه، بازیابی مجدد محتوا تنها در صورتی منطقی است که تغییر کرده باشد.

برخی از این ربات‌ها درخواست‌های HTTP مشروط را پیاده‌سازی می‌کنند، برچسب‌های زمانی یا موجودیت را مقایسه می‌کنند تا ببینند آیا آخرین نسخه‌ای که بازیابی کرده‌اند به روزرسانی شده است یا خیر. این بسیار شبیه به روشی است که یک HTTP Cache اعتبار کپی محلی منبعی که قبلاً واکشی شده را بررسی می‌کند.

Response Handling

از آنجایی که بسیاری از ربات‌ها در درجه اول به دریافت محتوای درخواستی از طریق متدهای ساده GET علاقمند هستند، اغلب در نحوه رسیدگی به پاسخ کار زیادی انجام نمی‌دهند. با این حال، ربات‌هایی که از برخی ویژگی‌های HTTP (مانند درخواست‌های شرطی) استفاده می‌کنند و همچنین آن‌هایی که می‌خواهند بهتر کاوش کنند و با سرورها تعامل داشته باشند، باید بتوانند انواع مختلف پاسخ‌های HTTP را مدیریت کنند.

Status codes

به طور کلی، ربات‌ها باید بتوانند حداقل کدهای وضعیت رایج یا مورد انتظار را مدیریت کنند. همه ربات‌ها باید کدهای وضعیت HTTP مانند 200 OK و 404 Not Found را درک کنند. آن‌ها همچنین باید بتوانند با کدهای وضعیتی که به صراحت بر اساس دسته بندی کلی پاسخ درک نمی‌کنند، مقابله کنند.





توجه به این نکته ضروری است که برخی از سرورها همیشه کدهای خطای مناسب را برنمی‌گردانند. برخی از سرورها حتی کد 200 OK را با متن پیام که خطا را توصیف می‌کند، برنمی‌گردانند! انجام کارهای زیادی در این مورد سخت است - این فقط چیزی است که اجراءکنندگان باید از آن آگاه باشند.

Entities

همراه با اطلاعات جاسازی شده در هدرهای HTTP، ربات‌ها می‌توانند به دنبال اطلاعات در خود موجودیت بگردند. تگ‌های متا HTML، مانند تگ متا http-equiv، ابزاری برای نویسنده‌گان محتوا برای جاسازی اطلاعات اضافی درباره منابع هستند.

<meta http-equiv="Refresh" content="1;URL=index.html">

این تگ به گیرنده دستور می‌دهد تا با سند به گونه‌ای رفتار کند که گویی هدر پاسخ HTTP آن حاوی یک هدر Refresh HTTP با مقدار "URL=index.html,1" است.

برخی از سرورها در واقع محتويات صفحات HTML را قبل از ارسال آن‌ها تجزیه می‌کنند و دستورالعمل‌های http-equiv را به عنوان هدر درج می‌کنند. با این حال، برخی هم این کار را نمی‌کنند. پیاده‌کننده‌های ربات ممکن است بخواهند عنصر HEAD HTML را برای جستجوی اطلاعات http-equiv اسند کنند.

User-Agent Targeting

مدیران وب باید در نظر داشته باشند که بسیاری از ربات‌ها از سایتها که آن‌ها بازدید می‌کنند و بنابراین باید از آن‌ها انتظار درخواست را داشته باشند. بسیاری از سایتها، محتوا را برای User-Agent‌های مختلف بهینه می‌کنند و سعی می‌کنند انواع مرورگرها را شناسایی کنند تا از پشتیبانی از ویژگی‌های مختلف سایت اطمینان حاصل کنند. با انجام این کار، سایتها به جای محتوا، صفحات خطای خطا را در اختیار ربات‌ها قرار می‌دهند. انجام جستجوی متنی برای عبارت «مرورگر شما از فریم‌ها پشتیبانی نمی‌کند» در برخی از موتورهای جستجو، فهرستی از نتایج را برای صفحات خطای حاوی این عبارت به دست می‌دهد، در حالی که در واقع کلاینت HTTP اصلاً یک مرورگر نبود، بلکه یک ربات بوده است.

مدیران سایت باید یک استراتژی برای رسیدگی به درخواست‌های ربات برنامه ریزی کنند. به عنوان مثال، به جای محدود کردن توسعه محتوا خود به پشتیبانی مرورگر خاص، می‌توانند صفحاتی را برای مرورگرها و ربات‌های بدون ویژگی (non-feature) ایجاد کنند. حداقل، آن‌ها باید انتظار داشته باشند که ربات‌ها از سایتها که آن‌ها بازدید نموده و در هنگام بازدید از آن‌ها غافل نشوند.





Misbehaving Robots

راههای زیادی وجود دارد که ربات‌های سرکش می‌توانند باعث آشفتگی شوند. در اینجا چند اشتباہی که ربات‌ها می‌توانند مرتكب شوند و تأثیر اعمال نادرست آن‌ها آورده شده است:

Runaway robots

ربات‌ها درخواست‌های HTTP را بسیار سریع‌تر از وب‌گردهای انسانی ارسال می‌کنند و معمولاً روی رایانه‌های سریع با لینک‌های شبکه سریع اجرا می‌شوند. اگر یک ربات دارای یک خطای منطقی برنامه نویسی باشد، یا در چرخه‌ای گیر بیفتد، می‌تواند بار شدیدی را بر روی یک وب‌سور پرتاپ کند - احتمالاً به اندازه‌ای که منجر به Overload سرور و عدم دسترسی سرویس برای دیگران شود. همه نویسندهای ربات باید در طراحی ربات‌های خود به این موضوع توجه زیادی داشته باشند.

Stale URLs

برخی از ربات‌ها از لیست URL‌ها بازدید می‌کنند. این لیست‌ها می‌توانند قدیمی باشند. اگر یک وب‌سایت تغییر بزرگی در محتوای خود ایجاد کند، ربات‌ها ممکن است تعداد زیادی URL غیر موجود را درخواست کنند. این امر برخی از مدیران وب سایت را آزار می‌دهد، که دوست ندارند گزارش‌های خطای آن‌ها با درخواست‌های دسترسی برای اسناد موجود پر شود و دوست ندارند که ظرفیت وب‌سور آن‌ها با هزینه‌های سربار صفحات خطا کاهش یابد.

Long, wrong URLs

در نتیجه چرخه‌ها و خطاهای برنامه نویسی، ربات‌ها ممکن است URL‌های بزرگ و بی معنی را از وب‌سایت‌ها درخواست کنند. اگر URL به اندازه کافی طولانی باشد، ممکن است عملکرد وب‌سور را کاهش دهد، گزارش‌های دسترسی به سرور وب را به هم ریخته و حتی باعث از کار افتادن سرورهای وب شکننده شود.

Nosy robots

برخی از روبات‌ها ممکن است URL‌هایی دریافت کنند که به داده‌های خصوصی اشاره می‌کنند و آن داده‌ها را به راحتی از طریق موتورهای جستجوی اینترنتی و سایر برنامه‌ها، در دسترس قرار می‌دهند. اگر صاحب داده‌ها به طور فعال صفحات وب را تبلیغ نکرده باشد، ممکن است انتشار رباتیک را در بهترین حالت یک مزاحم و در بدترین حالت تجاوز به حریم خصوصی بداند.





معمولًاً این اتفاق می‌افتد زیرا یک لینک به محتوای «خصوصی» که ربات دنبال می‌کرد از قبل وجود دارد (یعنی محتوا آنقدر که مالک فکر می‌کرد مخفی نیست، یا مالک فراموش کرده است که یک لینک قبلی را حذف کند).

پیاده‌کننده‌های رباتی که حجم زیادی از داده‌ها را از وب بازیابی می‌کنند باید بدانند که روبات‌های آن‌ها احتمالاً در نقطه‌ای می‌توانند داده‌های حساس را بازیابی کنند - داده‌هایی که پیاده‌کننده سایت هرگز قصد نداشت از طریق اینترنت قابل دسترسی باشد. این داده‌های حساس می‌توانند شامل فایل‌های رمز عبور یا حتی اطلاعات کارت اعتباری باشد.

واضح است که مکانیزمی برای نادیده گرفتن محتوا پس از اشاره به این موضوع (و حذف آن از هر فهرست جستجو یا آرشیو) مهم است. کاربران موتورهای جستجوی مخرب و آرشیو، از توانایی‌های Web Crawler‌ها در مقیاس بزرگ برای یافتن محتوا استفاده می‌کنند—بعضی از موتورهای جستجو، مانند Google، در واقع نمایش‌هایی از صفحاتی را که Crawl کرده اند بایگانی می‌کنند، بنابراین حتی اگر محتوا حذف شود، هنوز برای مدتی می‌توان محتوا را یافته و به آن دسترسی داشت.

Dynamic gateway access

Robots ها همیشه نمی‌دانند به چه چیزی دسترسی دارند. یک ربات ممکن است یک URL را واکشی کند که محتوای آن از یک برنامه Gateway می‌آید. در این مورد، داده‌های به دست‌آمده ممکن است برای هدف خاص باشند و ممکن است محاسبه آن گران باشد. بسیاری از مدیران وب‌سایتها از ربات‌های ساده لوح درخواست اسناد که از Gateway ها می‌آیند، خوششان نمی‌آید.

Excluding Robots

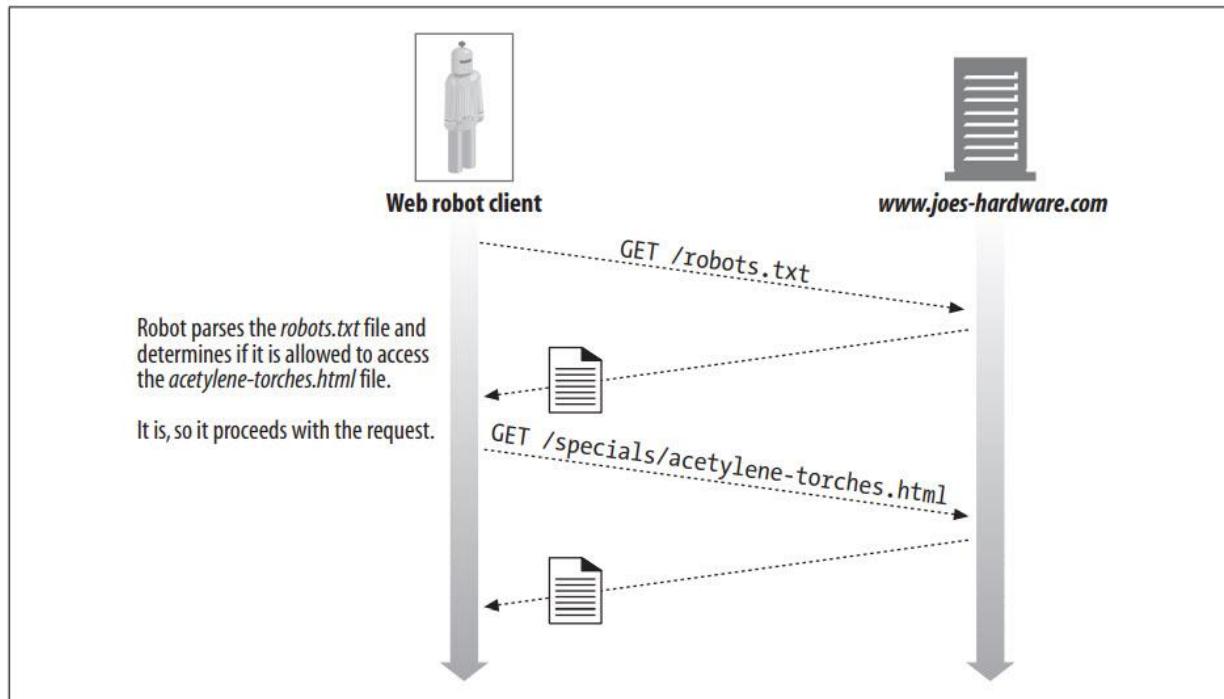
جامعه ربات مشکلاتی را که دسترسی به وب سایت رباتیک می‌تواند ایجاد کند، در کرده است. در سال ۱۹۹۴، یک تکنیک ساده و داوطلبانه پیشنهاد شد تا ربات‌ها را از جایی که به آن‌ها تعلق ندارند دور نگه دارد و مکانیزمی برای کنترل بهتر رفتارشان برای مدیران وب‌سایت فراهم کند. این استاندارد "Robots Exclusion Standard" یا استاندارد حذف ربات‌ها نام داشت، اما اغلب فقط robots.txt نامیده می‌شود، فایلی که اطلاعات کنترل دسترسی در آن ذخیره می‌شود.

ایده robots.txt ساده است. هر وب سرور می‌تواند یک فایل اختیاری به نام robots.txt را در مسیر وب سرور یا وب اپلیکیشن ایجاد کند. این فایل حاوی اطلاعاتی در مورد این است که چه رباتی می‌تواند به چه بخش‌هایی از سرور دسترسی داشته باشد. اگر روباتی از این استاندارد داوطلبانه پیروی کند، قبل از دسترسی به هر منبع دیگری از آن سایت، فایل robots.txt را از وب سایت درخواست می‌کند.





به عنوان مثال، ربات موجود در شکل زیر می‌خواهد `acetylene-torches.html` را با آدرس مشخص، از `Joe's Hardware` دانلود کند. قبل از اینکه ربات بتواند صفحه را درخواست کند، باید فایل `robots.txt` را بررسی کند تا ببیند آیا مجوز واکشی این صفحه را دارد یا خیر. در این مثال، فایل `robots.txt` ربات را مسدود نمی‌کند، بنابراین ربات به صفحه دسترسی خواهد داشت و امکان دریافت آن را دارد.



The Robots Exclusion Standard

استاندارد حذف روبات‌ها یک استاندارد موقت است. در زمان نگارش این مقاله، هیچ نهاد رسمی استاندارد، مالک این استاندارد نیست و فروشنده‌گان زیر مجموعه‌های مختلفی از استاندارد را پیاده سازی می‌کنند. با این حال، برخی از توانایی‌های مدیریت دسترسی ربات‌ها به وبسایتها، حتی اگر ناقص باشد، بهتر از هیچ چیز است و اکثر فروشنده‌گان اصلی و `Crawler`‌های موتور جستجو از استاندارد `Exclusion` پشتیبانی می‌کنند.

سه نسخه از استاندارد حذف روبات‌ها وجود دارد، اگرچه که نام این نسخه‌ها به خوبی تعریف نشده است. ما شماره‌گذاری نسخه نشان داده شده در جدول زیر را اتخاذ می‌کنیم.





Version	Title and description	Date
0.0	A Standard for Robot Exclusion—Martijn Koster's original <i>robots.txt</i> mechanism with Disallow directive	June 1994
1.0	A Method for Web Robots Control—Martijn Koster's IETF draft with additional support for Allow	Nov. 1996
2.0	An Extended Standard for Robot Exclusion—Sean Conner's extension including regex and timing information; not widely supported	Nov. 1996

امروزه اکثر ربات‌ها از استانداردهای v0.0 یا v1.0 استفاده می‌کنند. استاندارد v2.0 بسیار پیچیده‌تر است و به طور گسترده مورد استفاده قرار نگرفته است و ممکن است هرگز مورد استفاده قرار نگیرد. ما در اینجا بر روی استاندارد v1.0 تمرکز خواهیم کرد، زیرا استفاده گسترده‌ای دارد و کاملاً با نسخه v0.0 سازگار است.

Web Sites and robots.txt Files

قبل از بازدید از هر URL در یک وب سایت، یک ربات باید فایل robots.txt در وب سایت را در صورت وجود بازیابی و پردازش کند. یک منبع robots.txt برای کل وب سایت وجود دارد که با نام میزبان و پورت تعريف شده است. اگر سایت به صورت مجازی میزبانی شود، می‌تواند مانند هر فایل دیگری برای هر docroot مجازی یک فایل robots.txt متفاوت وجود داشته باشد.

در حال حاضر، هیچ راهی برای نصب فایل‌های robots.txt «محلى» در زیر شاخه‌های جداگانه یک وب‌سایت وجود ندارد. مدیر وب سایت مسئول ایجاد یک فایل robots.txt انبوه است که قوانین حذف را برای تمام محتوای وب سایت شرح می‌دهد.

Fetching robots.txt

ربات‌ها منبع robots.txt را با استفاده از متد HTTP GET، مانند هر فایل دیگری در سرور وب، واکشی می‌کنند. سرور فایل robots.txt را، در صورت وجود، به صورت متن/متن ساده برمی‌گرداند. اگر سرور با کد وضعیت 404 Not Found پاسخ دهد، ربات می‌تواند فرض کند که هیچ محدودیتی برای دسترسی رباتیک وجود ندارد و می‌تواند هر فایلی را درخواست کند.

ربات‌ها باید اطلاعات شناسایی را در هدرهای User-Agent و From ارسال کنند تا به مدیران سایت کمک کنند دسترسی‌های رباتیک را ردیابی کنند و در صورت نیاز مدیر سایت به درخواست یا شکایت از ربات، اطلاعات تماس را ارائه دهند. در اینجا یک نمونه درخواست HTTP Crawler از یک ربات وب تجاری وجود دارد:

GET /robots.txt HTTP/1.0

Host: www.joes-hardware.com





User-Agent: Slurp/2.0

Date: Wed Oct 3 20:22:48 EST 2001

Response codes

بسیاری از وب سایتها منبع robots.txt ندارند، اما ربات این را نمی‌داند. باید سعی کند منبع robots.txt را از هر سایتی دریافت کند. ربات بسته به نتیجه بازیابی robots.txt اقدامات مختلفی انجام می‌دهد:

- اگر سرور با وضعیت موفقیت آمیز پاسخ دهد (کد وضعیت HTTP 2XX)، ربات باید محتوا را تجزیه کند و قوانین حذف را برای واکشی از آن سایت اعمال کند.
- اگر پاسخ سرور نشان دهد که منبع وجود ندارد (کد وضعیت HTTP 404)، ربات می‌تواند فرض کند که هیچ قانونی فعال نبوده و دسترسی به سایت توسط robots.txt محدود نشده است.
- اگر پاسخ سرور محدودیت دسترسی را نشان دهد (کد وضعیت HTTP 401 یا 403)، ربات باید دسترسی به سایت را کاملاً محدود در نظر بگیرد.
- اگر تلاش درخواست منجر به شکست موقت شود (کد وضعیت HTTP 503)، ربات باید بازدید از سایت را تا زمان بازیابی منبع به تعویق بیندازد.
- اگر پاسخ سرور نشان دهنده تغییر مسیر باشد (کد وضعیت HTTP 3XX)، ربات باید تغییر مسیرها را تا زمانی که منبع پیدا شود دنبال کند.

robots.txt File Format

فایل robots.txt یک Syntax بسیار ساده و line-oriented دارد. سه نوع خط در فایل robots.txt وجود دارد: خطوط خالی، خطوط Comment و خطوط قانون. خطوط قوانین شبیه هدرهای HTTP (`<Field>:<Value>`) هستند و برای تطبیق الگو استفاده می‌شوند. به عنوان مثال:

```
# this robots.txt file allows Slurp & Webcrawler to crawl
# the public parts of our site, but no other robots...
```

```
User-Agent: slurp
User-Agent: webcrawler
Disallow: /private
```

```
User-Agent: *
Disallow:
```





خطوط در یک فایل robots.txt به طور منطقی به ایجاد "رکوردها" منجر می‌شوند. هر رکورد مجموعه‌ای از قوانین حذف را برای مجموعه خاصی از روبات‌ها توصیف می‌کند. به این ترتیب، قوانین حذف متفاوتی را می‌توان برای روبات‌های مختلف اعمال کرد.

هر رکورد شامل مجموعه‌ای از خطوط قانون است که با یک خط خالی یا کاراکتر انتهای فایل خاتمه می‌یابد. یک رکورد با یک یا چند خط User-Agent شروع می‌شود که مشخص می‌کند کدام ربات تحت تأثیر این رکورد قرار می‌گیرد و به دنبال آن خطوط Allow و Disallow می‌گویند که این روبات‌ها به چه URL هایی می‌توانند دسترسی داشته باشند.

مثال قبلی یک فایل robots.txt را نشان می‌دهد که به روبات‌های Webcrawler و Slurp اجازه می‌دهد به هر فایلی به جز آن فایل‌های موجود در زیر شاخه خصوصی دسترسی داشته باشند. همین فایل همچنین از دسترسی ربات‌های دیگر به هر چیزی در سایت جلوگیری می‌کند.

بیایید به خطوط User-Agent، Allow و Disallow نگاه کنیم.

The User-Agent line

هر رکورد ربات با یک یا چند خط User-Agent به شکل زیر شروع می‌شود:

User-Agent: <robot-name> or User-Agent: *

نام ربات (انتخاب شده توسط طراح ربات) در سربرگ HTTP GET درخواست User-Agent ربات ارسال می‌شود.

هنگامی که یک ربات یک فایل robots.txt را پردازش می‌کند، باید از رکوردهای زیر پیروی کند:

- اولین نام ربات که شامل رشته است (عدم حساسیت به حروف کوچک و بزرگ)
- اولین نام ربات که به صورت * است.

اگر ربات نتواند یک خط User-Agent مطابق با نام خود پیدا کند و نتواند یک خط "User-Agent: *" پیدا کند، هیچ رکوردهای مطابقت نداشته و دسترسی نامحدود است.

از آنجایی که نام ربات به حروف کوچک و بزرگ حساس نیست، مراقب تطابق‌های نادرست باشید. به عنوان مثال، "User-Agent: bot" با تمام ربات‌هایی به نام‌های Bot، Bottom-Feeder، Robot و Spambot مطابقت دارد. "User-Agent: Bother-Me" نامحدود است.





The Disallow and Allow lines

خطوط Disallow و Allow بلافاصله از خطوط User-Agent قرار می‌گیرند. آن‌ها توضیح می‌دهند که کدام مسیرهای URL به صراحت برای روبات‌های مشخص شده ممنوع یا به صراحت مجاز هستند.

ربات باید URL مورد نظر را به ترتیب با تمام قوانین Disallow و Allow مطابقت دهد. اولین مطابقت یافت شده استفاده می‌شود. اگر مطابقت پیدا نشد، URL مجاز است.

برای تطابق خط Allow/Disallow با URL، مسیر قانون باید پیشوند مسیر URL حساس به حروف بزرگ و کوچک باشد. به عنوان مثال، "Disallow: /tmp" با همه این URL‌ها مطابقت دارد:

http://www.joes-hardware.com/tmp

http://www.joes-hardware.com/tmp/

http://www.joes-hardware.com/tmp/pliers.html

http://www.joes-hardware.com/tmpspc/stuff.txt

Disallow/Allow prefix matching

در اینجا چند جزئیات بیشتر در مورد تطبیق پیشوند Allow/Disallow وجود دارد:

- قوانین Allow/Disallow به تطابق پیشوند حساس به حروف بزرگ و کوچک نیاز دارند. ستاره هیچ معنای خاصی ندارد (برخلاف خطوط User-Agent)، اما Universal Wildcarding Effect را می‌توان از رشته خالی به دست آورد.
- هر Escaped Charachter ای مانند %XX در مسیر قانون یا مسیر URL قبل از مقایسه به بایت‌ها بازگردانده می‌شود (Unescaped Back). به استثنای %2F که معادل فوردارد اسلش بوده و باید دقیقاً مطابقت داشته باشد.
- اگر مسیر قانون رشته خالی باشد، با همه چیز مطابقت دارد.

جدول زیر چندین نمونه از تطابق بین مسیرهای قانون و مسیرهای URL را فهرست می‌کند:





Rule path	URL path	Match?	Comments
/tmp	/tmp	✓	Rule path == URL path
/tmp	/tmpfile.html	✓	Rule path is a prefix of URL path
/tmp	/tmp/a.html	✓	Rule path is a prefix of URL path
/tmp/	/tmp	✗	/tmp/ is not a prefix of /tmp
	README.TXT	✓	Empty rule path matches everything
/~fred/hi.html	%7Efred/hi.html	✓	%7E is treated the same as ~
/%7Efred/hi.html	/~fred/hi.html	✓	%7E is treated the same as ~
/%7efred/hi.html	/%7Efred/hi.html	✓	Case isn't significant in escapes
/~fred/hi.html	~fred%2Fhi.html	✗	%2F is slash, but slash is a special case that must match exactly

تطبيق پیشوند معمولاً به خوبی کار می‌کند، اما چند جا وجود دارد که به اندازه کافی رسا نیست. اگر زیردایرکتوری‌های خاصی وجود دارد که می‌خواهید Crawling برای آن‌ها نیز ممنوع شود، صرف نظر از اینکه پیشوند مسیر چیست، هیچ وسیله‌ای برای این کار ارائه نمی‌دهد. برای مثال، ممکن است بخواهید از Crawling زیرشاخه‌های کنترل نسخه RCS خودداری کنید. نسخه ۱.۰ طرح robots.txt هیچ راهی برای پشتیبانی از این ارائه نمی‌دهد، به جز برشمردن جداگانه هر مسیر به هر زیرشاخه RCS.

Other robots.txt Wisdom

در اینجا قوانین دیگری در رابطه با تجزیه فایل robots.txt وجود دارد:

فایل robots.txt ممکن است حاوی فیلدهایی غیر از User-Agent، Disallow و Allow باشد، زیرا مشخصات پیشرفت می‌کند. یک ربات باید هر زمینه‌ای را که نمی‌فهمد نادیده بگیرد. برای سازگاری با قبل(Backward Compatibility)، شکستن خطوط مجاز نیست.

Comment در هر نقطه از فایل مجاز است. آن‌ها از فضای خالی اختیاری تشکیل شده‌اند و به دنبال آن یک کاراکتر (#) و سپس Comment مورد نظر تا به کاراکتر پایان خط برسیم.

نسخه 0.0 از Robots Exclusion Standard از خط Allow پشتیبانی نمی‌کند. برخی از ربات‌ها فقط مشخصات نسخه 0.0 را اجرا می‌کنند و خطوط مجاز را نادیده می‌گیرند. در این شرایط، یک ربات محافظه کارانه رفتار می‌کند و URL‌های مجاز را بازیابی نمی‌کند.





Caching and Expiration of robots.txt

اگر یک ربات مجبور شود قبل از هر دسترسی به فایل، یک فایل robots.txt را دوباره واکشی کند، بار روی سرورهای وب را دو برابر می‌کند و همچنین کارایی ربات را کاهش می‌دهد. در عوض، از روبات‌ها انتظار می‌رود که فایل robots.txt را به صورت دوره‌ای واکشی کرده و نتایج را در Cache نگه دارند. کپی شده robots.txt باشد توسط ربات استفاده شود تا زمانی که فایل robots.txt منقضی شود.

mekanisem‌های استاندارد HTTP cache-control هم توسط سرور مبدا و هم ربات‌ها برای کنترل ذخیره‌سازی robots.txt استفاده می‌شوند. روبات‌ها باید در پاسخ HTTP به هدرهای Cache-Control و Expires توجه داشته باشند.

امروزه بسیاری از Crawler های تولیدی با کلاینت HTTP/1.1 کار نمی‌کنند. مدیران وبسایت‌ها باید توجه داشته باشند که آن Crawler ها لزوماً دستورالعمل‌های Cache ارائه شده برای منبع robots.txt را درک نمی‌کنند.

اگر دستورالعمل‌های Cache-Control وجود نداشته باشد، مشخصات پیش‌نویس (Draft Specification) Caching به مدت هفت روز را می‌دهد. اما، در عمل، این اغلب خیلی طولانی است. اگر فایل robots.txt برای یک هفته در Cache ذخیره شود، فایل robots.txt جدید ایجاد شده هیچ تاثیری نخواهد داشت و مدیر سایت، مدیر ربات را به عدم رعایت Robots Exclusion Standard متهم می‌کند.

Robot Exclusion Perl Code

چند کتابخانه پرل در دسترس عموم برای تعامل با فایل‌های robots.txt وجود دارد. یک مثال مژول WWW::RobotRules است که برای آرشیو عمومی پرل CPAN موجود است.

فایل robots.txt تجزیه شده در شیء WWW::RobotRules نگهداری می‌شود، که روش‌هایی را برای بررسی آینکه آیا دسترسی به یک URL داده شده ممنوع است یا خیر، ارائه می‌کند. همان شیء WWW::RobotRules می‌تواند چندین فایل robots.txt را تجزیه کند.

در اینجا روش‌های اولیه در WWW::RobotRules API آمده است:

ایجاد یک شیء RobotRules

```
$rules = WWW::RobotRules->new($robot_name);
```

بارگزاری فایل robots.txt





```
$rules->parse($url, $content, $fresh_until);
```

بررسی اینکه آیا URL سایت قابل واکشی است یا خیر

```
$can_fetch = $rules->allowed($url);
```

در اینجا یک برنامه کوتاه Perl وجود دارد که استفاده از WWW::RobotRules را نشان می‌دهد:

```
require WWW::RobotRules;
```

```
# Create the RobotRules object, naming the robot "SuperRobot"
```

```
my $robotsrules = new WWW::RobotRules 'SuperRobot/1.0';
```

```
use LWP::Simple qw(get);
```

```
# Get and parse the robots.txt file for Joe's Hardware, accumulating the rules
```

```
$url = "http://www.joes-hardware.com/robots.txt";
```

```
my $robots_txt = get $url;
```

```
$robotsrules->parse($url, $robots_txt);
```

```
# Get and parse the robots.txt file for Mary's Antiques, accumulating the rules
```

```
$url = "http://www.marys-antiques.com/robots.txt";
```

```
my $robots_txt = get $url;
```

```
;$robotsrules->parse($url, $robots_txt)$
```

```
# Now RobotRules contains the set of robot exclusion rules for several
```

```
# different sites. It keeps them all separate. Now we can use RobotRules
```

```
# to test if a robot is allowed to access various URLs.
```

```
if ($robotsrules->allowed($some_target_url))
```

```
{
```

```
    $c = get $url;
```

```
    ...
```

```
}
```





فایل زیر یک فایل robots.txt فرضی برای www.marys-antiques.com است:

```
#####
# This is the robots.txt file for Mary's Antiques web site
#####
# Keep Suzy's robot out of all the dynamic URLs because it doesn't
# understand them, and out of all the private data, except for the
# small section Mary has reserved on the site for Suzy.
```

User-Agent: Suzy-Spider

Disallow: /dynamic

Allow: /private/suzy-stuff

Disallow: /private

```
# The Furniture-Finder robot was specially designed to understand
# Mary's antique store's furniture inventory program, so let it
# crawl that resource, but keep it out of all the other dynamic
# resources and out of all the private data.
```

User-Agent: Furniture-Finder

Allow: /dynamic/check-inventory

Disallow: /dynamic

Disallow: /private

Keep everyone else out of the dynamic gateways and private data.

User-Agent: *

Disallow: /dynamic

Disallow: /private





این فایل robots.txt حاوی یک رکورد برای ربات به نام **SuzySpider**، یک رکورد برای ربات به نام **FurnitureFinder** و یک رکورد پیش فرض برای همه ربات‌های دیگر است. هر رکورد مجموعه متفاوتی از سیاست‌های دسترسی را برای روبات‌های مختلف اعمال می‌کند:

- **Gateway Exclusion Record** برای Crawling، ربات را از **SuzySpider** در آدرس‌های اینترنتی موجودی فروشگاه که با **/dynamic** شروع می‌شوند و از داده‌های خصوصی کاربر خارج می‌شوند، حفظ می‌کند، به جز قسمتی که برای **Suzy** رزرو شده است.
- رکورد ربات **FurnitureFinder** به ربات اجازه می‌دهد تا URL دروازه موجودی مبلمان را **Crawl** کند.
- شاید این ربات فرمت و قوانین دروازه **Mary** را درک کند.
- همه ربات‌های دیگر از تمام صفحات وب پویا و خصوصی دور نگه داشته می‌شوند، اگرچه می‌توانند بقیه URL ها را **Crawl** کنند.

جدول زیر نمونه‌هایی را برای دسترسی ربات‌های مختلف به وب سایت **Mary's Antiques** فهرست می‌کند.

URL	SuzySpider	FurnitureFinder	NosyBot
http://www.marys-antiques.com/	✓	✓	✓
http://www.marys-antiques.com/index.html	✓	✓	✓
http://www.marys-antiques.com/private/payroll.xls	✗	✗	✗
http://www.marys-antiques.com/private/suzy-stuff/taxes.txt	✓	✗	✗
http://www.marys-antiques.com/dynamic/buy-stuff?id=3546	✗	✗	✗
http://www.marys-antiques.com/dynamic/check-inventory?kitchen	✗	✓	✗

HTML Robot-Control META Tags

فایل robots.txt به مدیر سایت اجازه می‌دهد تا ربات‌ها را از برخی یا همه یک وب سایت حذف کند. یکی از معایب فایل robots.txt این است که متعلق به مدیر وب سایت است نه نویسنده محتوا.

نویسنده‌گان صفحه HTML راه مستقیم‌تری برای محدود کردن روبات‌ها از صفحات جداگانه دارند. آن‌ها می‌توانند تگ‌های کنترل ربات را مستقیماً به اسناد HTML اضافه کنند. ربات‌هایی که به تگ‌های HTML کنترل ربات پاییند هستند همچنان می‌توانند اسناد را واکشی کنند، اما اگر تگ **Robot Exclusion** وجود داشته باشد، اسناد را نادیده می‌گیرند. به عنوان مثال، یک ربات موتور جستجوی اینترنتی سند را در فهرست جستجوی خود قرار نمی‌دهد. همانند استاندارد robots.txt، مشارکت تشویق می‌شود اما اجرا نمی‌شود.





تگهای Robot Exclusion با استفاده از تگهای HTML META، با استفاده از فرم پیاده سازی می‌شوند:

```
<META NAME="ROBOTS" CONTENT=directive-list>
```

Robot META directives

نوع مختلفی از دستورالعمل‌های META ربات وجود دارد و احتمالاً دستورالعمل‌های جدید به مرور زمان و با گسترش فعالیت‌ها و مجموعه ویژگی‌های موتورهای جستجو و روبات‌های آن‌ها اضافه خواهند شد. دو دستورالعمل META ربات که اغلب مورد استفاده قرار می‌گیرند عبارتند از:

NOINDEX

این تنظیم به یک ربات می‌گوید که محتوای صفحه را پردازش نکند و سند را نادیده بگیرد (یعنی محتوا را در هیچ فهرست یا پایگاه داده‌ای درج نکند).

```
<META NAME="ROBOTS" CONTENT="NOINDEX">
```

NOFOLLOW

این تنظیم به یک ربات می‌گوید که هیچ لینک خروجی را از صفحه Crawl نکند.

```
<META NAME="ROBOTS" CONTENT="NOFOLLOW">
```

علاوه بر NOINDEX و NOFOLLOW، دستورالعمل‌های مخالف FOLLOW و INDEX، دستورالعمل NOARCHIVE و دستورالعمل‌های ALL و NONE نیز وجود دارد. این دستورالعمل‌های META Tag ربات به شرح زیر خلاصه می‌شوند:

INDEX

به ربات می‌گوید که ممکن است محتویات صفحه را ایندکس کند.

FOLLOW

به یک ربات می‌گوید که ممکن است لینک‌های خروجی را در صفحه Crawl کند.

NOARCHIVE

به یک ربات می‌گوید که نباید یک Local Copy از صفحه را در Cache نگه دارد.

ALL

.FOLLOW، INDEX





NONE

معادل .NOFOLLOW, .NOINDEX

تگ‌های متا ربات، مانند تمام تگ‌های متا HTML، باید در بخش HEAD یک صفحه HTML ظاهر شوند:

```
<html>
<head>
    <meta name="robots" content="noindex,nofollow">
    <title>...</title>
</head>
<body>
    ...
</body>
</html>
```

توجه داشته باشید که نام "robots" تگ و محتوا به حروف بزرگ و کوچک حساس نیستند.

بدیهی است که نباید دستورالعمل‌های متناقض یا تکراری را مشخص کنید، مانند:

```
<meta name="robots" content="INDEX,NOINDEX,NOFOLLOW,FOLLOW,FOLLOW">
```

رفتاری که احتمالاً تعریف نشده است و مطمئناً از اجرای ربات به اجرای ربات متفاوت خواهد بود.

Search engine META tags

ما فقط تگ‌های متا ربات را مورد بحث قرار دادیم که برای کنترل فعالیت Indexing و Crawling ربات‌های وب استفاده می‌شود. تمام تگ‌های متا ربات‌ها حاوی ویژگی "name="robots"" هستند.

بسیاری از انواع دیگر تگ‌های متا در دسترس هستند، از جمله مواردی که در جدول زیر نشان داده شده است. تگ‌های متا KEYWORDS و DESCRIPTION برای ربات‌های موتور جستجو فهرست‌کننده محتوا مفید هستند.





name=	content=	Description
DESCRIPTION	<text>	Allows an author to define a short text summary of the web page. Many search engines look at META DESCRIPTION tags, allowing page authors to specify appropriate short abstracts to describe their web pages. <meta name="description" content="Welcome to Mary's Antiques web site">
KEYWORDS	<comma list>	Associates a comma-separated list of words that describe the web page, to assist in keyword searches. <meta name="keywords" content="antiques,mary,furniture,restoration">
REVISIT-AFTER ^a	<no. days>	Instructs the robot or search engine that the page should be revisited, presumably because it is subject to change, after the specified number of days. <meta name="revisit-after" content="10 days">

Robot Etiquette

در سال ۱۹۹۳، مارتین کوستر، پیشگام در جامعه ربات‌های وب، فهرستی از دستورالعمل‌ها را برای نویسندگان ربات‌های وب نوشت. در حالی که برخی از توصیه‌ها قدیمی هستند، بسیاری از آن‌ها هنوز کاملاً مفید هستند. رساله اصلی مارتین، "Guidelines for Robot Writers" را می‌توان در <http://www.robotstxt.org/wc/guidelines.html> یافت.

جدول زیر یک به روز رسانی مدرن برای طراحان و اپراتورهای ربات ارائه می‌دهد که عمدتاً بر اساس روح و محتوای فهرست اصلی است. بیشتر این دستورالعمل‌ها ربات‌های وب جهانی را هدف قرار داده‌اند. با این حال، آن‌ها برای Crawler‌های مقیاس کوچکتر نیز قابل استفاده هستند.

Guideline	Description
(1) Identification	
Identify Your Robot	Use the HTTP User-Agent field to tell web servers the name of your robot. This will help administrators understand what your robot is doing. Some robots also include a URL describing the purpose and policies of the robot in the User-Agent header.
Identify Your Machine	Make sure your robot runs from a machine with a DNS entry, so web sites can reverse-DNS the robot IP address into a hostname. This will help the administrator identify the organization responsible for the robot.
Identify a Contact	Use the HTTP From field to provide a contact email address.





Guideline	Description
(2) Operations	
Be Alert	Your robot will generate questions and complaints. Some of this is caused by robots that run astray. You must be cautious and watchful that your robot is behaving correctly. If your robot runs around the clock, you need to be extra careful. You may need to have operations people monitoring the robot 24 × 7 until your robot is well seasoned.
Be Prepared	When you begin a major robotic journey, be sure to notify people at your organization. Your organization will want to watch for network bandwidth consumption and be ready for any public inquiries.
Monitor and Log	Your robot should be richly equipped with diagnostics and logging, so you can track progress, identify any robot traps, and sanity check that everything is working right. We cannot stress enough the importance of monitoring and logging a robot's behavior. Problems and complaints will arise, and having detailed logs of a crawler's behavior can help a robot operator backtrack to what has happened. This is important not only for debugging your errant web crawler but also for defending its behavior against unjustified complaints.
Learn and Adapt	Each crawl, you will learn new things. Adapt your robot so it improves each time and avoids the common pitfalls.
(3) Limit Yourself	
Filter on URL	If a URL looks like it refers to data that you don't understand or are not interested in, you might want to skip it. For example, URLs ending in ".Z", ".gz", ".tar", or ".zip" are likely to be compressed files or archives. URLs ending in ".exe" are likely to be programs. URLs ending in ".gif", ".tif", ".jpg" are likely to be images. Make sure you get what you are after.
Filter Dynamic URLs	Usually, robots don't want to crawl content from dynamic gateways. The robot won't know how to properly format and post queries to gateways, and the results are likely to be erratic or transient. If a URL contains "cgi" or has a "?", the robot may want to avoid crawling the URL.
Filter with Accept Headers	Your robot should use HTTP Accept headers to tell servers what kind of content it understands.
Adhere to <i>robots.txt</i>	Your robot should adhere to the <i>robots.txt</i> controls on the site.
Throttle Yourself	Your robot should count the number of accesses to each site and when they occurred, and use this information to ensure that it doesn't visit any site too frequently. When a robot accesses a site more frequently than every few minutes, administrators get suspicious. When a robot accesses a site every few seconds, some administrators get angry. When a robot hammers a site as fast as it can, shutting out all other traffic, administrators will be furious. In general, you should limit your robot to a few requests per minute maximum, and ensure a few seconds between each request. You also should limit the total number of accesses to a site, to prevent loops.
(4) Tolerate Loops and Dups and Other Problems	
Handle All Return Codes	You must be prepared to handle all HTTP status codes, including redirects and errors. You should also log and monitor these codes. A large number of non-success results on a site should cause investigation. It may be that many URLs are stale, or the server refuses to serve documents to robots.
Canonicalize URLs	Try to remove common aliases by normalizing all URLs into a standard form.
Aggressively Avoid Cycles	Work very hard to detect and avoid cycles. Treat the process of operating a crawl as a feedback loop. The results of problems and their resolutions should be fed back into the next crawl, making your crawler better with each iteration.





Guideline	Description
Monitor for Traps	Some types of cycles are intentional and malicious. These may be intentionally hard to detect. Monitor for large numbers of accesses to a site with strange URLs. These may be traps.
Maintain a Blacklist	When you find traps, cycles, broken sites, and sites that want your robot to stay away, add them to a blacklist, and don't visit them again.
(5) Scalability	
Understand Space	Work out the math in advance for how large a problem you are solving. You may be surprised how much memory your application will require to complete a robotic task, because of the huge scale of the Web.
Understand Bandwidth	Understand how much network bandwidth you have available and how much you will need to complete your robotic task in the required time. Monitor the actual usage of network bandwidth. You probably will find that the outgoing bandwidth (requests) is much smaller than the incoming bandwidth (responses). By monitoring network usage, you also may find the potential to better optimize your robot, allowing it to take better advantage of the network bandwidth by better usage of its TCP connections. ^a
Understand Time	Understand how long it should take for your robot to complete its task, and sanity check that the progress matches your estimate. If your robot is way off your estimate, there probably is a problem worth investigating.
Divide and Conquer	For large-scale crawls, you will likely need to apply more hardware to get the job done, either using big multiprocessor servers with multiple network cards, or using multiple smaller computers working in unison.
(6) Reliability	
Test Thoroughly	Test your robot thoroughly internally before unleashing it on the world. When you are ready to test off-site, run a few, small, maiden voyages first. Collect lots of results and analyze your performance and memory use, estimating how they will scale up to the larger problem.
Checkpoint	Any serious robot will need to save a snapshot of its progress, from which it can restart on failure. There will be failures: you will find software bugs, and hardware will fail. Large-scale robots can't start from scratch each time this happens. Design in a checkpoint/restart feature from the beginning.
Fault Resiliency	Anticipate failures, and design your robot to be able to keep making progress when they occur.
(7) Public Relations	
Be Prepared	Your robot probably will upset a number of people. Be prepared to respond quickly to their enquiries. Make a web page policy statement describing your robot, and include detailed instructions on how to create a <i>robots.txt</i> file.
Be Understanding	Some of the people who contact you about your robot will be well informed and supportive; others will be naïve. A few will be unusually angry. Some may well seem insane. It's generally unproductive to argue the importance of your robotic endeavor. Explain the Robots Exclusion Standard, and if they are still unhappy, remove the complainant URLs immediately from your crawl and add them to the blacklist.
Be Responsive	Most unhappy webmasters are just unclear about robots. If you respond immediately and professionally, 90% of the complaints will disappear quickly. On the other hand, if you wait several days before responding, while your robot continues to visit a site, expect to find a very vocal, angry opponent.





Search Engines

گسترده‌ترین ربات‌های وب توسط موتورهای جستجوی اینترنتی استفاده می‌شود. موتورهای جستجوی اینترنتی به کاربران این امکان را می‌دهند که اسناد مربوط به هر موضوعی را در سراسر جهان پیدا کنند.

امروزه بسیاری از محبوب‌ترین سایتها در وب موتورهای جستجو هستند. آن‌ها به عنوان نقطه شروع برای بسیاری از کاربران وب عمل می‌کنند و خدمات ارزشمندی را ارائه می‌دهند که به کاربران کمک می‌کند اطلاعات مورد علاقه خود را پیدا کنند.

Crawler‌های وب موتورهای جستجوی اینترنتی را با بازیابی اسناد موجود در وب تغذیه می‌کنند و به موتورهای جستجو اجازه می‌دهند تا **Index**‌هایی از کلماتی که در چه اسنادی ظاهر می‌شوند را ایجاد کنند. موتورهای جستجو منبع اصلی ربات‌های وب هستند - بیایید نگاهی گذرا به نحوه کار آن‌ها بیندازیم.

Think Big

زمانی که وب در مراحل ابتدایی خود بود، موتورهای جستجو پایگاه داده‌های بودند که به کاربران کمک می‌کردند اسناد مورد نظر خود را در وب بیابند. امروزه با میلیاردها صفحه قابل دسترسی در وب، موتورهای جستجو برای کمک به کاربران اینترنت مجبور به یافتن اطلاعات ضروری شده و کاملاً پیچیده شده‌اند، چراکه باید برای مدیریت مقیاس وسیعی از وب تکامل یابند.

با میلیاردها صفحه وب و میلیون‌ها کاربر به دنبال اطلاعات، موتورهای جستجو باید Crawler‌های پیچیده‌ای را برای بازیابی این میلیاردها صفحه وب ایجاد نموده و همچنین می‌بایست برای مدیریت بار پرس و جو ای که میلیون‌ها کاربر ایجاد می‌کنند، ساختارهای پیچیده توازن بار را مستقر نمایند.

به وظیفه یک Crawler وب فکر کنید که باید میلیاردها پرس و جوی HTTP صادر نموده تا صفحات مورد نیاز Search Index را بازیابی کند. اگر تکمیل هر درخواست نیم ثانیه طول بکشد (که احتمالاً برای برخی از سرورها کند بوده و برای برخی دیگر سریع است)، هنوز هم (برای ۱ میلیارد سند) زمان زیادی طول خواهد کشید:

$$0.5 \text{ seconds} \times (1,000,000,000) / ((60 \text{ sec/day}) \times (60 \text{ min/hour}) \times (24 \text{ hour/day}))$$

واضح است که Crawler‌های مقیاس بزرگ باید با هوش‌تر باشند، درخواست‌ها را موازی‌سازی کنند و از بانک‌های ماشین‌ها برای تکمیل کار استفاده کنند. با این حال، به دلیل مقیاس آن، تلاش برای Crawling در کل وب هنوز یک چالش دلهزه آور است.



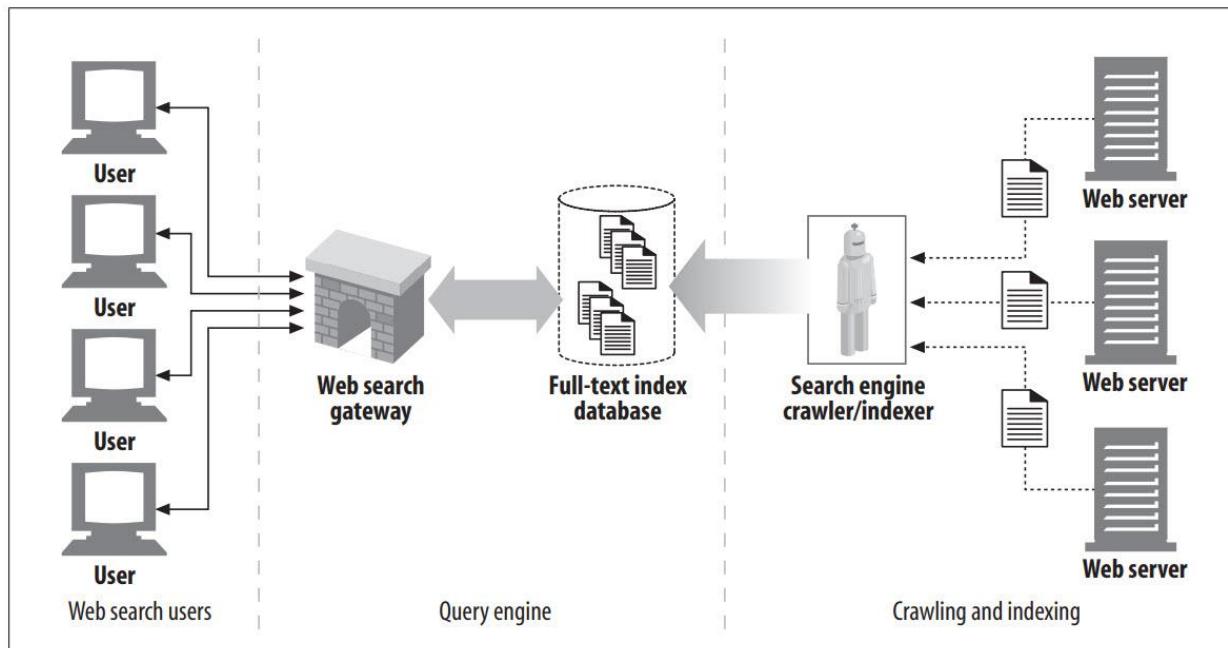


Modern Search Engine Architecture

موتورهای جستجوی امروزی پایگاه‌های اطلاعاتی محلی پیچیده‌ای به نام «full-text indexes» درباره صفحات وب در سراسر جهان و محتوای آن‌ها می‌سازند. این Index‌ها به عنوان نوعی کاتالوگ کارت برای تمام اسناد موجود در وب عمل می‌کنند.

full-text index‌های موتورهای جستجو، صفحات وب را جمع آوری کرده و به خانه می‌آورند و به HotBot اضافه می‌کنند. در همان زمان، کاربران موتورهای جستجو از طریق دروازه‌های جستجوی وب مانند (http://www.google.com) یا گوگل (http://www.hotbot.com) پرس و جوهایی را بر اساس full-text index صادر می‌کنند. از آنجایی که صفحات وب دائمًا در حال تغییر هستند و به دلیل زمان زیادی که ممکن است برای Crawling بخش بزرگی از وب طول بکشد، full-text index در بهترین حالت یک snapshot از وب است.

معماری سطح بالا یک موتور جستجوی مدرن در شکل زیر نشان داده شده است.



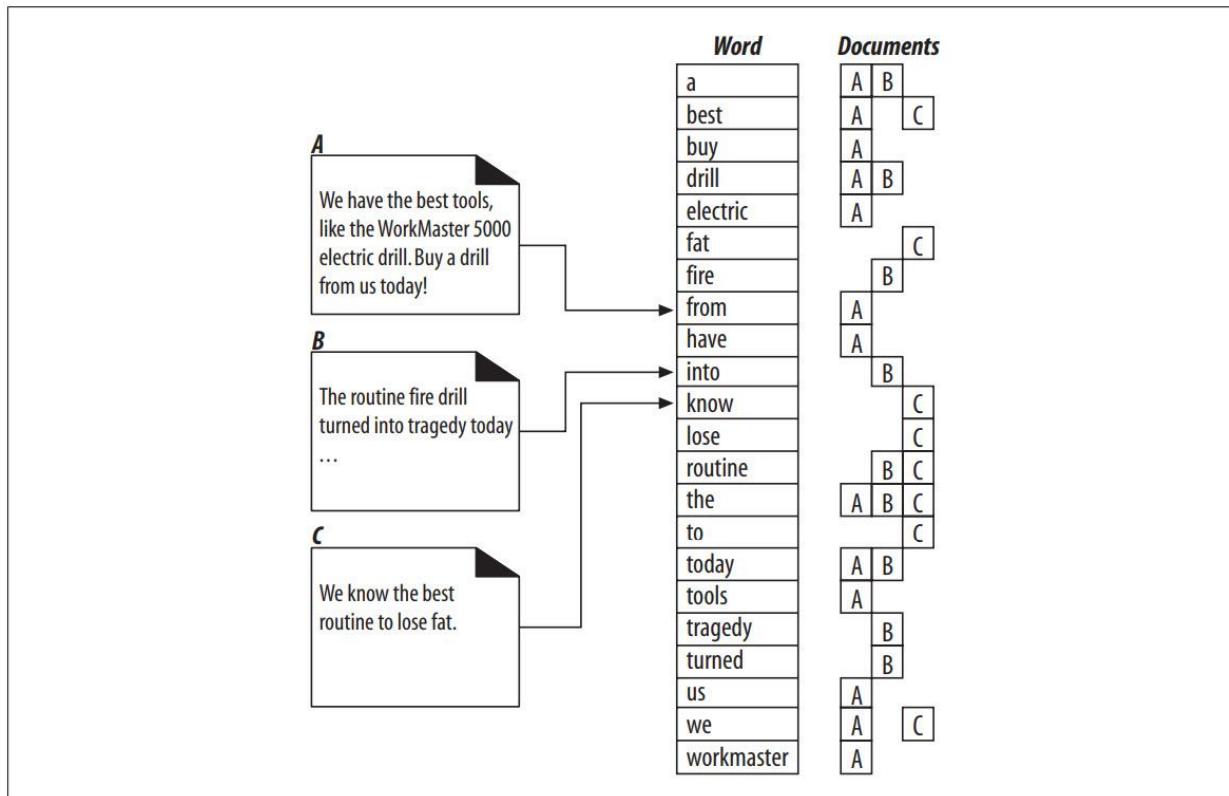
Full-Text Index

پایگاه داده‌ای است که یک کلمه را می‌گیرد و بلافاصله تمام اسناد حاوی آن کلمه را به شما می‌گوید. خود اسناد پس از ایجاد Index نیازی به اسکن ندارند.





شکل زیر سه سند و full-text index مربوطه را نشان می‌دهد. full-text index اسناد حاوی هر کلمه را فهرست می‌کند.



به عنوان مثال:

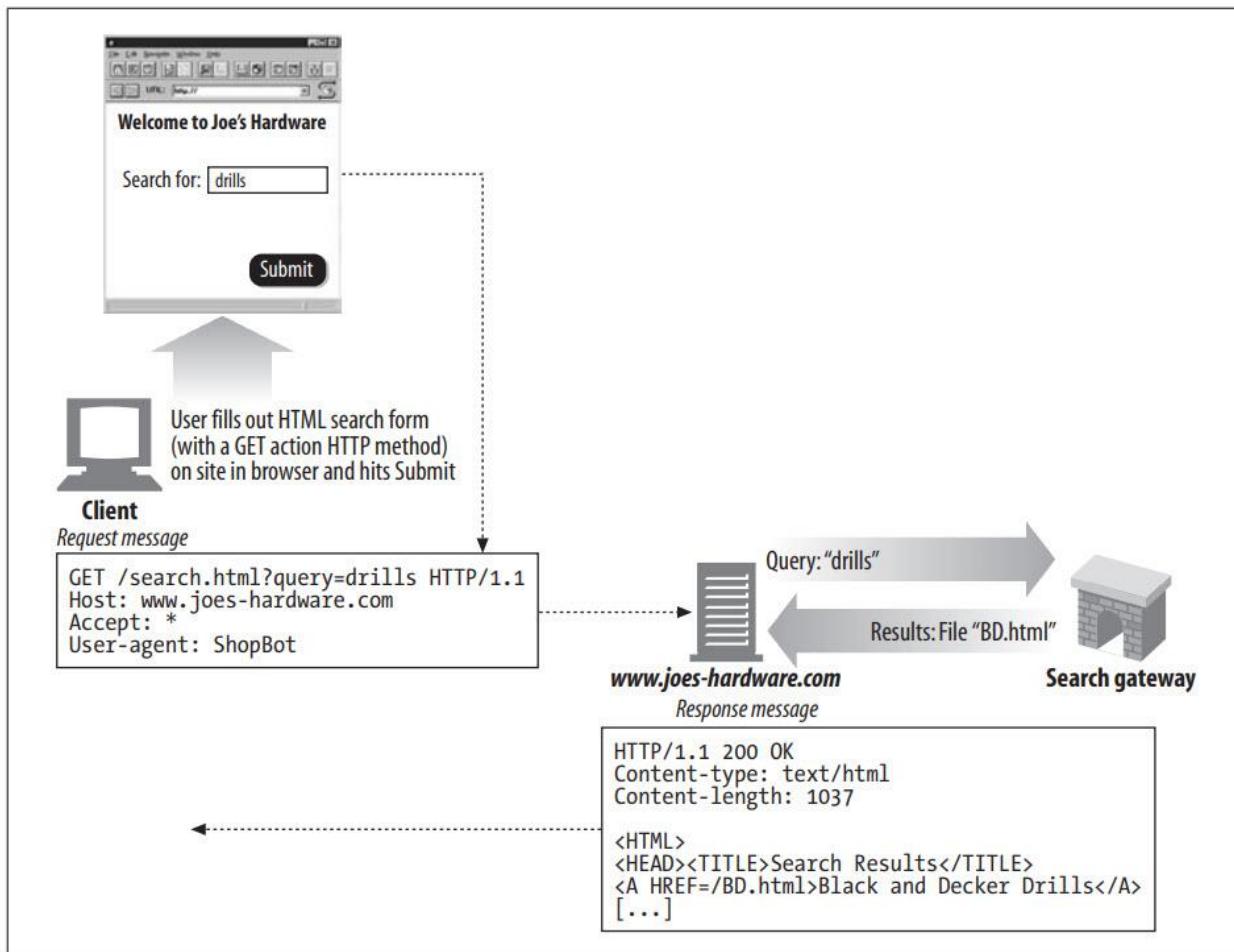
- کلمه a در اسناد A و B آمده است.
- کلمه best در اسناد A و C آمده است.
- کلمه drill در اسناد A و B آمده است.
- کلمه routine در اسناد B و C آمده است.
- کلمه the در تمامی اسناد A, B و C آمده است.

Posting the Query

هنگامی که کاربر یک پرس و جو را برای یک Gateway موتور جستجوی وب ارسال می‌کند، یک فرم HTML را پر می‌کند و مرورگر او فرم را با استفاده از یک درخواست HTTP GET یا POST به Gateway می‌فرماید. برنامه Gateway عبارت جستجو را استخراج می‌کند و پرس و جوی رابط کاربری وب را به عبارت مورد استفاده برای جستجوی full-text index تبدیل می‌کند.



شکل زیر یک پرس و جو ساده کاربر را در سایت www.joes-hardware.com نشان می‌دهد. کاربر عبارت "drills" را در فرم جعبه جستجو تایپ می‌کند و مرورگر آن را به یک درخواست GET با پارامتر query به عنوان "drills" بخشی از URL ترجمه می‌کند. وب سرور Joe's Hardware درخواست را دریافت می‌کند و آن را به برنامه Gateway جستجوی خود می‌دهد، که لیست حاصل از اسناد را به وب سرور برمی‌گرداند. در انتها نیز نتایج را به یک صفحه HTML برای کاربر قالب بندی می‌کند.



Sorting and Presenting the Results

هنگامی که یک موتور جستجو از فهرست خود برای تعیین نتایج یک پرس و جو استفاده می‌کند، برنامه Gateway نتایج را می‌گیرد و صفحه نتایج را برای کاربر نهایی می‌سازد.

از آنجایی که بسیاری از صفحات وب می‌توانند حاوی هر کلمه‌ای باشند، موتورهای جستجو از الگوریتم‌های هوشمندانه‌ای استفاده می‌کنند تا نتایج را رتبه بندی کنند. به عنوان مثال، در شکل پیشین کلمه "best" در چندین سند ظاهر می‌شود. موتورهای جستجو باید از ترتیبی که باید فهرست اسناد





نتیجه را ارائه کنند بدانند تا مرتبط‌ترین نتایج را به کاربران ارائه دهند. این موضوع Relevancy Ranking نامیده می‌شود که شامل فرآیند امتیازدهی و ترتیب فهرستی از نتایج جستجو است.

برای کمک بهتر به این فرآیند، بسیاری از موتورهای جستجوی بزرگتر در واقع از داده‌های سرشماری جمع آوری شده در طول Crawling در وب استفاده می‌کنند. برای مثال، شمارش تعداد لینک‌هایی که به یک صفحه معین اشاره می‌کنند می‌تواند به تعیین محبوبیت آن کمک کند و از این اطلاعات می‌توان برای وزن دادن به ترتیب ارائه نتایج استفاده کرد. الگوریتم‌ها، نکات مربوط به Crawling و سایر ترفندهای مورد استفاده موتورهای جستجو، برخی از محافظت‌شدۀ ترین اسرار آن‌هاست.

Spoofing

از آنجایی که کاربران اغلب وقتی آنچه را که به دنبال آن هستند در چند نتیجه اول جستجوی جستجو نمی‌بینند، نامید می‌شوند، ترتیب نتایج جستجو می‌تواند در یافتن یک سایت مهم باشد.

انگیزه زیادی برای وبمسترها وجود دارد که سعی کنند سایتها خود را در نزدیکی بالای بخش نتایج برای کلماتی که فکر می‌کنند بهترین توصیف سایتشان است، فهرست کنند، به خصوص اگر سایتها دارای خدمات تجاری بوده و به کاربران برای یافتن آن‌ها و استفاده از آن‌ها ممکن است الگوریتم‌های استفاده از آن‌ها ممکن است این هستند.

این تمایل برای فهرست بهتر منجر به بازی‌های زیادی در سیستم جستجو شده است و یک کشمکش دائمی بین اجراکنندگان موتورهای جستجو و کسانی که به دنبال فهرست بر جسته سایتها خود هستند ایجاد کرده است. بسیاری از وب مسترها هزاران کلمه کلیدی (بعضی نامربوط) را لیست می‌کنند و صفحات Fake یا جعلی را به کار می‌برند - حتی برنامه‌های دروازه‌ای که صفحات جعلی تولید می‌کنند که ممکن است الگوریتم‌های مرتبط موتورهای جستجو را برای کلمات خاص فریب دهند.

در نتیجه همه این‌ها، پیاده‌کنندگان موتورهای جستجو و ربات‌ها باید دائمًا الگوریتم‌های مربوط به خود را تغییر دهند تا بهتر این جعل‌ها را شناسایی نمایند.



For More Information

<http://www.robotstxt.org/wc/robots.html>

<http://www.searchengineworld.com>

<http://www.searchtools.com>

http://search.cpan.org/doc/ILYAZ/perl_st/WWW/RobotRules.pm

<http://www.conman.org/people/spc/robots2.html>

Managing Gigabytes: Compressing and Indexing Documents and Images





فصل دهم - HTTP-NG

با نزدیک شدن به اتمام این کتاب، HTTP دهمین تولد خود را جشن می‌گیرد و این یک دهه کامل برای این پروتکل اینترنتی بوده است. امروزه HTTP اکثریت مطلق ترافیک دیجیتال را در سراسر جهان جابجا می‌کند.

اما همانطور که HTTP در سال‌های نوجوانی خود رشد می‌کند، با چند چالش رو برو می‌شود. از برخی جهات، سرعت پذیرش HTTP از طراحی آن جلوتر رفته است. امروزه، مردم از HTTP به عنوان پایه‌ای برای بسیاری از برنامه‌های کاربردی مختلف، بر روی بسیاری از فناوری‌های مختلف شبکه استفاده می‌کنند.

این فصل به تشریح برخی از روندها و چالش‌های آینده HTTP و پیشنهادی برای معماری نسل بعدی به نام HTTP-NG می‌پردازد. در حالی که گروه کاری برای HTTP-NG منحل شده است و پذیرش سریع آن در حال حاضر بعيد به نظر می‌رسد، با این وجود برخی از جهت گیری‌های بالقوه آینده HTTP را مشخص می‌کند.

HTTP's Growing Pains

HTTP در ابتدا به عنوان یک تکنیک ساده برای دسترسی به محتوای چند رسانه‌ای مرتبط از سرورهای اطلاعات توزیع شده تصور شد. اما، در دهه گذشته، HTTP و مشتقات آن نقش بسیار گسترده‌تری را ایفا کردند.

HTTP/1.1 اکنون Fingerprinting و Tagging را برای ردیابی نسخه‌های سند، روش‌هایی برای پشتیبانی از بارگذاری اسناد و تعامل با Gateway های برنامه‌ای، پشتیبانی از محتوای چند زبانه، امنیت و احراز هویت، حافظه پنهان (Cache) برای کاهش ترافیک، Pipelining برای کاهش تأخیر، اتصالات مداوم (Persistent Connections) برای کاهش زمان راه اندازی و بهبود پهنای باند و دسترسی به محدوده برای پیاده سازی بروز رسانی‌های جزئی. برنامه‌های افزودنی و مشتقات HTTP حتی فراتر رفته‌اند و از انتشار اسناد، ارائه برنامه‌ها، پیام‌های دلخواه، پخش ویدئو و پایه‌های دسترسی چند رسانه‌ای بی‌سیم پشتیبانی می‌کنند. HTTP در حال تبدیل شدن به نوعی "سیستم عامل" برای برنامه‌های کاربردی رسانه‌های توزیع شده است.

طراحی HTTP/1.1، در حالی که به خوبی در نظر گرفته شده است، شروع به نشان دادن برخی فشارها کرده است زیرا HTTP بیشتر و بیشتر به عنوان یک بستر یکپارچه برای عملیات‌های پیچیده از راه دور استفاده می‌شود. حداقل چهار ناحیه وجود دارد که HTTP برخی از دردهای رشد را نشان می‌دهد:

Complexity

HTTP بسیار پیچیده است و ویژگی‌های آن به یکدیگر وابسته هستند. اجرای صحیح نرم‌افزار HTTP به دلیل نیازهای پیچیده و در هم تنیده و اختلاط مدیریت اتصال، مدیریت پیام و منطق عملکردی، قطعاً در دنیاک و مستعد خطا است.





Extensibility

گسترش تدریجی HTTP دشوار است. بسیاری از برنامه‌های HTTP قدیمی وجود دارند که ناسازگاری‌هایی را برای برنامه‌های افزودنی پروتکل ایجاد می‌کنند، زیرا فاقد فناوری برای برنامه‌های افزودنی عملکرد مستقل هستند.

Performance

HTTP ناکارآمدی عملکرد دارد. بسیاری از این ناکارآمدی‌ها با پذیرش گسترده فناوری‌های دسترسی بی‌سیم با تأخیر بالا و بازده کم جدی‌تر خواهند شد.

Transport dependence

HTTP حول یک پشتۀ شبکه TCP/IP طراحی شده است. در حالی که هیچ محدودیتی برای زیرپشتۀ‌های جایگزین وجود ندارد، کار کمی در این زمینه انجام شده است. HTTP برای اینکه به عنوان یک پلتفرم پیام رسانی گسترده‌تر در برنامه‌های کاربردی تعییه شده و بی‌سیم مفید باشد، نیاز به پشتیبانی بهتر از زیرپشتۀ‌های جایگزین دارد.

HTTP-NG Activity

در تابستان ۱۹۹۷، کنسرسیوم جهانی وب پروژه ویژه‌ای را برای بررسی و پیشنهاد نسخه جدید اصلی HTTP راه اندازی کرد که مشکلات مربوط به پیچیدگی، توسعه پذیری، عملکرد و وابستگی حمل و نقل را برطرف می‌کرد. این HTTP جدید (HTTP-NG: HTTP Next Generation) نام داشت.

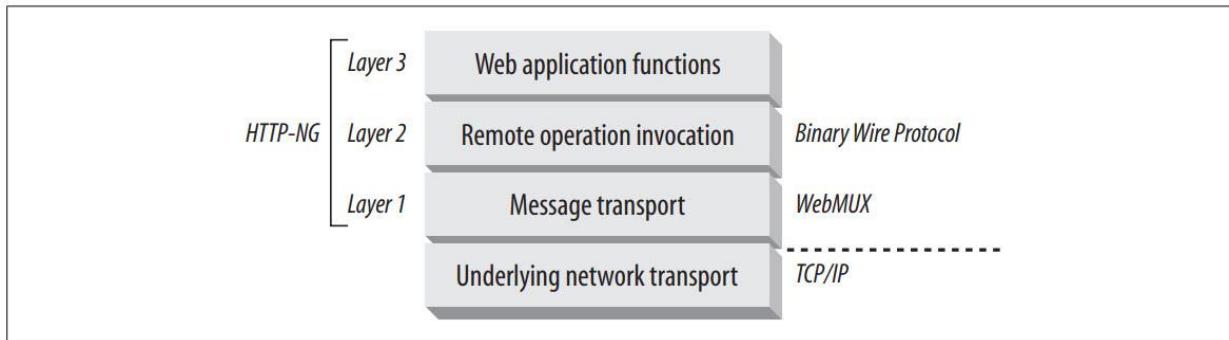
مجموعه‌ای از پیشنهادات HTTP-NG در جلسه IETF در دسامبر ۱۹۹۸ ارائه شد. این فناوری به طور گسترده پیاده سازی نشده است (و ممکن است هرگز اجرا نشود)، اما HTTP-NG نشان دهنده جدی‌ترین تلاش برای گسترش نسل HTTP است. بیایید HTTPNG را با جزئیات بیشتری بررسی کنیم.

Modularize and Enhance

موضوع HTTP-NG را می‌توان در این کلمات توصیف کرد: مدولار کردن (Modularize) و تقویت کردن (Enhance).

گروه کاری HTTP-NG به جای اینکه مدیریت اتصال، مدیریت پیام، منطق پردازش سرور و روش‌های پروتکل را در هم آمیخته باشد، مدولار کردن پروتکل را در سه لایه پیشنهاد کرد که در شکل زیر نشان داده شده است:





لایه ۱، لایه انتقال پیام، بر ارائه پیام‌های غیر شفاف بین نقاط پایانی، مستقل از عملکرد پیام‌ها تمرکز دارد. لایه انتقال پیام از زیرپشتنهای مختلف (مثلاً پشتنهای برای محیط‌های بی‌سیم) پشتیبانی می‌کند و بر مشکلات تحويل کارآمد پیام و مدیریت آن تمرکز می‌کند. تیم پروژه HTTP-NG پروتکلی به نام WebMUX برای این لایه پیشنهاد کرد.

لایه ۲، لایه فراخوانی راه دور، عملکرد Request/Response را تعریف می‌کند که در آن کلاینت‌ها می‌توانند عملیات منابع سرور را فراخوانی کنند. این لایه مستقل از انتقال پیام و معنای دقیق عملیات است. این فقط یک راه استاندارد برای فراخوانی هر عملیات سرور ارائه می‌دهد. این لایه تلاش می‌کند تا چارچوبی توسعه‌پذیر و شی‌گرا Binary Wire، HTTP-NG، DCOM، CORBA، Java RMI و RMI را بیشتر شبیه به پیشنهاد کرد.

لایه ۳، لایه برنامه وب، بیشتر منطق مدیریت محتوا را فراهم می‌کند. تمام متدهای HTTP/1.1 (GET، POST، PUT، وغیره) و همچنین پارامترهای هدر HTTP/1.1، در اینجا تعریف می‌شوند. این لایه همچنین از سایر خدمات ساخته شده بر روی فراخوانی راه دور مانند WebDAV پشتیبانی می‌کند.

هنگامی که اجزای HTTP مدولار می‌شوند، می‌توان آن‌ها را برای ارائه عملکرد بهتر و عملکرد غنی تر ارتقا داد.

Distributed Objects

بسیاری از اهداف فلسفه و عملکرد HTTP-NG به شدت از سیستم‌های ساختار یافته، شی‌گرا و اشیاء توزیع شده مانند DCOM و CORBA وام گرفته شده‌اند. سیستم‌های اشیاء توزیع شده می‌توانند به توسعه پذیری و عملکرد ویژگی‌ها کمک کنند.

جامعه‌ای از محققان از سال ۱۹۹۶ بر سر همگرایی بین HTTP و سیستم‌های پیچیده‌تر اشیاء توزیع شده بحث کرده‌اند. برای کسب اطلاعات بیشتر در مورد مزایای الگوی اشیاء توزیع شده برای وب، مقاله اولیه Xerox PARC با عنوان "مهاجرت وب به سوی اشیاء توزیع شده" را مطالعه نمایید.
(<ftp://ftp.parc.xerox.com/pub/ilu/misc/webilu.html>)





فلسفه جاه طلبانه یکسان سازی وب و اشیاء توزیع شده مقاومت در برابر پذیرش HTTP-NG در برخی جوامع ایجاد کرد. برخی از سیستم‌های اشیاء توزیع شده گذشته از اجرای سنگین وزن و پیچیدگی رسمی رنج می‌بردند. تیم پژوهش HTTP-NG تلاش کرد تا برخی از این نگرانی‌ها را در الزامات برطرف کند.

Layer 1: Messaging

بیایید نگاهی دقیق‌تر به سه لایه HTTP-NG بیندازیم که از پایین‌ترین لایه شروع می‌شود. لایه انتقال پیام به تحويل کارآمد پیام‌ها، مستقل از معنا و هدف پیام‌ها مربوط می‌شود. لایه انتقال پیام بدون توجه به پشتیبان شبکه واقعی، یک API برای پیام‌رسانی را فراهم می‌کند.

این لایه بر بهبود عملکرد پیام‌رسانی تمرکز دارد، از جمله:

- Pipelining و دسته بندی (Batching) پیام‌ها برای کاهش تأخیر رفت و برگشت
- استفاده مجدد از اتصالات برای کاهش تأخیر و بهبود پهنای باند تحويلی
- چندین جریان پیام به صورت موازی، روی یک اتصال، برای بهینه سازی اتصالات مشترک و در عین حال جلوگیری از Starvation جریان پیام
- تقسیم بندی پیام کارآمد برای آسان‌تر کردن تعیین مرزهای پیام

تیم HTTP-NG بیشتر انرژی خود را برای توسعه پروتکل WebMUX برای انتقال پیام لایه یک سرمایه گذاری کرد. WebMUX یک پروتکل پیام با کارایی بالا است که پیام‌ها را در یک اتصال TCP مالتی پلکس تقسیم می‌کند. ما WebMUX را در ادامه این فصل با کمی جزئیات بیشتر مورد بحث قرار خواهیم داد.

Layer 2: Remote Invocation

لایه میانی معماری HTTP-NG از فراخوانی روش راه دور پشتیبانی می‌کند. این لایه یک چارچوب Request/Response عمومی را فراهم می‌کند که در آن کلاینت‌ها عملیات منابع سرور را فراخوانی می‌کنند. این لایه به پیاده سازی و مفاهیم عملیات خاص (کش، امنیت، منطق روش و غیره) مربوط نمی‌شود. این فقط به اینترفیس مربوط می‌شود تا به کلاینت‌ها اجازه دهد تا از راه دور عملیات سرور را فراخوانی کنند.

بسیاری از استانداردهای فراخوانی روش از راه دور در حال حاضر در دسترس هستند (CORBA، DCOM، و جاوا RMI) و این لایه برای پشتیبانی از همه ویژگی‌های فوق العاده این سیستم‌ها در نظر گرفته نشده است. با این حال، یک هدف صریح برای گسترش غنای پشتیبانی HTTP RMI از آنچه توسط HTTP/1.1 ارائه شده وجود دارد. به طور خاص، هدفی برای ارائه پشتیبانی فراخوانی از راه دور عمومی‌تر، به شیوه‌ای قابل توسعه و شی گرا وجود دارد.





تیم HTTP-NG، Binary Wire Protocol را برای این لایه پیشنهاد کرد. این پروتکل از یک فناوری با کارایی بالا و توسعه‌پذیر برای فراخوانی عملیات توصیف شده روی سرور و انتقال نتایج پشتیبانی می‌کند. در ادامه این فصل، در مورد Binary Wire Protocol با جزئیات بیشتری صحبت می‌کنیم.

Layer 3: Web Application

لایه Web Application جایی است که معناشناسی و منطق برنامه کاربردی انجام می‌شود. گروه کاری-HTTP NG از وسوسه گسترش ویژگی‌های برنامه HTTP خودداری کرد و در عوض بر زیرساخت‌های رسمی تمرکز نمود.

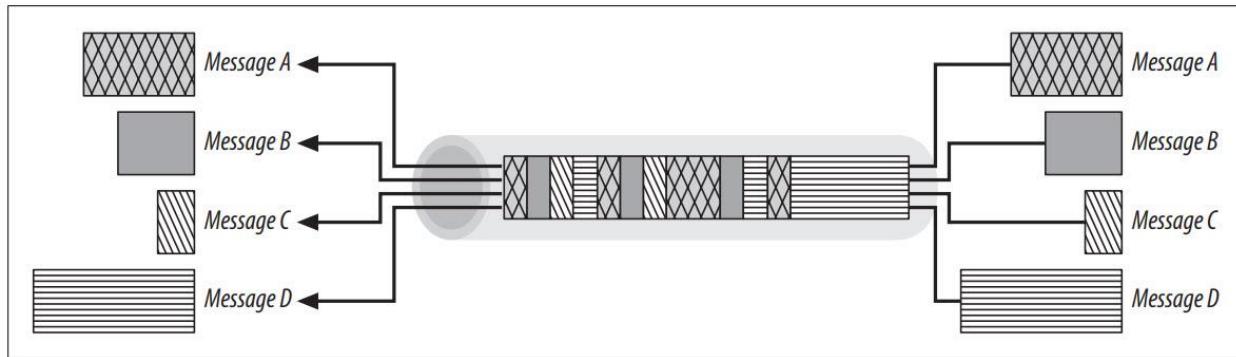
این لایه سیستمی را برای ارائه خدمات ویژه برنامه توصیف می‌کند. این خدمات یکپارچه نیستند. API های مختلف ممکن است برای برنامه‌های مختلف موجود باشد. به عنوان مثال، برنامه وب برای HTTP/1.1 یک برنامه کاربردی متفاوت از WebDAV را تشکیل می‌دهد، اگرچه ممکن است برخی از بخش‌های مشترک را به اشتراک بگذارند. معماری HTTP-NG به چندین برنامه اجازه می‌دهد تا در این سطح همزیستی داشته باشند، امکانات زیربنایی را به اشتراک بگذارند و مکانیزمی برای افزودن برنامه‌های کاربردی جدید فراهم می‌کند.

فلسفه لایه Web Application این است که عملکردی معادل برای HTTP/1.1 و رابطه‌ای افزونه (Extension Interfaces) ارائه دهد، در حالی که آن‌ها را در چارچوبی از اشیاء توزیع شده قابل توسعه باز می‌سازد. می‌توانید در <http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-larner-interfaces-00.txt> درباره رابطه‌ای لایه Web Application اطلاعات بیشتری کسب کنید.

WebMUX

گروه کاری HTTP-NG بیشتر انرژی خود را در توسعه استاندارد WebMUX برای انتقال پیام سرمایه گذاری کرده است. WebMUX یک سیستم پیام پیچیده و با کارایی بالا است که در آن پیام‌ها می‌توانند به صورت موازی از طریق یک اتصال TCP مالتی پلکس منتقل شوند. جریان‌های پیام فردی که با نرخ‌های مختلف تولید و مصرف می‌شوند، می‌توانند به طور مؤثری روی یک یا تعداد کمی از اتصالات TCP بسته‌بندی و چندگانه شوند.





در اینجا برخی از اهداف مهم پروتکل WebMUX آورده شده است:

- طراحی ساده
 - کارایی بالا
 - Multiplexing - چندین جریان داده (از پروتکل‌های سطح بالاتر دلخواه) می‌توانند به صورت پویا و کارآمد در یک اتصال واحد، بدون توقف داده‌ها در انتظار تولیدکنندگان کند قرار گیرند.
 - کنترل جریان مبتنی بر اعتبار - داده‌ها با نرخ‌های متفاوتی تولید و مصرف می‌شوند و فرستنده‌ها و گیرنده‌گان مقادیر متفاوتی از حافظه و منابع CPU در دسترس دارند. WebMUX از یک طرح کنترل جریان "مبتنی بر اعتبار" استفاده می‌کند، که در آن گیرنده‌گان علاقه خود را به دریافت داده‌ها برای جلوگیری از بن بست کمبود منابع اعلام می‌کنند.
 - Alignment Preserving - تراز داده‌ها در جریان چندگانه حفظ می‌شود تا داده‌های باینری را بتوان به طور موثر ارسال و پردازش کرد.
 - Rich Functionality - این رابط به اندازه کافی غنی است که از یک API سوکت پشتیبانی کند.
- همچنین شما می‌توانید اطلاعات بیشتری در مورد پروتکل WebMUX را در [مطالعه نمایید.](http://www.w3.org/Protocols/MUX/WD-mux-980722.html)

Binary Wire Protocol

تیم HTTP-NG پروتکل Binary Wire را برای بهبود نحوه پشتیبانی پروتکل HTTP نسل بعدی از عملیات راه دور پیشنهاد کرد.

Object type، HTTP-NG این特یپ، HTTP-NG یک مدل اجرایی توسعه پذیرتر و شی گرایی از مدل ارائه شده با HTTP/1.1 را پیشنهاد می‌کند، که در آن همه متدهای به صورت ایستا در سرورها تعریف شده بودند.





پروتکل Binary Wire درخواست‌های فرآخوانی عملیات را از کلاینت به سرور و پاسخ‌های نتیجه عملیات را از سرور به کلاینت از طریق یک اتصال stateful حمل می‌کند. اتصال stateful کارایی بیشتری را فراهم می‌کند.

پیام‌های درخواست شامل عملیات، شی مورد نظر و مقادیر داده اختیاری است. پیام‌های پاسخ وضعیت خاتمه عملیات، شماره سریال درخواست تطبیق (که به ترتیب دلخواه درخواست‌ها و پاسخ‌های موازی اجازه می‌دهد) و مقادیر بازگشتی اختیاری را بازمی‌گردانند. علاوه بر پیام‌های درخواست و پاسخ، این پروتکل چندین پیام کنترل داخلی را تعریف می‌کند که برای بهبود کارایی و استحکام اتصال استفاده می‌شود.

همچنین شما می‌توانید در <http://www.w3.org/Protocols/ HTTP-NG/1998/08/draft- janssen-httpng-wire-00.txt> درباره پروتکل Binary Wire اطلاعات بیشتری کسب نمایید.

Current Status

در پایان سال ۱۹۹۸، تیم HTTP-NG به این نتیجه رسید که برای ارائه پیشنهادات HTTP-NG به IETF برای استانداردسازی خیلی زود است. این نگرانی وجود داشت که صنعت و جامعه هنوز به طور کامل با HTTP/1.1 سازگار نشده‌اند و اینکه معماری مجدد HTTP-NG قابل توجه در الگوی اشیاء توزیع شده بدون یک برنامه انتقال روشی بسیار مخرب بود.

در همین راستا دو پیشنهاد ارائه شد:

به جای تلاش برای ارتقای کل معماری HTTP-NG در یک مرحله، تمرکز بر فناوری حمل و نقل WebMUX پیشنهاد شد. اما در زمان نگارش این مقاله، علاقه کافی برای ایجاد یک گروه کاری WebMUX وجود نداشته است.

تلاشی برای بررسی اینکه آیا انواع پروتکل‌های رسمی را می‌توان به اندازه کافی برای استفاده در وب، شاید با استفاده از XML، انعطاف پذیر ساخت یا خیر، آغاز شد. این امر به ویژه برای یک سیستم اشیاء توزیع شده که قابل گسترش است، مهم است. این کار هنوز در حال انجام است.

در زمان نگارش این مقاله، هیچ تلاش مهمی برای هدایت HTTP-NG در حال انجام نیست. اما، با استفاده از HTTP، استفاده رو به رشد آن به عنوان پلتفرمی برای کاربردهای متنوع، و پذیرش روزافزون فناوری اینترنت بی‌سیم و مصرف‌کننده، برخی از تکنیک‌های پیشنهاد شده در HTTP-NG ممکن است در سال‌های نوجوانی HTTP قابل توجه باشد.





For More Information

<http://www.w3.org/Protocols/HTTP-NG>

<http://www.w3.org/Protocols/MUX/WD-mux-980722.html>

<http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-janssen-httpng-wire-00.txt>

<http://www.w3.org/Protocols/HTTP-NG/1998/08/draft-larner-nginterfaces-00.txt>

<ftp://ftp.parc.xerox.com/pub/ilu/misc/webilu.html>





فصل یازدهم – Client Identification and Cookies

سرورهای وب ممکن است با هزاران کلاینت مختلف به طور همزمان صحبت کنند. این سرورها اغلب به جای تلقی کردن همه درخواست‌ها به عنوان کلاینت‌های ناشناس نیاز دارند تا متوجه شوند که با چه کسی صحبت می‌کنند. این فصل به برخی از فناوری‌هایی می‌پردازد که سرورها می‌توانند برای شناسایی افرادی که با آن‌ها صحبت می‌کنند استفاده کنند.

The Personal Touch

HTTP زندگی خود را به عنوان یک پروتکل Request/Response ناشناس، بدون وضعیت (Stateless)، آغاز کرد. درخواستی از یک کلاینت می‌آید، توسط سرور پرداش می‌شود و یک پاسخ به کلاینت ارسال می‌شود. اطلاعات کمی برای تعیین اینکه چه کاربری درخواست را ارسال کرده یا برای پیگیری دنباله ای از درخواست‌های کاربر بازدید کننده در دسترس وب سرور بود.

وب سایت‌های مدرن می‌خواهند که یک تماس شخصی (Personal Touch) ارائه دهند. آن‌ها می‌خواهند درباره کاربرانی که در انتهای دیگر اتصالات قرار دارند بیشتر بدانند و بتوانند آن کاربران را هنگام Browse ردیابی کنند. سایت‌های خرید آنلاین محبوب مانند Amazon.com سایت‌های خود را به چند روش برای شما شخصی سازی می‌کنند:

Personal greetings

پیام‌های خوش آمدگویی و محتویات صفحه به طور ویژه برای کاربر ایجاد می‌شوند تا تجربه خرید را شخصی‌تر کنند.

Targeted recommendations

فروشگاه‌ها با آگاهی از علایق مشتری می‌توانند محصولاتی را پیشنهاد دهند که مشتری از آن‌ها استقبال خواهد کرد. فروشگاه‌ها همچنین می‌توانند تخفیف‌های ویژه تولد را در نزدیکی تولد مشتریان و سایر روزهای مهم اجرا کنند.

Administrative information on file

خریداران آنلاین از پر کردن فرم‌های دست و پا گیر آدرس و کارت اعتباری بارها و بارها متنفرند. برخی از سایت‌ها این جزئیات را در یک پایگاه داده ذخیره می‌کنند. هنگامی که آن‌ها شما را شناسایی کردن، می‌توانند از اطلاعات موجود استفاده نموده و تجربه خرید را بسیار راحت‌تر کنند.

Session tracking



تراکنش‌های HTTP بدون وضعیت یا Request/Response Stateless هستند. هر تراکنش‌های HTTP به صورت مجزا انجام می‌شود. بسیاری از وبسایت‌ها می‌خواهند هنگام تعامل با سایت، حالت افزایشی ایجاد کنند (به عنوان مثال، پر کردن یک سبد خرید آنلاین). برای انجام این کار، وبسایت‌ها به راهی برای تشخیص تراکنش‌های HTTP از کاربران مختلف نیاز دارند.

این فصل تعدادی از تکنیک‌های مورد استفاده برای شناسایی کاربران در HTTP را خلاصه می‌کند. خود HTTP با مجموعه‌ای غنی از ویژگی‌های شناسایی متولد نشده است. طراحان اولیه وب سایت، فناوری‌های خود را برای شناسایی کاربران ساختند. هر تکنیکی نقاط قوت و ضعف خود را دارد. در این فصل، مکانیسم‌های زیر را برای شناسایی کاربران مورد بحث قرار خواهیم داد:

هدرهای HTTP که حاوی اطلاعاتی در مورد هویت کاربر هستند.

- ردیابی آدرس IP کلاینت، برای شناسایی کاربران با آدرس IP آن‌ها
- ورود کاربر، با استفاده از احراز هویت برای شناسایی کاربران
- Fat URL ها، تکنیکی برای جاسازی هویت در URL ها
- کوکی‌ها، یک تکنیک قدرتمند اما کارآمد برای حفظ هویت پایدار

HTTP Headers

جدول زیر هفت هدر درخواست HTTP را نشان می‌دهد که معمولاً اطلاعات مربوط به کاربر را حمل می‌کنند. اکنون سه مورد اول را مورد بحث قرار خواهیم داد. چهار هدر آخر برای تکنیک‌های شناسایی پیشرفته‌تر استفاده می‌شوند که بعداً در مورد آن‌ها صحبت خواهیم کرد.

Header name	Header type	Description
From	Request	User's email address
User-Agent	Request	User's browser software
Referer	Request	Page user came from by following link
Authorization	Request	Username and password (discussed later)
Client-ip	Extension (Request)	Client's IP address (discussed later)
X-Forwarded-For	Extension (Request)	Client's IP address (discussed later)
Cookie	Extension (Request)	Server-generated ID label (discussed later)

هدر From حاوی آدرس ایمیل کاربر است. در حالت ایده‌آل، این یک منبع قابل دوام برای شناسایی کاربر خواهد بود، زیرا هر کاربر یک آدرس ایمیل متفاوت خواهد داشت. با این حال، تعداد کمی از مرورگرها به دلیل نگرانی در مورد سرورهای مخرب که آدرس‌های ایمیل را جمع‌آوری می‌کنند و از آن‌ها





برای توزیع نامه‌های ناخواسته استفاده می‌کنند، هدرهای From ارسال می‌کنند. در عمل، هدرهای Rbots های خودکار یا Spider ها ارسال می‌شوند، به طوری که اگر چیزی اشتباه شود، یک وب‌مستر جایی برای ارسال شکایات ایمیلی را خواهد داشت.

هدر User-Agent اطلاعات مربوط به مرورگری که کاربر از آن استفاده می‌کند، شامل نام و نسخه برنامه و اغلب اطلاعات مربوط به سیستم عامل را به سرور ارسال می‌کند. این موضوع گاهی اوقات برای سفارشی کردن محتوا برای تعامل خوب با مرورگرهای خاص و ویژگی‌های آن‌ها مفید است، اما این کار کمک زیادی به شناسایی کاربر خاص به هیچ وجه معنی دار نمی‌کند. در اینجا دو عنوان User-Agent وجود دارد که یکی توسط Netscape و دیگری توسط Microsoft Internet Explorer ارسال شده است:

Navigator 6.2

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:0.9.4) Gecko/20011128
Netscape6/6.2.1

Internet Explorer 6.01

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

هدر Referer URL صفحه‌ای را که کاربر از آن آمده است را ارائه می‌دهد. هدر Referer به تنها یی و مستقیماً کاربر را شناسایی نمی‌کند، اما نشان می‌دهد که کاربر قبل از چه صفحه‌ای بازدید کرده است. می‌توانید از این برای درک بهتر رفتار مرورگر کاربر و علاقیکاربر استفاده کنید. برای مثال، اگر به وب سروری رسیدید که از یک سایت بیسیمال می‌آید، سرور ممکن است استنباط کند که شما طرفدار بیسیمال هستید.

هدرهای Referer و User-Agent، From برای اهداف شناسایی قابل اعتماد، کافی نیستند. بخش‌های باقی‌مانده طرح‌های دقیق‌تری را برای شناسایی کاربران خاص مورد بحث قرار می‌دهند.

Client IP Address

پیشگامان اولیه وب سعی کردند از آدرس IP کلاینت به عنوان نوعی شناسایی استفاده کنند. این طرح در صورتی کار می‌کند که هر کاربر یک آدرس IP مجزا داشته باشد، اگر آدرس IP به ندرت (اگر همیشه) تغییر کند و اگر وب سرور بتواند آدرس IP کلاینت را برای هر درخواست تعیین کند. در حالی که آدرس IP کلاینت معمولاً در هدرهای HTTP وجود ندارد، وب سرورها می‌توانند آدرس IP طرف دیگر اتصال TCP را که درخواست HTTP را حمل می‌کند، پیدا کنند. به عنوان مثال، در سیستم‌های یونیکس، فراخوانی تابع getpeername آدرس IP کلاینت دستگاه فرستنده را برمی‌گرداند:

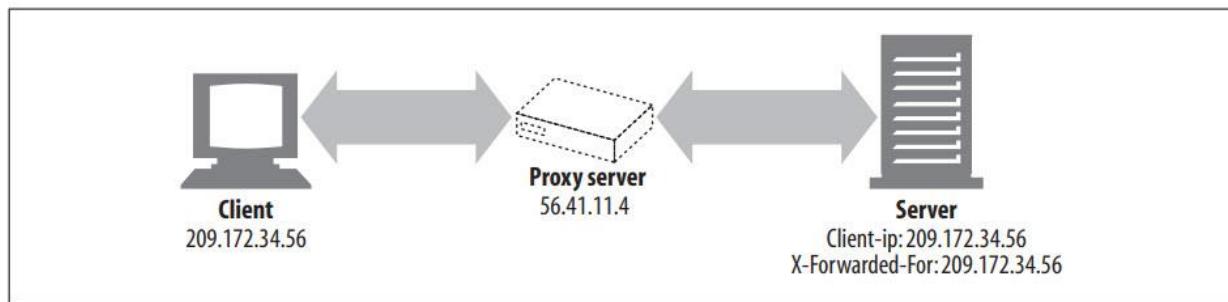
```
status = getpeername(tcp_connection_socket,...);
```





متأسفانه، استفاده از آدرس IP کلاینت برای شناسایی کاربر دارای نقاط ضعف متعددی است که اثربخشی آن را به عنوان یک فناوری شناسایی کاربر محدود می‌کند:

- آدرس‌های IP کلاینت فقط کامپیوتر مورد استفاده را توصیف می‌کند نه کاربر را. در این صورت اگر چندین کاربر از یک کامپیوتر مشترک استفاده کنند، قابل تشخیص نیستند.
- بسیاری از ارائه دهنگان خدمات اینترنتی به صورت پویا آدرس‌های IP را به کاربران هنگام ورود به سیستم اختصاص می‌دهند. هر بار که وارد می‌شوند، آدرس متفاوتی دریافت می‌کنند، بنابراین سرورهای وب نمی‌توانند فرض کنند که آدرس‌های IP، کاربر را در جلسات ورود شناسایی می‌کند.
- برای افزایش امنیت و مدیریت آدرس‌های کمیاب، بسیاری از کاربران، به اینترنت از طریق فایروال‌های ترجمه آدرس شبکه (NAT) دسترسی پیدا می‌کنند. این دستگاه‌های NAT آدرس‌های IP کلاینت‌های واقعی پشت فایروال را پنهان می‌کنند و آدرس IP کلاینت واقعی را به یک آدرس IP مشترک فایروال (و شماره پورت‌های مختلف) تبدیل می‌کنند.
- پروکسی‌ها و Gatway HTTP های معمولاً اتصالات TCP جدید را به سرور مبدا باز می‌کنند. وب سرور به جای آدرس سرویس گیرنده، آدرس IP سرور پروکسی را می‌بیند. برخی از پراکسی‌ها با افزودن هدرهای X-Forwarded-For HTTP یا Client-ip (شکل زیر). اما همه پروکسی‌ها از این رفتار پشتیبانی نمی‌کنند.



برخی از وب سایتها هنوز از آدرس‌های IP کلاینت برای پیگیری کاربران در بین جلسات استفاده می‌کنند. مکان‌های زیادی وجود دارد که هدف گذاری آدرس IP به خوبی کار نمی‌کند.

برخی از سایتها حتی از آدرس‌های IP کلاینت به عنوان یک ویژگی امنیتی استفاده می‌کنند و اسناد را فقط به کاربران از یک آدرس IP خاص ارائه می‌کنند. در حالی که این ممکن است در محدوده یک اینترنت کافی باشد، اما در اینترنت خراب می‌شود (به دلیل سهولت جعل آدرس‌های IP). وجود پروکسی‌های رهگیری در مسیر نیز این طرح را شکسته است. فصل ۱۴ طرح‌های بسیار قوی‌تری را برای کنترل دسترسی به اسناد ممتاز مورد بحث قرار می‌دهد.





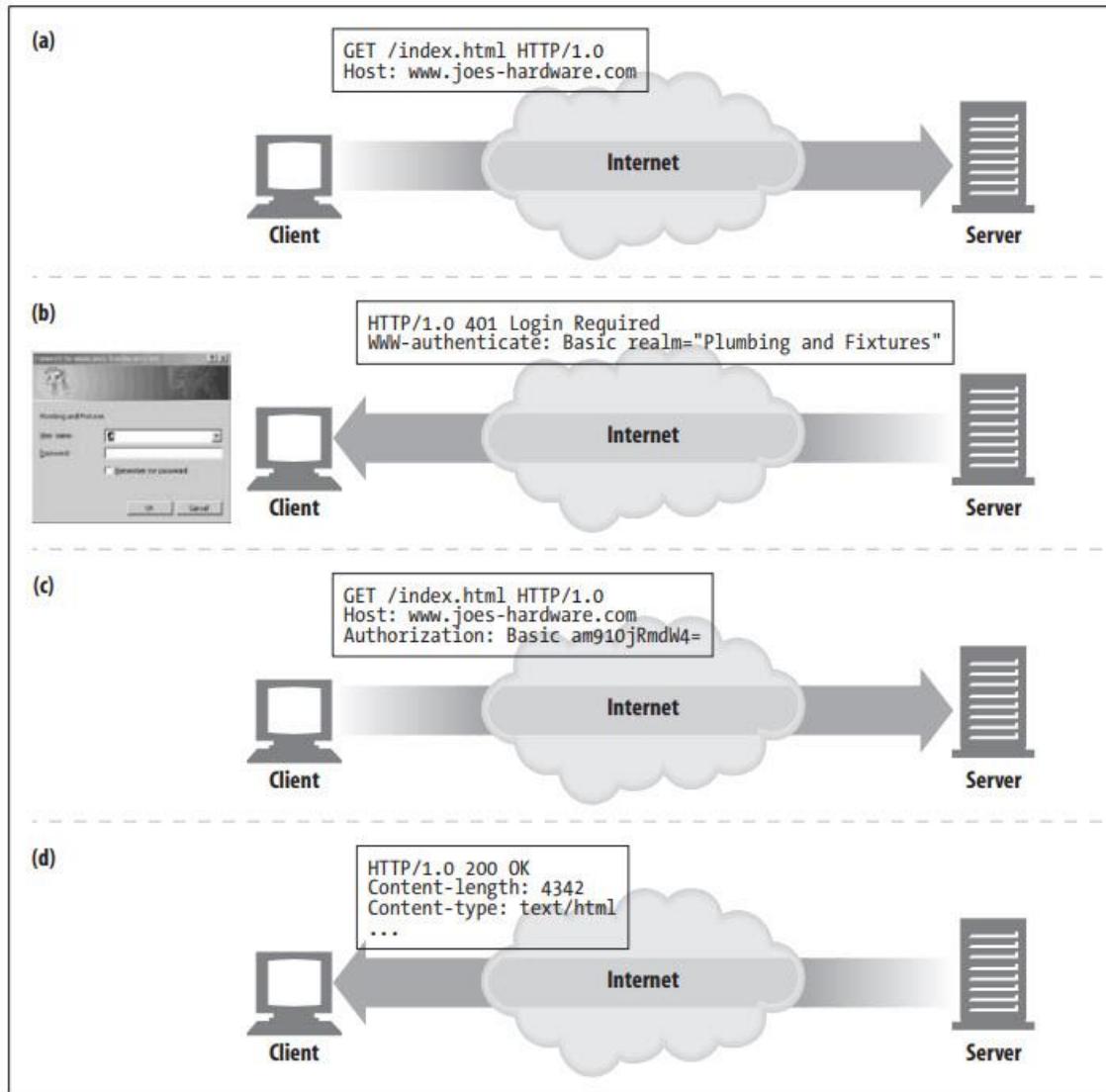
User Login

به جای تلاش منفعلانه برای حدس زدن هویت یک کاربر از آدرس IP او، یک وب سرور می‌تواند به طور صریح از کاربر بپرسد که او کیست و از او بخواهد با نام کاربری و رمز عبور، احراز هویت (ورود به سیستم) شود.

برای کمک به آسان‌تر کردن ورود به وبسایت، HTTP یک مکانیسم داخلی برای ارسال اطلاعات نام کاربری به وبسایتها، با استفاده از هدرهای Authorization و WWW-Authenticate دارد. پس از ورود، مرورگرها به طور مداوم این اطلاعات ورود را با هر درخواست به سایت ارسال می‌کنند، بنابراین اطلاعات همیشه در دسترس هستند. ما این احراز هویت HTTP را با جزئیات بیشتری در فصل ۱۲ مورد بحث قرار خواهیم داد، اما اجازه دهید اکنون نگاهی گذرا به آن بیندازیم.

اگر سروری بخواهد کاربر قبل از دسترسی به سایت ثبت نام کند، می‌تواند کد پاسخ مورد نیاز ورود به سیستم HTTP 401 را به مرورگر ارسال کند. سپس مرورگر یک کادر محاوره‌ای ورود به سیستم را نمایش می‌دهد و اطلاعات را در درخواست بعدی با استفاده از هدر Authorization به مرورگر ارائه می‌کند. این موضوع در شکل زیر نشان داده شده است.





فرآیندی که در این شکل اتفاق می‌افتد عبارتست از:

- در بخش a شکل بالا، یک مرورگر از سایت www.joes-hardware.com درخواست می‌کند.
- سایت هویت کاربر را نمی‌داند، بنابراین در بخش b شکل بالا، سرور با بازگرداندن کد پاسخ 401 Login Required HTTP درخواست ورود می‌کند و هدر WWW-Authenticate را اضافه می‌کند. این باعث می‌شود که در مرورگر یک کادر محاوره‌ای برای ورود به سیستم ظاهر شود.
- هنگامی که کاربر یک نام کاربری و یک رمز عبور را وارد می‌کند (برای بررسی هویت خود)، مرورگر درخواست اصلی را تکرار می‌کند. این بار یک هدر Authorization اضافه می‌کند که نام کاربری و رمز عبور را مشخص می‌کند. نام کاربری و رمز عبور درهم می‌شوند تا از دید ناظران تصادفی یا تصادفی شبکه پنهان شوند.





- اکنون سرور از هویت کاربر آگاه است.
- برای درخواست‌های بعدی، مرورگر به طور خودکار نام کاربری و رمز عبور ذخیره شده را در صورت درخواست صادر می‌کند و اغلب حتی در صورت عدم درخواست، آن را به سایت ارسال می‌کند. این موضوع سبب می‌شود که با ارسال هدر مجوز (به عنوان نشانه هویت شما) در هر درخواست به سرور، یک بار به یک سایت وارد شده و هویت شما در طول جلسه حفظ شود.

با این حال، ورود به وب سایتها خسته کننده است. همانطور که فرد از سایتی به سایت دیگر مرور می‌کند، باید برای هر سایتی اقدام به لگین نماید. بدتر از همه، این احتمال وجود دارد که فرد باید نام‌های کاربری و رمزهای عبور مختلف را برای سایتها مختلف به خاطر بسپارد. نام کاربری مورد علاقه او، "fred" از قبل توسط شخص دیگری در زمان بازدید از بسیاری از سایتها انتخاب شده است و برخی از سایتها قوانین متفاوتی در مورد طول و ترکیب نام‌های کاربری و رمز عبور خواهند داشت. خیلی زود، fred اینترنت را رها می‌کند و به تماسای اپرا باز می‌گردد. بخش بعدی راه حل این مشکل را مورد بحث قرار می‌دهد.

Fat URLs

برخی از وب‌سایتها با ایجاد نسخه‌های ویژه از هر URL برای هر کاربر، هویت کاربر را ردیابی می‌کنند. به طور معمول، یک URL واقعی با افروzen برخی از اطلاعات وضعیت به ابتدا یا انتهای مسیر URL گسترش می‌یابد. همانطور که کاربر سایت را مرور می‌کند، وب سرور به صورت پویا لینک‌هایی ایجاد می‌کند که همچنان اطلاعات وضعیت موجود در URL‌ها را حفظ می‌کند.

نشانی‌های اینترنتی که برای گنجاندن اطلاعات وضعیت کاربر اصلاح می‌شوند، Fat URLs نامیده می‌شوند. در زیر چند نمونه از این نوع URL‌های استفاده شده در وب سایت تجارت الکترونیک Amazon.com آورده شده است. هر URL با یک شماره شناسایی منحصر به فرد کاربر (002-1145265-8016838، در این مورد) مشخص می‌شود که به ردیابی کاربر در هنگام مرور فروشگاه کمک می‌کند.





```
...
<a href="/exec/obidos/tg/browse/-/229220/ref=gr_gifts/002-1145265-8016838">All  
Gifts</a><br>
<a href="/exec/obidos/wishlist/ref=gr_pl1_/002-1145265-8016838">Wish List</a><br>
...
<a href="http://s1.amazon.com/exec/varzea/tg/armed-forces/-//ref=gr_af_/002-1145265-  
8016838">Salute Our Troops</a><br>
<a href="/exec/obidos/tg/browse/-/749188/ref=gr_p4_/002-1145265-8016838">Free  
Shipping</a><br>

<a href="/exec/obidos/tg/browse/-/468532/ref=gr_returns/002-1145265-8016838">Easy  
Returns</a>
...

```

شما می‌توانید از Fat URL ها استفاده کنید تا تراکنش‌های HTTP مستقل با یک وب سرور را به یک "Session" متصل کنید. اولین باری که کاربر از وب سایت بازدید می‌کند، یک شناسه منحصر به فرد به روشی که برای سرور قابل تشخیص باشد ایجاد می‌گردد. این مقدار به URL اضافه می‌شود و سرور، کلاینت را به این URL هدایت می‌کند. هر زمان که سرور درخواستی برای Fat URL دریافت می‌کند، می‌تواند هر حالت افزایشی مرتبط با آن شناسه کاربری (سبدهای خرید، پروفایل‌ها و غیره) را جستجو کند و تمام لینک‌های خروجی را بازنویسی می‌کند تا آن‌ها را Fat نموده تا شناسه کاربری را حفظ کند.

از Fat URL ها می‌توان به منظور شناسایی کاربران هنگام مرور یک سایت استفاده کرد. اما این فناوری چندین مشکل جدی دارد. برخی از این مشکلات عبارتند از:

Ugly URLs

های نمایش داده شده در مرورگر برای کاربران جدید گیج کننده است.

Can't share URLs

Fat URL ها حاوی اطلاعات وضعیتی در مورد یک کاربر و جلسه خاص هستند. اگر آن URL را برای شخص دیگری پست کنید، ممکن است ناخواسته اطلاعات شخصی انباسته شده خود را به اشتراک بگذارید.

Breaks caching

تولید نسخه‌های خاص کاربر از هر URL به این معنی است که دیگر URL‌هایی که معمولاً به حافظه کش دسترسی دارند وجود ندارد.





Extra server load

سرور باید صفحات HTML را بازنویسی کند تا URL ها را Fat کند.

Escape hatches

برای کاربر بسیار آسان است که به طور تصادفی از جلسه Fat URL با پرش به سایت دیگری یا با درخواست یک URL خاص "escape" کند. Fat URL ها تنها در صورتی کار می‌کنند که کاربر به شدت لینک‌های از پیش اصلاح شده را دنبال کند. اگر کاربر escape کند، ممکن است پیشرفت خود را از دست بدهد (شاید یک سبد خرید پر شده) و باید دوباره شروع کند.

Not persistent across sessions

وقتی کاربر از سیستم خارج می‌شود، همه اطلاعات از بین می‌رود، مگر اینکه Fat URL خاص را نشانه‌گذاری کند.

Cookies

کوکی‌ها بهترین راه فعلی برای شناسایی کاربران و اجازه دادن به جلسات دائمی هستند. آن‌ها از بسیاری از مشکلات تکنیک‌های قبلی رنج نمی‌برند، اما اغلب در ارتباط با آن تکنیک‌ها برای ارزش بیشتر استفاده می‌شوند. کوکی‌ها ابتدا توسط نت اسکیپ توسعه داده شدند اما اکنون توسط همه مرورگرهای اصلی پشتیبانی می‌شوند.

از آنجایی که کوکی‌ها مهم هستند و هدرهای HTTP جدیدی را تعریف می‌کنند، ما آن‌ها را با جزئیات بیشتری نسبت به تکنیک‌های قبلی بررسی می‌کنیم. وجود کوکی‌ها بر روی Cache نیز تأثیر می‌گذارد و اکثر Cache ها و مرورگرهای ذخیره هر محتوای کوکی شده را ممنوع می‌کنند. بخش‌های بعدی جزئیات بیشتری را ارائه می‌دهند.

Types of Cookies

می‌توانید کوکی‌ها را به طور کلی به دو نوع دسته‌بندی کنید: کوکی‌های Session و کوکی‌های Persistent. کوکی Session یک کوکی موقت است که تنظیمات و Preference ها را هنگام حرکت کاربر در یک سایت پیگیری می‌کند. هنگامی که کاربر از مرورگر خارج می‌شود، یک کوکی Session حذف می‌شود. کوکی‌های Persistent می‌توانند بیشتر عمر کنند. آن‌ها بر روی دیسک ذخیره می‌شوند و پس از خروج مرورگر و راه اندازی مجدد کامپیوتر نیز جان سالم به در می‌برند. کوکی‌های Persistent اغلب برای حفظ پیکربندی مربوط به پروفایل یا نام ورود به سایتی که کاربر به طور دوره‌ای از آن بازدید می‌کند استفاده می‌شود.

تنها تفاوت بین کوکی‌های Session و کوکی‌های Persistent، زمانی است که منقضی می‌شوند. همانطور که بعداً خواهیم دید، اگر در یک کوکی پارامتر Discard آن تنظیم شده باشد یا اگر پارامتر



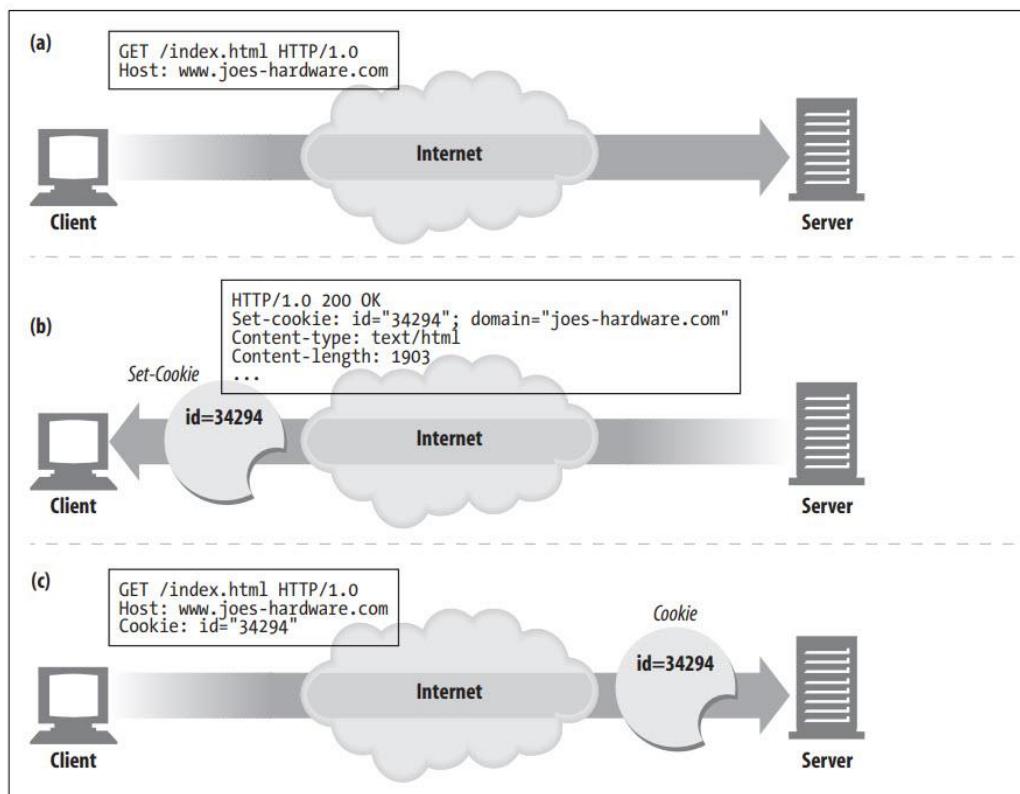


Session وجود نداشته باشد که زمان انقضای طولانی تری را نشان دهد، ما یک کوکی Max-Age یا Expires داریم.

How Cookies Work

کوکی‌ها مانند برچسب‌های "Hello, My Name Is" هستند که توسط سرورها بر روی کاربران چسبانده شده‌اند. هنگامی که یک کاربر از یک وب سایت بازدید می‌کند، وب سایت می‌تواند تمام برچسب‌های متصل شده توسط آن سرور به کاربر را بخواند.

اولین باری که کاربر از یک وب سایت بازدید می‌کند، وب سرور چیزی در مورد کاربر نمی‌داند (بخش a شکل زیر). سرور وب انتظار دارد که همین کاربر دوباره بازگردد، بنابراین می‌خواهد یک کوکی منحصر به فرد را بر روی کاربر بگذارد تا بتواند این کاربر را در آینده شناسایی کند. کوکی حاوی یک لیست دلخواه از اطلاعات است و با استفاده از هدرهای پاسخ HTTP مانند Set-Cookie2 یا Set-Cookie به کاربر پیوست می‌شود.



کوکی‌ها می‌توانند حاوی هر اطلاعاتی باشند، اما اغلب حاوی یک شماره شناسایی منحصر به فرد هستند که توسط سرور برای اهداف ردیابی ایجاد می‌شود. به عنوان مثال، در بخش b شکل بالا، سرور یک کوکی





بر روی کاربر قرار می‌دهد که به مقدار $id=34294$ اشاره می‌کند. سرور می‌تواند از این شماره برای جستجوی اطلاعات پایگاه داده‌ای که سرور برای بازدیدکنندگان خود جمع آوری می‌کند (سابقه خرید، اطلاعات آدرس و غیره) استفاده کند.

با این حال، کوکی‌ها فقط به شماره شناسه محدود نمی‌شوند. بسیاری از سرورهای وب انتخاب می‌کنند که اطلاعات را مستقیماً در کوکی‌ها نگه دارند. مثلاً:

Cookie: name="Brian Totty"; phone="555-1212"

مرورگر محتويات کوکی ارسال شده از سرور را در هدرهای Set-Cookie2 یا Set-Cookie به خاطر می‌آورد و مجموعه کوکی‌ها را در پایگاه داده کوکی مرورگر ذخیره می‌کند (آن را مانند یک چمدان با برچسب‌هایی از کشورهای مختلف در نظر بگیرید). هنگامی که کاربر در آینده به همان سایت باز می‌گردد (بخش C شکل بالا)، مرورگر کوکی‌هایی را که توسط آن سرور بر روی کاربر قرار داده است انتخاب می‌کند و آن‌ها را در هدر درخواست کوکی ارسال می‌کند.

Cookie Jar: Client-Side State

ایده اصلی کوکی‌ها این است که به مرورگر اجازه دهنند مجموعه‌ای از اطلاعات خاص سرور را جمع آوری کند و هر بار که کاربر آن را بازدید می‌کند این اطلاعات را به سرور ارائه دهد.

از آنجایی که مرورگر وظیفه ذخیره اطلاعات کوکی را بر عهده دارد، به این سیستم Client-Side State می‌گویند. نام رسمی مشخصات کوکی HTTP State Management Mechanism است.

Netscape Navigator cookies

مرورگرهای مختلف، کوکی‌ها را به روش‌های مختلف ذخیره می‌کنند. کوکی‌ها را در Netscape Navigator یک فایل متنی به نام **cookies.txt** ذخیره می‌کند. مثلاً:





```
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file! Do not edit.
#
# domain           allh   path    secure expires   name      value
#
www.fedex.com        FALSE  /       FALSE  1136109676 cc        /us/
.bankofamericaonline.com TRUE   /       FALSE  1009789256 state    CA
.cnn.com             TRUE   /       FALSE  1035069235 SelEdition www
secure.eepulse.net   FALSE  /eePulse FALSE  1007162968 cid      %FE%FF%002
www.reformamt.org    TRUE   /forum   FALSE  1033761379 LastVisit 1003520952
www.reformamt.org    TRUE   /forum   FALSE  1033761379 UserName  Guest
...
...
```

هر خط از فایل متنی نشان دهنده یک کوکی است. هفت فیلد جدا شده با تپ وجود دارد:

domain

نشان دهنده Domain مربوط به کوکی است.

allh

نشان دهنده این است که همه میزبان‌های یک دامنه، کوکی را دریافت نموده، یا فقط میزبان خاص نام‌گذاری شده آن‌ها را دریافت نمایند.

path

نشان دهنده پیشوند مسیر در دامنه مرتبط با کوکی است.

secure

نشان دهنده این است که آیا ما باید این کوکی را فقط در صورت داشتن اتصال SSL ارسال کنیم یا خیر.

expiration

نشان دهنده تاریخ انقضای کوکی به شکل ثانیه و از ۱ ژانویه ۰۰:۰۰:۰۰ GMT ۱۹۷۰ است.

name

نشان دهنده نام متغیر کوکی است.

value

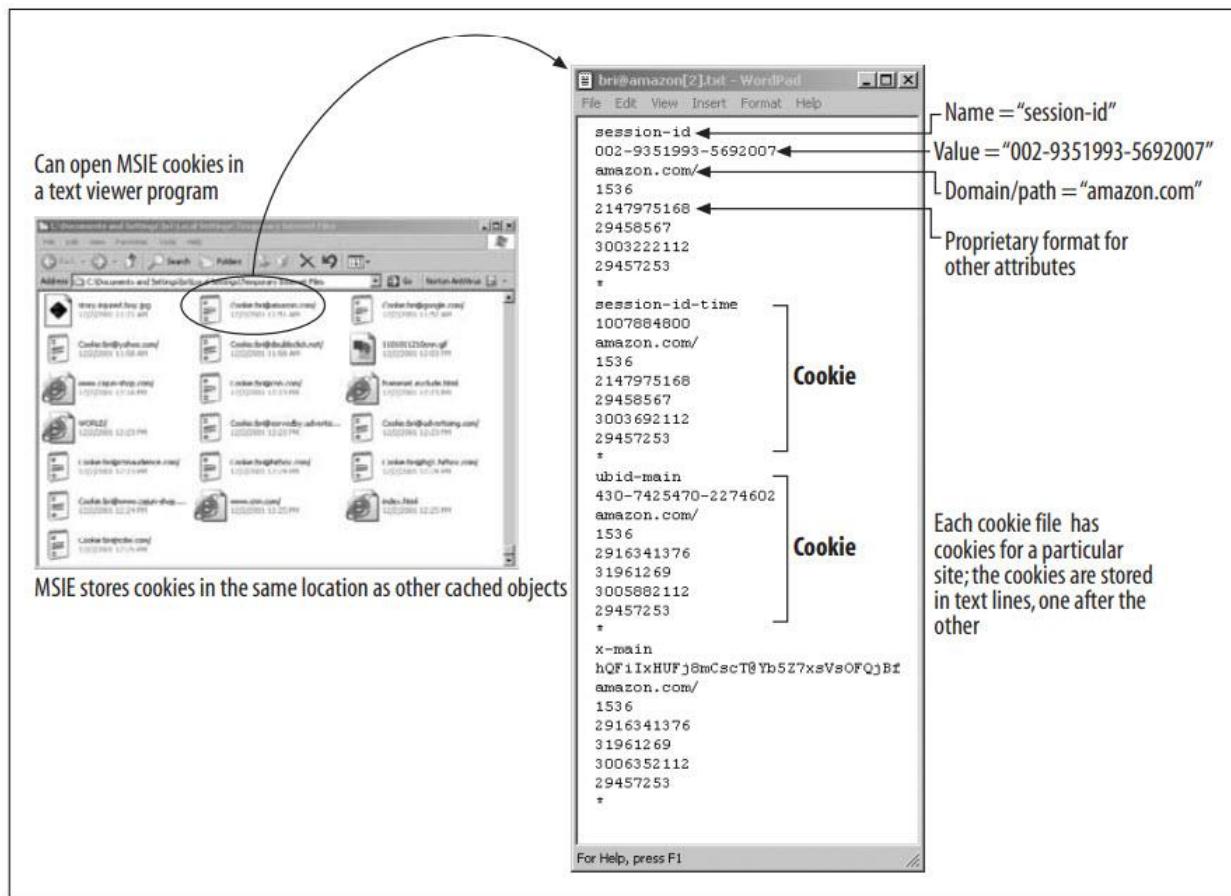
نشان دهنده مقدار متغیر کوکی است.





Microsoft Internet Explorer cookies

مايكروسافت اينترنت اكسپلورر، کوکي ها را در فايل های متنی جداگانه در ديركتوري Cache ذخیره می کند. همانطور که در شکل زير نشان داده شده است، می توانيد اين ديركتوري را برای مشاهده کوکي ها مرور کنيد. فرمت فايل های کوکي اينترنت اكسپلورر اختصاصي است، اما بسياري از زمينه ها به راحتی قابل درک هستند. هر کوکي يكی پس از ديگري در فايل ذخیره می شود و هر کوکي از چندين خط تشکيل شده است.



خط اول هر کوکي در فايل حاوي نام متغير کوکي است. خط بعدی مقدار متغير است. خط سوم شامل دامنه و مسیر است. خطوط باقی مانده داده های اختصاصی (احتمالاً شامل تاریخ و سایر فلگ ها) هستند.

Different Cookies for Different Sites

یک مرورگر می تواند صدها یا هزاران کوکی در کوکی داخلی خود داشته باشد، اما مرورگرها همه کوکی ها را به هر سایتی ارسال نمی کنند. در واقع، آن ها معمولاً فقط دو یا سه کوکی به هر سایت ارسال می کنند. در اینجا دليل آن است:





- جابجایی تمام آن بایتهای کوکی عملکرد را به طور چشمگیری کاهش می‌دهد. مرورگرها در واقع بایتهای کوکی بیشتری نسبت به بایتهای محتواهای واقعی جابجا می‌کنند!
 - بیشتر این کوکی‌ها برای اکثر سایتها به دلیل داشتن جفت‌های `name/value` خاص سرور، بیهودگی غیرقابل تشخیص هستند.
 - ارسال همه کوکی‌ها به همه سایتها باعث نگرانی بالقوه حفظ حریم خصوصی می‌شود، زیرا سایتها می‌دانند که به آن‌ها اعتماد ندارید اطلاعاتی را که فقط برای سایت دیگری در نظر گرفته‌اید دریافت می‌کنند.
- به طور کلی، یک مرورگر تنها کوکی‌هایی را که سرور ایجاد کرده است به سرور ارسال می‌کند. کوکی‌های تولید شده توسط `joes-hardware.com` به `joes-hardware.com` ارسال می‌شوند و نه به `.marys-movies.com` یا `books.com`
- بسیاری از وبسایتها برای مدیریت تبلیغات با فروشنندگان شخص ثالث قرارداد می‌بندند. این تبلیغات به گونه‌ای ساخته می‌شوند که به نظر می‌رسند بخش جدایی‌ناپذیر وبسایت هستند و کوکی‌های Push Persistent را می‌دهند. هنگامی که کاربر به وبسایت دیگری که توسط همان شرکت تبلیغاتی سرویس دهی می‌شود، می‌رود، کوکی‌های Persistent که قبلاً تنظیم شده‌اند دوباره توسط مرورگر ارسال می‌شود (زیرا دامنه‌ها مطابقت دارند). یک شرکت بازاریابی می‌تواند از این تکنیک، همراه با هدر `Referer`، برای ایجاد یک مجموعه داده جامع از پروفایل‌های کاربر و عادت‌های مرور‌وی استفاده کند. مرورگرهای مدرن به شما امکان می‌دهند تنظیمات حریم خصوصی را برای محدود کردن کوکی‌های شخص ثالث پیکربندی کنید.

Cookie Domain attribute

سروری که یک کوکی ایجاد می‌کند، می‌تواند با افزودن یک ویژگی `Domain` به هدر پاسخ Set-Cookie کنترل کند که کدام سایتها آن کوکی را ببینند. به عنوان مثال، هدر پاسخ HTTP زیر به مرورگر می‌گوید که کوکی `user=mary17` را به هر سایتی در دامنه `airtravelbargains.com` ارسال کند:

`Set-cookie: user="mary17"; domain="airtravelbargains.com"`

اگر کاربر از `specials.airtravelbargains.com`، `www.airtravelbargains.com` یا هر سایتی که به `airtravelbargains.com` ختم می‌شود بازدید کند، هدر کوکی زیر صادر می‌شود:

`Cookie: user="mary17"`





Cookie Path attribute

مشخصات کوکی حتی به شما امکان می‌دهد کوکی‌ها را با بخش‌هایی از وبسایتها مرتبط کنید. این کار با استفاده از ویژگی Path انجام می‌شود که نشان دهنده پیشوند مسیر URL است که در آن هر کوکی معتبر است.

برای مثال، یک وب سرور ممکن است بین دو سازمان به اشتراک گذاشته شود که هر کدام کوکی‌های جداگانه دارند. سایت www.airtravelbargains.com ممکن است بخشی از وب سایت خود را با استفاده از یک کوکی جداگانه برای پیگیری اندازه ماشین ترجیحی کاربر به اجاره خودرو اختصاص دهد - مثلاً <http://www.airtravelbargains.com/autos/>. یک کوکی مخصوص اجاره خودکار ممکن است به این صورت ایجاد شود:

Set-cookie: pref=compact; domain="airtravelbargains.com"; path=/autos/

اگر کاربر به <http://www.airtravelbargains.com/specials.html> مراجعه کند، فقط این کوکی را دریافت خواهد کرد:

Cookie: user="mary17"

اما اگر او به <http://www.airtravelbargains.com/autos/cheapo/index.html> برود، هر دوی این کوکی‌ها را دریافت می‌کند:

Cookie: user="mary17"

Cookie: pref=compact

بنابراین، کوکی‌ها تکه‌هایی از حالت هستند که توسط سرورها بر روی کلاینت قرار می‌گیرند، توسط کلاینت‌ها نگهداری می‌شوند و تنها به آن سایتها که مناسب هستند بازگردانده می‌شوند. بیایید با جزئیات بیشتری به فناوری و استانداردهای کوکی نگاه کنیم.

Cookie Ingredients

دو نسخه مختلف از مشخصات کوکی در حال استفاده وجود دارد: کوکی‌های نسخه 0 (که گاهی اوقات «کوکی‌های Netscape» نامیده می‌شوند)، و کوکی‌های نسخه 1 («RFC 2965»). کوکی‌های نسخه 1 یک برنامه افروزنی کم استفاده از کوکی‌های نسخه 0 هستند.





مشخصات کوکی نسخه 0 یا نسخه 1 به عنوان بخشی از مشخصات HTTP/1.1 ثبت نشده است. دو سند کمکی اولیه وجود دارد که به بهترین شکل استفاده از کوکی‌ها را توصیف می‌کند که در جدول زیر خلاصه شده است.

Title	Description	Location
Persistent Client State: HTTP Cookies	Original Netscape cookie standard	http://home.netscape.com/newsref/std/cookie_spec.html
RFC 2965: HTTP State Management Mechanism	October 2000 cookie standard, obsoletes RFC 2109	http://www.ietf.org/rfc/rfc2965.txt

Version 0 (Netscape) Cookies

مشخصات اولیه کوکی توسط **Netscape** تعریف شده است. این کوکی‌های «نسخه 0» هدر پاسخ **Set-Cookie** هدر درخواست کوکی و فیلدهای موجود برای کنترل کوکی‌ها را تعریف می‌کنند. کوکی‌های نسخه 0 به شکل زیر هستند:

Set-Cookie: name=value [; expires=date] [; path=path] [; domain=domain] [; secure]

Cookie: name1=value1 [; name2=value2] ...

Version 0 Set-Cookie header

هدر **Set-Cookie** یک نام کوکی و مقدار کوکی اجباری دارد. می‌توان آن را با ویژگی‌های کوکی اختیاری، که با نقطه ویرگول از هم جدا می‌شوند، دنبال کرد. فیلدهای **Set-Cookie** در ادامه توضیح داده شده است.

NAME=VALUE

اجباری. هر دو **NAME** و **VALUE** دنباله‌ای از کاراکترها هستند، به استثنای نقطه ویرگول، کاما، علامت مساوی و فضای خالی، مگر اینکه در گیومه‌های دوگانه نقل قول شوند. وب سرور می‌تواند هر ارتباط **NAME=VALUE** را ایجاد کند که در بازدیدهای بعدی از سایت به وب سرور بازگردانده می‌شود.

Set-Cookie: customer=Mary

Expires

اختیاری. این ویژگی یک رشته تاریخ را مشخص نموده که طول عمر معتبر آن کوکی می‌باشد. پس از رسیدن به تاریخ انقضا، کوکی دیگر ذخیره یا ارائه نخواهد شد. تاریخ به صورت زیر تنظیم شده است:

Weekday, DD-Mon-YY HH:MM:SS GMT





تنها منطقه زمانی قانونی **GMT** است و جداگاندهای بین عناصر تاریخ باید خط تیره باشند. اگر **Expires** مشخص نشده باشد، کوکی پس از پایان جلسه کاربر منقضی می‌شود.

Set-Cookie: foo=bar; expires=Wednesday, 09-Nov-99 23:12:40 GMT

Domain

اختیاری. یک مرورگر کوکی را فقط به نام میزبان سرور در دامنه مشخص شده ارسال می‌کند. این به سرورها اجازه می‌دهد کوکی‌ها را فقط به دامنه‌ای خاصی محدود کنند. دامنه "acme.com" با نام میزبان www.cnn.com مطابقت داشته اما با shipping.crate.acme.com و anvil.acme.com ندارد.

فقط میزبان‌های درون دامنه مشخص شده می‌توانند یک کوکی برای یک دامنه تنظیم کنند و دامنه‌ها باید حداقل دو یا سه نقطه در خود داشته باشند تا از دامنه‌هایی به شکل .edu..com و «va.us» جلوگیری شود. هر دامنه‌ای که در مجموعه ثابت **Top-Level Domain** ویژه فهرست شده در اینجا قرار می‌گیرد، تنها به دو دوره نیاز دارد. هر دامنه دیگری به حداقل سه دامنه نیاز دارد. Top-Level Domain های ویژه عبارتند از: ..edu ..com ..pro ..aero ..coop ..museum ..name ..info ..biz ..int ..mil ..gov ..org ..net.

اگر دامنه مشخص نشده باشد، نام میزبان سروری که پاسخ Set-Cookie را ایجاد کرده است، پیشفرض است.

Set-Cookie: SHIPPING=FEDEX; domain="joes-hardware.com"

Path

اختیاری. این ویژگی به شما امکان می‌دهد کوکی‌ها را به اسناد خاصی در یک سرور اختصاص دهید. اگر ویژگی **Path** پیشوند یک مسیر URL باشد، یک کوکی می‌تواند پیوست شود. مسیر "/" با "/foo" و "/foo/bar.html" مطابقت دارد. مسیر "/" با همه چیز در دامنه مطابقت دارد.

اگر مسیر مشخص نشده باشد، روی مسیر URL که پاسخ Set-Cookie را ایجاد کرده تنظیم می‌شود.

Set-Cookie: lastorder=00183; path=/orders

Secure

اختیاری. اگر این ویژگی گنجانده شود، کوکی تنها در صورتی ارسال می‌شود که HTTP از اتصال امن SSL استفاده کند.

Set-Cookie: private_id=519; secure





Version 0 Cookie header

هنگامی که یک کلاینت درخواستی را ارسال می‌کند، این درخواست شامل تمام کوکی‌های منقضی نشده است که با Path و Domain مطابقت دارند. همه کوکی‌ها در یک هدر کوکی ترکیب می‌شوند:

Cookie: session-id=002-1145265-8016838; session-id-time=1007884800

Version 1 (RFC 2965) Cookies

یک نسخه توسعه یافته از کوکی‌ها در RFC 2965 (قبل‌اً RFC 2109) تعریف شده است. این استاندارد نسخه 1 هدرهای Set-Cookie2 و Cookie2 را معرفی می‌کند، اما با سیستم نسخه 0 نیز تعامل دارد.

استاندارد کوکی 2965 RFC کمی پیچیده‌تر از استاندارد اصلی Netscape است و هنوز به طور کامل پشتیبانی نمی‌شود. تغییرات عمدۀ کوکی‌های RFC 2965 عبارتند از:

- متن توصیفی را با هر کوکی مرتبط کنید تا هدف آن را توضیح دهید.
- پشتیبانی از تخریب اجباری کوکی‌ها در هنگام خروج از مرورگر، بدون در نظر گرفتن انقضای حداکثر سن پیری (Max-Age) کوکی‌ها در ثانیه به جای تاریخ‌های مطلق امكان کنترل کوکی‌ها با شماره پورت URL، نه فقط Path و Domain
- هدر کوکی Port، Domain و Path Filters را (در صورت وجود) حمل می‌کند.
- شماره نسخه برای قابلیت همکاری \$ prefix در هدر کوکی برای تشخیص کلمات کلیدی اضافی از نامهای کاربری

ساختار کوکی نسخه 1 به شرح زیر است:





```
set-cookie      = "Set-Cookie2:" cookies
cookies        = 1#cookie
cookie         = NAME "=" VALUE *(";" set-cookie-av)
NAME           = attr
VALUE          = value
set-cookie-av   = "Comment" "=" value
| "CommentURL" "=" <"> http_URL <">
| "Discard"
| "Domain" "=" value
| "Max-Age" "=" value
| "Path" "=" value
| "Port" [ "=" <"> portlist <"> ]
| "Secure"
| "Version" "=" 1*DIGIT
portlist       = 1#portnum
portnum        = 1*DIGIT

cookie         = "Cookie:" cookie-version 1*((;" | ",") cookie-value)
cookie-value   = NAME "=" VALUE [";" path] [";" domain] [";" port]
cookie-version = "$Version" "=" value
NAME           = attr
VALUE          = value
path            = "$Path" "=" value
domain          = "$Domain" "=" value
port             = "$Port" [ "=" <"> value <"> ]

cookie2 = "Cookie2:" cookie-version
```

Version 1 Set-Cookie2 header

ویژگی‌های بیشتری در استاندارد کوکی نسخه 1 نسبت به استاندارد Netscape موجود است. در ادامه خلاصه‌ای سریع از ویژگی‌ها را ارائه داده می‌شود. برای توضیح بیشتر به RFC 2965 مراجعه کنید.

NAME=VALUE

اجباری. وب سرور می‌تواند هر ارتباط NAME=VALUE را ایجاد کند که در بازدیدهای بعدی از سایت به وب سرور بازگردانده می‌شود. نام نباید با "\$" شروع شود، زیرا این کاراکتر رزرو شده است.

Version

اجباری. مقدار این ویژگی یک عدد صحیح است که مطابق با نسخه مشخصات کوکی است. RFC 2965 مربوط به نسخه 1 است.

Set-Cookie2: Part="Rocket_Launcher_0001"; Version="1"





Comment

اختیاری. این ویژگی نشان می‌دهد که چگونه یک سرور قصد استفاده از کوکی را دارد. کاربر می‌تواند این خطمنشی را بررسی کند تا تصمیم بگیرد که آیا یک جلسه با این کوکی مجاز است یا خیر. مقدار باید در ساختار کدگذاری UTF-8 باشد.

CommentURL

اختیاری. این ویژگی یک نشانگر URL به مستندات دقیق در مورد هدف و خط مشی یک کوکی را ارائه می‌دهد. کاربر می‌تواند این خطمنشی را بررسی کند تا تصمیم بگیرد که آیا یک جلسه با این کوکی مجاز است یا خیر.

Discard

اختیاری. اگر این ویژگی وجود داشته باشد، به کلاینت دستور می‌دهد که کوکی را پس از پایان برنامه کلاینت کنار بگذارد.

Domain

اختیاری. یک مرورگر کوکی را فقط به نام میزبان سرور در دامنه مشخص شده ارسال می‌کند. این به سرورها اجازه می‌دهد کوکی‌ها را فقط به دامنه‌های خاصی محدود کنند. دامنه "acme.com" با نام میزبان "www.cnn.com" و "shipping.crate.acme.com" و "anvil.acme.com" مطابقت داشته، اما با Netscape است، اما چند قانون اضافی وجود دارد. برای جزئیات بیشتر به RFC 2965 مراجعه کنید.

Max-Age

اختیاری. مقدار این ویژگی یک عدد صحیح است که طول عمر کوکی را بر حسب ثانیه تنظیم می‌کند. کلاینت‌ها باید سن کوکی را طبق قوانین محاسبه سن 1.1 HTTP محاسبه کنند. وقتی سن یک کوکی از حداکثر سن بیشتر شود، کلاینت باید کوکی را دور بیندازد. مقدار صفر به این معنی است که کوکی با آن نام باید فوراً کنار گذاشته شود.

Path

اختیاری. این ویژگی به شما امکان می‌دهد کوکی‌ها را به اسناد خاصی در یک سرور اختصاص دهید. اگر ویژگی Path پیشوند یک مسیر URL باشد، یک کوکی می‌تواند پیوست شود. مسیر "/foo" با





" / " و "/foobar" مطابقت دارد. مسیر "/" با همه چیز در دامنه مطابقت دارد. اگر مسیر مشخص نشده باشد، روی مسیر URL که پاسخ Set-Cookie را ایجاد کرده تنظیم می‌شود.

Port

اختیاری. این ویژگی می‌تواند به عنوان یک کلمه کلیدی به تنها یابش، یا می‌تواند شامل فهرستی از پورت‌های جدا شده با کاما باشد که ممکن است یک کوکی روی آن اعمال شود. اگر لیست پورت وجود داشته باشد، کوکی را می‌توان فقط برای سرورهای آن‌ها با یک پورت در لیست مطابقت دارد ارائه کرد. اگر کلمه کلیدی به صورت مجزا ارائه شود، کوکی را می‌توان فقط به شماره پورت سرور پاسخ دهنده فعلی ارائه کرد.

Set-Cookie2: foo="bar"; Version="1"; Port="80,81,8080"

Set-Cookie2: foo="bar"; Version="1"; Port

Secure

اختیاری. اگر این ویژگی گنجانده شود، کوکی تنها در صورتی ارسال می‌شود که HTTP از اتصال امن SSL استفاده کند.

Version 1 Cookie header

کوکی‌های نسخه 1 اطلاعات بیشتری را در مورد هر کوکی تحویل داده شده با خود همراه می‌کنند و فیلترهایی را که هر کوکی ارسال می‌شود، توصیف می‌کند. هر کوکی منطبق باید شامل هر ویژگی Domain یا Path از هدرهای Set-Cookie2 مربوطه باشد.

به عنوان مثال، فرض کنید کلاینت این پنج پاسخ Set-Cookie2 را در گذشته از وب سایت www.joeshardware.com دریافت کرده است:

Set-Cookie2: ID="29046"; Domain=".joes-hardware.com"

Set-Cookie2: color=blue

Set-Cookie2: support-pref="L2"; Domain="customer-care.joes-hardware.com"

Set-Cookie2: Coupon="hammer027"; Version="1"; Path="/tools"

Set-Cookie2: Coupon="handvac103"; Version="1"; Path="/tools/cordless"





اگر کلاینت درخواست دیگری برای مسیر /tools/cordless/specials.html بدهد، از هدر طولانی Mancnd زیر عبور می‌کند:

```
Cookie: $Version="1";  
ID="29046"; $Domain=".joes-hardware.com";  
color="blue";  
Coupon="hammer027"; $Path="/tools";  
Coupon="handvac103"; $Path="/tools/cordless"
```

توجه داشته باشید که تمام کوکی‌های منطبق با فیلترهای Set-Cookie2 تحویل داده می‌شوند و کلمات کلیدی رزرو شده با علامت دلار (\$) شروع می‌شوند.

Version 1 Cookie2 header and version negotiation

هدر Cookie2request برای مذاکره در مورد قابلیت همکاری بین کلاینتها و سرورهایی استفاده می‌شود که نسخه‌های مختلف مشخصات کوکی را درک می‌کنند. هدر Cookie2 به سرور توصیه می‌کند که کوکی‌های سبک جدید را فهمیده و نسخه استاندارد کوکی پشتیبانی شده را ارائه کند (بهتر است آن را Version بنامیم):

Cookie2: \$Version="1"

اگر سرور کوکی‌های سبک جدید را بفهمد، Set-Cookie2header را می‌شناسد و باید هدرهای پاسخ Set-Cookie2 (به جای Set-Cookie) را ارسال کند. اگر یک کلاینت هم یک Set-Cookie2 و یک Set-Cookie قدیمی را نادیده می‌گیرد.

اگر یک کلاینت از کوکی‌های نسخه 0 و نسخه 1 پشتیبانی می‌کند اما هدر SetCookie نسخه 0 را از سرور دریافت می‌کند، باید کوکی‌ها را با هدر کوکی نسخه 0 ارسال کند. با این حال، کلاینت همچنین باید \$Version="1" را ارسال کند تا به سرور نشان دهد که می‌تواند ارتقا یابد.

Cookies and Session Tracking

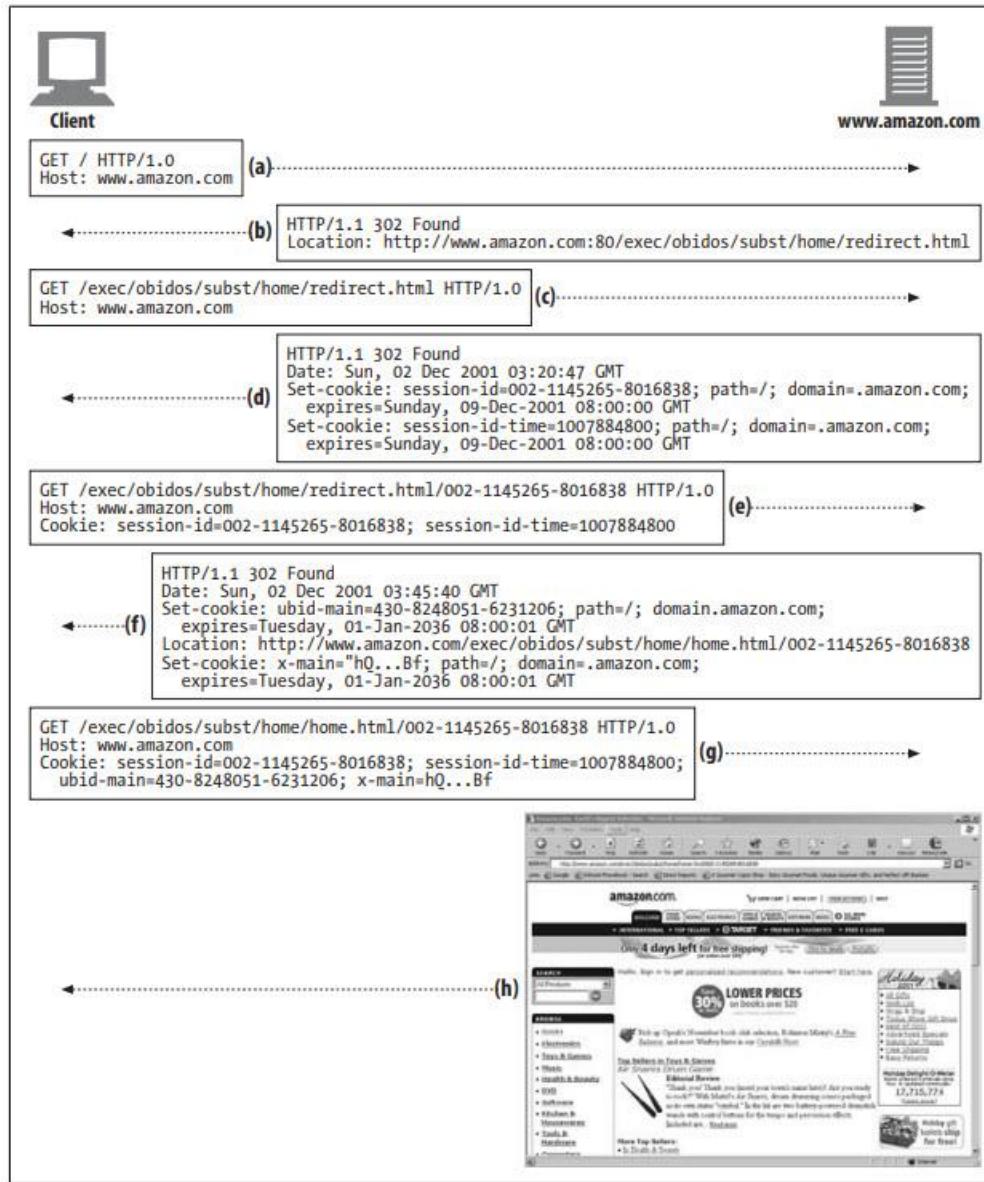
از کوکی‌ها می‌توان برای ردیابی کاربران در حالی که چندین تراکنش با یک وب سایت انجام می‌دهند استفاده کرد. وبسایت‌های تجارت الکترونیک از کوکی‌های جلسه برای پیگیری سبد خرید کاربران در هنگام مرور استفاده می‌کنند. بیایید سایت خرید محبوب Amazon.com را مثال بزنیم.





هنگامی که <http://www.amazon.com> را در مرورگر خود تایپ می کنید، زنجیرهای از تراکنش‌ها را شروع می کنید که در آن سرور وب اطلاعات شناسایی را از طریق یک سری تغییر مسیر، بازنویسی URL و تنظیمات کوکی به آن متصل می کند.

شکل زیر یک توالی تراکنش گرفته شده از بازدید Amazon.com را نشان می‌دهد:



بخش a شکل: مرورگر برای اولین بار درخواستی را برای صفحه اصلی Amazon.com ارسال می‌کند.

بخش b شکل: سرور کلاینت را به آدرس اینترنتی نرم افزار تجارت الکترونیک هدایت می‌کند.



بخش c شکل: کلاینت درخواستی را به URL هدایت شده ارسال می‌کند.

بخش d شکل: سرور دو کوکی Session را روی پاسخ قرار می‌دهد و کاربر را به URL دیگری هدایت می‌کند، بنابراین کلاینت با پیوست این کوکی‌ها دوباره درخواست را ارسال می‌کند. این URL جدید یک Fat URL است، به این معنی که برخی از حالت‌ها در URL تعبیه شده است. اگر کلاینت کوکی‌ها را غیرفعال کرده باشد، تا زمانی که کاربر لینک‌های Fat URL ایجاد شده توسط Amazon.com را دنبال کند و سایت را ترک نکند، هنوز می‌توان برخی از شناسایی‌های اولیه را انجام داد.

بخش e شکل: کلاینت URL جدید را درخواست می‌کند، اما اکنون دو کوکی پیوست شده را ارسال می‌کند.

بخش f شکل: سرور به صفحه home.html هدایت می‌شود و دو کوکی دیگر را پیوست می‌کند.

بخش g شکل: کلاینت صفحه home.html را واکشی می‌کند و هر چهار کوکی را ارسال می‌کند.

بخش h شکل: سرور محتوا را بازگردانی می‌کند.

Cookies and Caching

هنگام ذخیره اسنادی که با تراکنش‌های کوکی مرتبط هستند، باید مراقب باشید. شما نمی‌خواهید کوکی‌های کاربر قبلی را به یک کاربر اختصاص دهید یا بدتر از آن، محتوای سند شخصی‌شده شخص دیگری را به یک کاربر نشان دهید.

قوانین مربوط به کوکی‌ها و Cache به خوبی ایجاد نشده است. در اینجا چند اصل راهنمایی برای برخورد با Cache آورده شده است:

Mark documents uncacheable if they are

مالک سند بهتر می‌داند که آیا یک سند غیرقابل ذخیره (Uncacheable) است. اگر اسناد غیرقابل ذخیره هستند، صریحاً علامت‌گذاری کنید—بهویژه، از Cache-Control: nocache="Set-Cookie" استفاده کنید. روش عمومی‌تر دیگر استفاده از Cache-Control: public "Cookie" قابل ذخیره‌سازی هستند که منجر به صرفه‌جویی در پهنای باند وب خواهند شد.

Be cautious about caching Set-Cookie headers

اگر پاسخی دارای هدر Set-Cookie باشد، می‌توانید بدن را در Cache ذخیره کنید (مگر اینکه خلاف آن گفته شود)، اما باید در مورد ذخیره کردن هدر Set-Cookie بسیار محتاط





باشید. اگر یک هدر Set-Cookie را برای چندین کاربر ارسال کنید، ممکن است هدف گذاری کاربر را شکست دهید.

برخی از Cache ها، هدر Set-Cookie را قبل از ذخیره پاسخ در Cache حذف می‌کنند، اما این نیز می‌تواند مشکلاتی ایجاد کند، زیرا کلاینت‌هایی که از Cache ارائه می‌شوند، دیگر کوکی‌هایی را دریافت نمی‌کنند که معمولاً بدون Cache انجام می‌شوند. این وضعیت را می‌توان با وادار کردن Cache برای تأیید مجدد هر درخواست با سرور مبدا و ادغام هر هدر Set-Cookie بازگشتی با پاسخ کلاینت، بهبود بخشید. سرور مبدا می‌تواند با افزودن این هدر به کپی Cache شده، چنین اعتبار سنجی مجددی را دیکته کند:

Cache-Control: must-revalidate, max-age=0

Cache های محافظه‌کارتر ممکن است از Cache کردن پاسخ‌هایی که دارای هدر SetCookie هستند خودداری کنند، حتی اگر محتوا واقعاً قابل ذخیره‌سازی باشد. برخی از Cache ها زمانی که تصاویر-Set-Cookie در Cache ذخیره می‌شوند، حالت‌ها را مجاز می‌کنند، اما متن را نه.

Be cautious about requests with Cookie headers

هنگامی که درخواستی با هدر کوکی وارد می‌شود، اشاره‌ای به این موضوع دارد که محتوای حاصل ممکن است شخصی شود. محتوای شخصی‌شده باید غیرقابل ذخیره‌سازی علامت‌گذاری شود، اما ممکن است برخی از سرورها به اشتباه این محتوا را غیرقابل ذخیره‌سازی علامت‌گذاری نکنند.

Cache های محافظه کار ممکن است تصمیم بگیرند که هیچ سندی را که در پاسخ به درخواستی با عنوان کوکی ارائه می‌شود، ذخیره نکنند و دوباره، برخی از Cache ها حالت‌هایی را در زمانی که تصاویر کوکی‌شده در Cache ذخیره می‌شوند، اجازه می‌دهند، اما متن را نه. سیاست پذیرفته‌شده‌تر این است که تصاویر با هدرهای کوکی را در Cache با زمان انقضای صفر، نگه دارید، بدین ترتیب هر بار مجبور به تأیید مجدد می‌شود.

Cookies, Security, and Privacy

اعتقاد بر این است که کوکی‌ها به خودی خود یک خطر امنیتی فوق العاده نیستند، زیرا می‌توان آن‌ها را غیرفعال کرد و بسیاری از ردیابی‌ها را می‌توان از طریق تجزیه و تحلیل گزارش یا ابزارهای دیگر انجام داد. در واقع، با ارائه یک روش استاندارد و موشکافانه برای نگهداری اطلاعات شخصی در پایگاه‌های داده راه دور





و استفاده از کوکی‌های ناشناس به عنوان کلید، می‌توان فرکانس ارتباط داده‌های حساس از کاربر به سرور را کاهش داد.

با این حال، خوب است در هنگام برخورد با حريم خصوصی و ردیابی کاربران محتاط باشد، زیرا همیشه امکان سوء استفاده وجود دارد. بزرگترین سوء استفاده از وب سایتها شخص ثالث است که از کوکی‌های Persistent برای ردیابی کاربران استفاده می‌کنند. این عمل، همراه با آدرس‌های IP و اطلاعات هدر Referer، این شرکت‌های بازاریابی را قادر می‌سازد تا پروفایل‌های کاربر و الگوهای مرور نسبتاً دقیقی بسازند.

با وجود همه تبلیغات منفی، عقل مرسوم این است که رسیدگی به جلسه و راحتی تراکنش‌های کوکی‌ها بر بیشتر خطرات برتری دارد، البته باید در مورد افرادی که اطلاعات شخصی را در اختیار آن‌ها قرار می‌دهید و خطمنشی‌های رازداری سایتها را بررسی می‌کنید احتیاط کنید.

Computer Incident Advisory Capability (بخشی از وزارت انرژی ایالات متحده) در سال ۱۹۹۸ ارزیابی ای از خطرات بیش از حد ارائه شده کوکی‌ها نوشت. در اینجا گزیده‌ای از آن گزارش آمده است:

CIAC I-034: Internet Cookies

(<http://www.ciac.org/ciac/bulletins/i-034.shtml>)

کوکی‌ها قطعات کوتاهی از داده‌ها هستند که توسط سرورهای وب برای کمک به شناسایی کاربران وب مورد استفاده قرار می‌گیرند. مفاهیم و شایعات رایج در مورد کارهایی که یک کوکی می‌تواند انجام دهد تقریباً به ابعاد عرفانی رسیده است و کاربران را ترسانده و مدیران آن‌ها را نگران کرده است.

ارزیابی آسیب‌پذیری:

آسیب‌پذیری سیستم‌ها در برابر آسیب یا جاسوسی با استفاده از کوکی‌های مرورگر وب اساساً وجود ندارد. کوکی‌ها فقط می‌توانند به سرور وب اطلاع دهند که قبلاً در آنجا بوده‌اید یا نه و می‌توانند دفعه بعد که بازدید می‌کنید، اطلاعات کوتاهی (مانند شماره کاربر) را از سرور وب به خود ارسال کنند. بیشتر کوکی‌ها فقط تا زمانی دوام می‌آورند که مرورگر خود را باز نگه داشته اید و در صورت ترک آن از بین می‌روند. نوع دوم کوکی معروف به کوکی Persistent دارای تاریخ انقضا است و تا آن تاریخ روی دیسک شما ذخیره می‌شود. یک کوکی Persistent می‌تواند برای ردیابی عادات مرور کاربر با شناسایی او در زمان بازگشت به سایت مورد استفاده قرار گیرد. اطلاعاتی در مورد اینکه از کجا آمده‌اید و از چه صفحات وب بازدید می‌کنید، قبلاً در فایل‌های گزارش وب سرور وجود دارد و همچنین می‌تواند برای ردیابی عادات مرور کاربران استفاده شود، کوکی‌ها فقط کار را آسان‌تر می‌کنند.





For More Information

Cookies: Simon St.Laurent, McGraw-Hill

<http://www.ietf.org/rfc/rfc2965.txt>

<http://www.ietf.org/rfc/rfc2964.txt>

http://home.netscape.com/newsref/std/cookie_spec.html





فصل دوازدهم – Basic Authentication

میلیون‌ها نفر از وب برای انجام تراکنش‌های خصوصی و دسترسی به داده‌های خصوصی استفاده می‌کنند. وب دسترسی به این اطلاعات را بسیار آسان می‌کند، اما آسان به اندازه کافی خوب نیست. ما نیاز به اطمینان داریم که چه کسی می‌تواند به داده‌های حساس ما نگاه کند و چه کسی می‌تواند تراکنش‌های ممتاز ما را انجام دهد. همه اطلاعات برای عموم مردم در نظر گرفته نشده است.

ما باید احساس راحتی کنیم که کاربران غیرمجاز نمی‌توانند پروفایل‌های سفر آنلاین ما را مشاهده کنند یا اسناد را بدون رضایت ما در وب سایتها می‌ منتشر کنند. ما باید مطمئن شویم که حساس‌ترین اسناد برنامه‌ریزی شرکتی ما در دسترس اعضای غیرمجاز نیست.

سرورها به راهی نیاز دارند تا هویت کاربر را تصدیق نمایند. هنگامی که یک سرور هویت کاربر را بداند، می‌تواند تصمیم بگیرد که کاربر می‌تواند به کدام تراکنش‌ها و منابع دسترسی داشته باشد. احراز هویت به معنای اثبات این موضوع است که شما چه کسی هستید. معمولاً شما با ارائه یک نام کاربری و یک رمز عبور مخفی احراز هویت می‌کنید. HTTP یک تسهیلات بومی برای احراز هویت HTTP فراهم می‌کند. در حالی که مطمئناً می‌توانید قابلیت‌های احراز هویت خود را در بالای فرم‌ها و کوکی‌های HTTP قرار دهید، برای بسیاری از موقعیت‌ها، احراز هویت بومی HTTP به خوبی مطابقت دارد.

این فصل از کتاب به مبحث احراز هویت HTTP پرداخته و رایج‌ترین شکل احراز هویت HTTP، احراز هویت اولیه یا Basic Authentication را مورد بحث قرار می‌دهد. در فصل بعدی به تکنیک قدرتمندتری به نام احراز هویت Digest خواهیم پرداخت.

Authentication

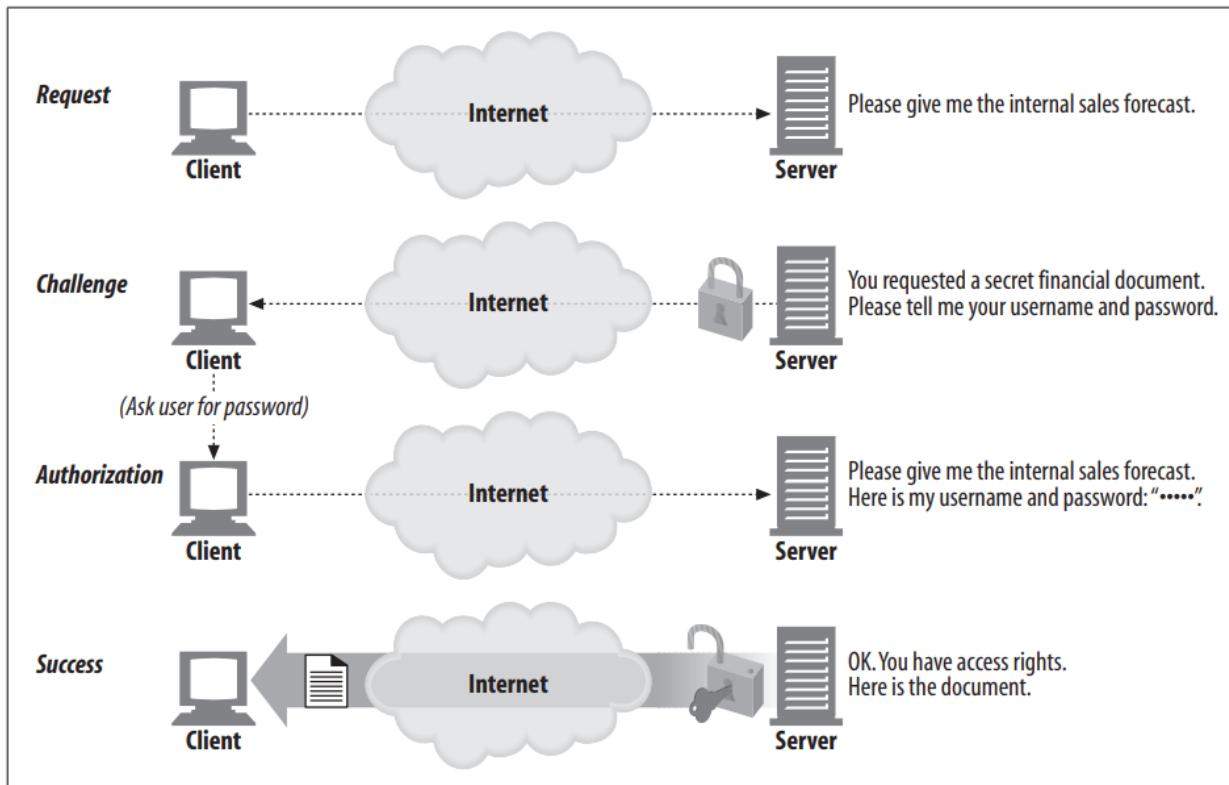
احراز هویت به معنای نشان دادن مدارکی از هویت شمامست. وقتی یک کارت شناسایی عکس دار، مانند گذرنامه یا گواهینامه رانندگی، نشان می‌دهید، شما در حال ارائه مدرکی هستید که نشان می‌دهد شما همان کسی هستید که ادعا می‌کنید. هنگامی که یک شماره پین را در دستگاه باجه خودکار تایپ می‌کنید، یا یک رمز عبور مخفی را در کادر محاوره ای رایانه تایپ می‌کنید، همچنین ثابت می‌کنید که همان چیزی هستید که ادعا می‌کنید.

در حال حاضر، هیچ یک از این طرح‌ها بدون خطای نیست. رمزهای عبور را می‌توان حدس زد یا شنید و کارت‌های شناسایی را می‌توان دزدید یا جعل کرد. اما هر یک از شواهد پشتیبان به ایجاد یک اعتماد معقول کمک می‌کند که شما همان چیزی هستید که می‌گویید.



HTTP's Challenge/Response Authentication Framework

HTTP یک چارچوب challenge/response بومی را فراهم می‌کند تا احراز هویت کاربران را آسان کند. مدل احراز هویت HTTP در شکل زیر ترسیم شده است.



هر زمان که یک برنامه وب یک پیام درخواست HTTP دریافت می‌کند، سرور به جای اقدام بر روی درخواست، می‌تواند با یک "Authentication Challenge" پاسخ دهد و کاربر را به چالش بکشد تا با ارائه برخی اطلاعات مخفی نشان دهد که او کیست.

زمانی که کاربر درخواست را تکرار می‌کند، باید Credential مخفی (نام کاربری و رمز عبور) را ضمیمه کند. اگر Credential ها مطابقت نداشته باشند، سرور می‌تواند دوباره کاربر را به چالش بکشد یا خطا ایجاد کند. اگر Credential ها مطابقت داشته باشند، درخواست به طور معمول تکمیل می‌شود.

Authentication Protocols and Headers

HTTP یک چارچوب قابل توسعه برای پروتکلهای احراز هویت مختلف، از طریق مجموعه‌ای از هدرهای کنترلی قابل تنظیم، را فراهم می‌کند. قالب و محتوای هدرهای فهرست شده در جدول زیر بسته به





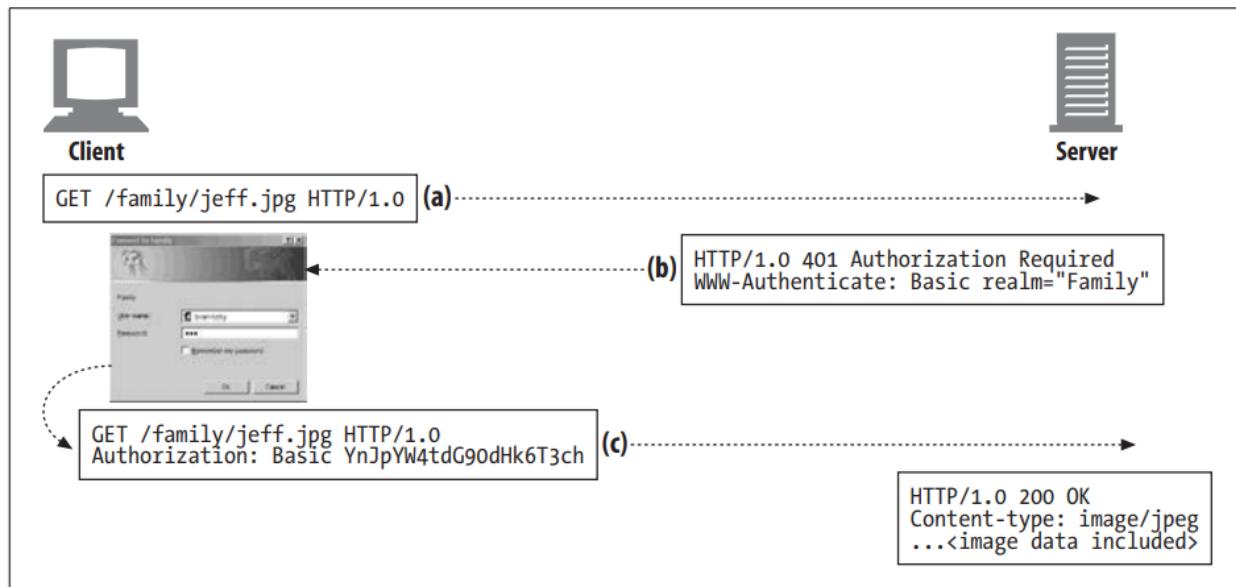
پروتکل احراز هویت متفاوت است. پروتکل احراز هویت نیز در هدرهای احراز هویت HTTP مشخص شده است.

Phase	Headers	Description	Method/Status
Request		The first request has no authentication.	GET
Challenge	WWW-Authenticate	The server rejects the request with a 401 status, indicating that the user needs to provide his username and password. Because the server might have different areas, each with its own password, the server describes the protection area in the WWW-Authenticate header. Also, the authentication algorithm is specified in the WWW-Authenticate header.	401 Unauthorized
Authorization	Authorization	The client retries the request, but this time attaching an Authorization header specifying the authentication algorithm, username, and password.	GET
Success	Authentication-Info	If the authorization credentials are correct, the server returns the document. Some authorization algorithms return some additional information about the authorization session in the optional Authentication-Info header.	200 OK

Digest و Basic Authentication دو پروتکل رسمی احراز هویت را تعریف می‌کند: Authentication در آینده، افراد می‌توانند پروتکلهای جدیدی را طراحی کنند که از چارچوب Basic Authentication استفاده می‌کنند. بقیه این فصل challenge/response HTTP برای جزئیات بیشتر در مورد Digest Authentication به فصل ۱۳ مراجعه کنید.

در ادامه اجازه دهید نگاهی به شکل زیر بیندازیم.





هنگامی که یک سرور یک کاربر را به چالش می‌کشد، یک پاسخ 401 غیر مجاز برمی‌گرداند و نحوه و مکان احراز هویت را در هدر **WWW-Authenticate** توضیح می‌دهد (بخش b شکل).

هنگامی که یک کلاینت به سرور اجازه ادامه کار را می‌دهد، درخواست را مجدداً ارسال می‌کند اما یک رمز عبور رمزگذاری شده و سایر پارامترهای احراز هویت را در یک هدر **Authorization** پیوست می‌کند (بخش c شکل).

هنگامی که یک درخواست مجاز با موفقیت تکمیل شد، سرور یک کد وضعیت **Normal** را برمی‌گرداند (به عنوان **OK** 200) و برای الگوریتم‌های احراز هویت پیشفرته، ممکن است اطلاعات اضافی را در یک هدر **Authentication-Info** ضمیمه کند (بخش d شکل).

Security Realms

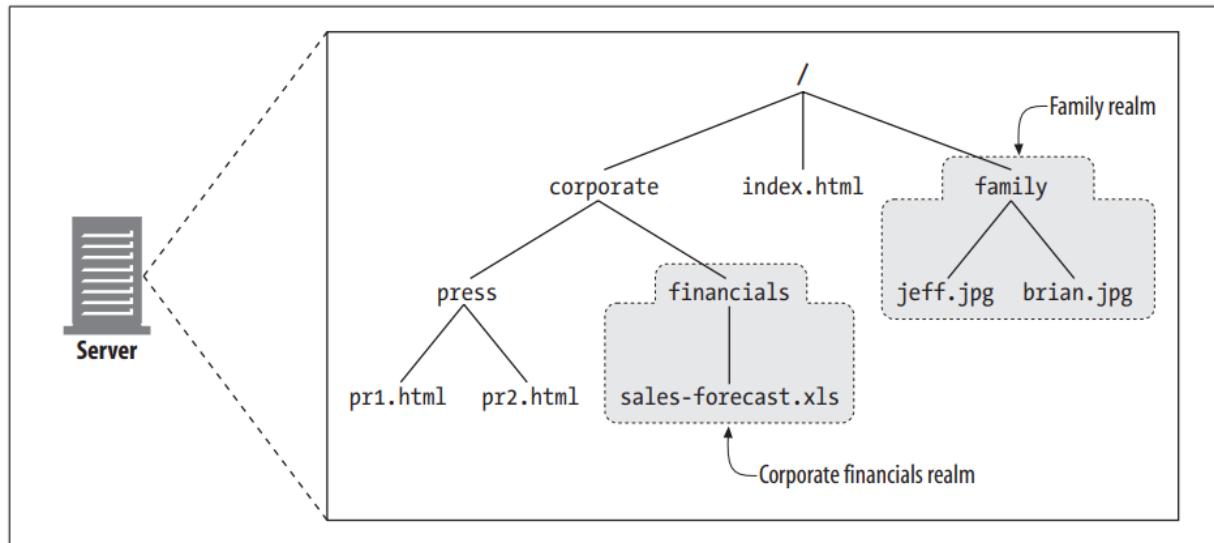
قبل از اینکه درباره جزئیات **Basic Authentication** صحبت کنیم، باید توضیح دهیم که چگونه HTTP به سرورها اجازه می‌دهد تا حقوق دسترسی مختلف را به منابع مختلف مرتبط کنند. شاید متوجه شده باشید که چالش **WWW-Authenticate** در بخش b شکل بالا شامل یک دستورالعمل **realm** است. وب سرورها اسناد **Security Realm** محافظت شده را در حوزه‌های امنیتی (**Security Realms**) گروه بندی می‌کنند. هر **Security Realm** می‌تواند مجموعه‌های مختلفی از کاربران مجاز داشته باشد.

به عنوان مثال، فرض کنید یک وب سرور دارای دو **Security Realm** است: یکی برای اطلاعات مالی شرکت و دیگری برای اسناد خانوادگی شخصی (شکل زیر را ببینید). کاربران مختلف دسترسی متفاوتی به





Realms ها خواهند داشت. احتمالاً مدیر عامل شرکت شما باید به پیش بینی فروش دسترسی داشته باشد، ولی ممکن است شما نخواهید به او اجازه دسترسی به عکس های تعطیلات خانوادگی خود را بدهید!



در اینجا یک چالش Basic Authentication فرضی، با realm مشخص شده است:

HTTP/1.0 401 Unauthorized

WWW-Authenticate: Basic realm="Corporate Financials"

یک realm باید دارای یک نام رشته توصیفی باشد، مانند "Corporate Financials"، تا به کاربر در درک نام کاربری و رمز عبور کمک کند. همچنین فهرست کردن نام میزبان سرور در نام realm ممکن است مفید باشد - برای مثال، "executive-committee@bigcompany.com"

Basic Authentication

Basic Authentication رایج ترین پروتکل احراز هویت HTTP است. تقریباً هر کلاینت و سرور اصلی HTTP/1.0 Basic Authentication را پیاده سازی می‌کند. در ابتدا در مشخصات Authentication توضیح داده شد، اما از آن زمان به RFC 2617 منتقل شده است، که احراز هویت HTTP را شرح می‌دهد.

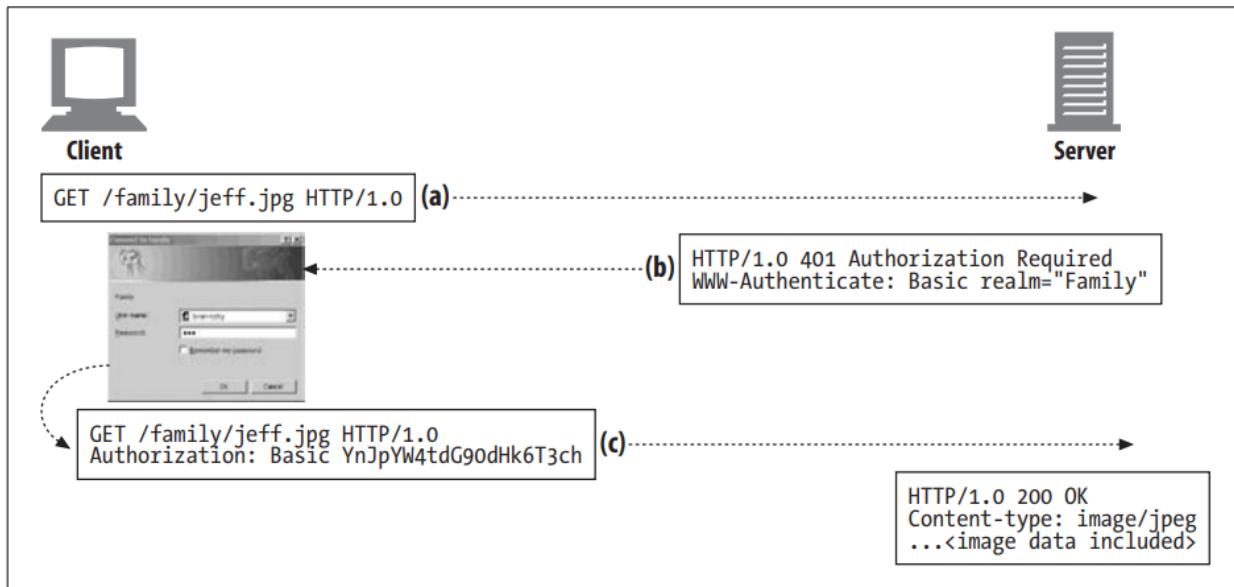
در Basic Authentication، یک وب سرور می‌تواند تراکنش را رد کند و کلاینت را برای یک نام کاربری و رمز عبور معتبر به چالش بکشد. سرور چالش احراز هویت را با بازگرداندن کد وضعیت 401 به جای 200 آغاز می‌کند و WWW-Authenticate قابل دسترسی با هدر پاسخ Security Realm را مشخص می‌کند. هنگامی که مرورگر چالش را دریافت می‌کند، قادر محاوره‌ای را باز می‌کند که نام کاربری و رمز عبور را برای این حوزه درخواست می‌کند. نام کاربری و رمز عبور با فرمت کمی درهم در داخل هدر درخواست مجوز به سرور ارسال می‌شود.





Basic Authentication Example

شکل زیر، که پیشتر نیز به آن اشاره شد، نمونه‌ای دقیق از Basic Authentication را نشان می‌دهد:



در بخش a از شکل، یک کاربر عکس خانوادگی شخصی /family/jeff.jpg را درخواست می‌کند.

در بخش b شکل، سرور 401 Authorization Required که مربوط به یک چالش رمز عبور است را برای عکس خانوادگی شخصی به همراه هدر WWW-Authenticate ارسال می‌کند. هدر برای realm به نام Family درخواست Basic Authentication می‌کند.

در بخش c شکل، مرورگر چالش 401 را دریافت می‌کند و قادر محاوره‌ای را باز می‌کند که نام کاربری و رمز عبور را برای realm مربوط به Family درخواست می‌کند. هنگامی که کاربر نام کاربری و رمز عبور را وارد می‌کند، مرورگر آنها را با یک دونقطه به آنها متصل می‌کند، آنها را در به صورت Base-64 "درهم" (که در بخش بعدی بحث می‌شود) کدگذاری نموده و آنها را در سربرگ Authorization ارسال می‌کند.

در بخش d شکل، سرور نام کاربری و رمز عبور را Decode نموده، صحت آنها را تأیید می‌کند و سند درخواستی را در یک پیام HTTP 200 OK برمی‌گرداند.

هدرهای احراز هویت Authorization و HTTP WWW-Authenticate در جدول زیر خلاصه شده است.



Challenge/Response	Header syntax and description
Challenge (server to client)	There may be different passwords for different parts of the site. The realm is a quoted string that names the set of documents being requested, so the user knows which password to use. WWW-Authenticate: Basic realm= <i>quoted-realm</i>
Response (client to server)	The username and password are joined together by a colon (:) and then converted to base-64 encoding, making it a bit easier to include international characters in usernames and passwords and making it less likely that a cursory examination will yield usernames and passwords while watching network traffic. Authorization: Basic <i>base64-username-and-password</i>

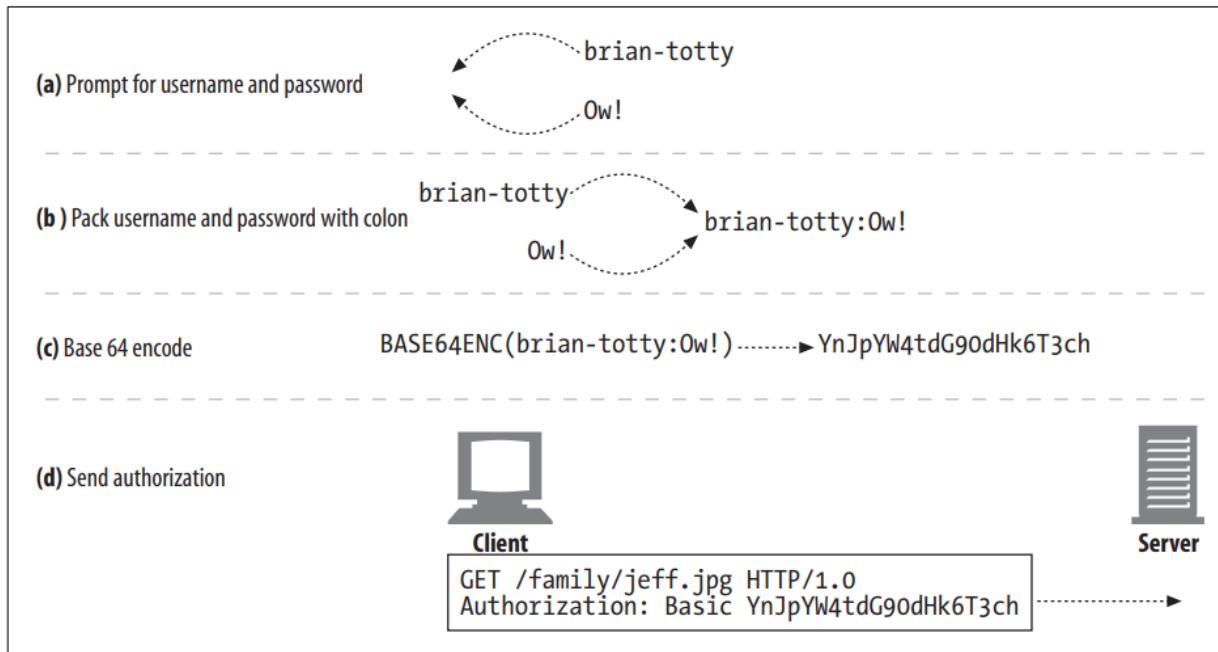
توجه داشته باشید که پروتکل AuthenticationInfo از هدر Basic Authentication که پیش تر اشاره شد، استفاده نمی کند.

Base-64 Username/Password Encoding

در HTTP، نام کاربری و رمز عبور را با هم بسته بندی می کند (با یک دونقطه که از هم جدا شده اند) و آنها را با استفاده از روش کدگذاری Base-64 کدگذاری می کند. اگر نمی دانید کدگذاری Base-64 چیست، نگران نباشید. نیازی نیست چیز زیادی در مورد آن بدانید. به طور خلاصه، کدگذاری-Base-64 دنباله‌ای از بایت‌های ۸ بیتی را می‌گیرد و دنباله بیت‌ها را به تکه‌های ۶ بیتی تقسیم می‌کند. هر قطعه ۶ بیتی برای انتخاب یک کاراکتر در یک 64-character خاص استفاده می‌شود که بیشتر از حروف و اعداد تشکیل شده است.

شكل زیر نمونه‌ای از استفاده از کدگذاری Base-64 برای Basic Authentication را نشان می‌دهد. در اینجا، نام کاربری "Ow!" و رمز عبور "brian-totty" است. مرورگر نام کاربری و رمز عبور را با یک دونقطه می‌پیوندد و رشته پر شده «brian-totty:Ow!» را ایجاد می‌کند. سپس این رشته با کدگذاری Base-64 بدین شکل می‌شود: "YnJpYW4tdG90dHk6T3ch"





رمزگذاری Base-64 برای دریافت رشته‌هایی از داده‌های باینری، متن و کاراکترهای بین‌المللی (که باعث ایجاد مشکلاتی در برخی سیستم‌ها می‌شد) اختراع شد و آن‌ها را به طور موقت به یک الفبای قابل حمل برای انتقال تبدیل کرد. بدین صورت رشته‌های اصلی می‌توانستند در سیستم راه دور بدون ترس از خرابی انتقال، شوند.

کدگذاری Base-64 می‌تواند برای نام‌های کاربری و رمزهای عبور حاوی کاراکترهای بین‌المللی یا سایر کاراکترهایی که در هدرهای HTTP غیرقانونی هستند (مانند علامت‌های نقل قول، دونقطه، و carriage return) مفید باشد. همچنین، از آنجایی که کدگذاری Base-64 به طور پیش‌پا افتاده نام کاربری و رمز عبور را به هم می‌زند، می‌تواند به جلوگیری از مشاهده تصادفی نام‌های کاربری و رمزهای عبور مدیران هنگام مدیریت سرورها و شبکه‌ها کمک کند.

Proxy Authentication

احراز هویت همچنین می‌تواند توسط سرورهای پروکسی واسطه انجام شود. برخی از سازمان‌ها از سرورهای پراکسی برای احراز هویت کاربران قبل از اینکه به آن‌ها اجازه دسترسی به سرورها، شبکه‌های محلی یا شبکه‌های بی‌سیم را بدهند، استفاده می‌کنند. سرورهای پراکسی می‌توانند راهی مناسب برای ارائه کنترل دسترسی یکپارچه در منابع سازمان باشند، زیرا خطمشی‌های دسترسی را می‌توان به صورت مرکزی بر روی سرور پراکسی مدیریت کرد. اولین گام در این فرآیند، ایجاد هویت از طریق احراز هویت پروکسی است.





مراحل مربوط به احراز هویت پروکسی با مراحل شناسایی وب سرور یکسان است. با این حال، هدرها و کدهای وضعیت متفاوت هستند. جدول زیر کدهای وضعیت و هدرهای مورد استفاده در وب سرور و احراز هویت پروکسی را در تضاد قرار می‌دهد.

Web server	Proxy server
Unauthorized status code: 401	Unauthorized status code: 407
WWW-Authenticate	Proxy-Authenticate
Authorization	Proxy-Authorization
Authentication-Info	Proxy-Authentication-Info

The Security Flaws of Basic Authentication

Basic Authentication ساده و راحت است، اما ایمن نیست. این فقط باید برای جلوگیری از دسترسی غیرعمدی از طرفهای غیر مخرب استفاده شود با در ترکیب با یک فناوری رمزگذاری مانند SSL استفاده شود.

نقایص امنیتی زیر را در نظر بگیرید:

- Basic Authentication نام کاربری و رمز عبور را به شکلی در سراسر شبکه ارسال می‌کند که می‌توان آن را Decode کرد. در واقع، Secret Password به صورت واضح ارسال می‌شود تا هر کسی بتواند آن را بخواند و بگیرد. کدگذاری Base-64 نام کاربری و رمز عبور را مبهم می‌کند و احتمال اینکه مهمان‌های دوستانه با مشاهده تصادفی شبکه رمزهای عبور را جمع‌آوری کنند، کمتر می‌شود. با این حال، با توجه به نام کاربری و رمز عبور کدگذاری شده با Base-64، Decode را می‌توان با معکوس کردن فرآیند کدگذاری انجام داد. حتی در چند ثانیه، با دست، با مداد و کاغذ قابل انجام است! گذروازه‌های کدگذاری شده با Base-64 به طور موثر در حالت شفاف ارسال می‌شوند. فرض کنید که اشخاص ثالث با انگیزه مخرب، نام کاربری و رمز عبور ارسال شده توسط Basic Authentication را رهگیری می‌کنند. اگر این یک نگرانی است، تمام تراکنش‌های HTTP خود را از طریق کانال‌های رمزگذاری شده SSL ارسال کنید یا از پروتکل احراز هویت امن‌تر مانند Digest Authentication استفاده کنید.

- حتی اگر Secret Password در طرحی کدگذاری شده باشد که Decode نمودن آن پیچیده‌تر باشد، شخص ثالث همچنان می‌تواند نام کاربری و رمز عبور مخدوش را ضبط کند و اطلاعات مخدوش را بارها و بارها برای دسترسی به سرورهای اصلی پخش کند. هیچ تلاشی برای جلوگیری از این حملات تکراری انجام نمی‌شود.





• حتی اگر Basic Authentication برای برنامه‌های غیر بحرانی مانند کنترل دسترسی اینترنت شرکتی یا محتوای شخصی شده استفاده شود، رفتار اجتماعی این امر را خطرناک می‌کند. بسیاری از کاربران، غرق در سرویس‌های محافظت شده با رمز عبور، نامهای کاربری و رمز عبور را به اشتراک می‌گذارند. یک شخص باهوش و مخرب ممکن است نام کاربری و رمز عبور را به طور واضح از یک سایت ایمیل اینترنتی رایگان دریافت کند و متوجه شود که همان نام کاربری و رمز عبور اجازه دسترسی به سایت‌های مهم بانکداری آنلاین را می‌دهد!

• احراز هویت اولیه هیچ محافظتی در برابر پروکسی‌ها یا واسطه‌هایی که به عنوان واسطه عمل می‌کنند، ارائه نمی‌دهد، هدرهای احراز هویت دست نخورده باقی می‌ماند، اما بقیه پیام را تغییر می‌دهد تا ماهیت تراکنش را به شدت تغییر دهد.

• Basic Authentication در برابر جعل توسط سرورهای تقلیبی آسیب‌پذیر است. اگر بتوان کاربر را به این باور رساند که در حال اتصال به یک میزبان معتبر محافظت شده توسط Basic Authentication است، در واقع زمانی که در حال اتصال به یک سرور یا دروازه متخاصم است، مهاجم می‌تواند رمز عبور درخواست کند، آن را برای استفاده بعدی ذخیره کند، و تظاهر به یک خطا کند.

با همه این‌ها، Basic Authentication هنوز برای ارائه شخصی‌سازی راحت یا کنترل دسترسی به اسناد در یک محیط دوستانه، یا در مواردی که حریم خصوصی مورد نظر است اما کاملاً ضروری نیست، مفید است. به این ترتیب، Basic Authentication برای جلوگیری از دسترسی تصادفی توسط کاربران کنجدکاو استفاده می‌شود.

به عنوان مثال، در داخل یک شرکت، مدیریت محصول ممکن است با رمز عبور از برنامه‌های محصول آینده محافظت کند تا توزیع زودرس را محدود کند. به همین ترتیب، ممکن است از عکس‌های شخصی یا طرف‌های دوستانه، به اندازه کافی ناخوشایند می‌کند. به همین ترتیب، ممکن است از وبسایت‌های خصوصی که کاملاً محترمانه نیستند یا حاوی اطلاعات ارزشمندی نیستند، با رمز عبور محافظت کنید، اما واقعاً به کسب و کار شخص دیگری هم مربوط نمی‌شود.

Basic Authentication را می‌توان با ترکیب آن با انتقال داده‌های رمزگذاری شده (مانند SSL) ایمن کرد تا نام کاربری و رمز عبور از افراد مخرب پنهان شود. این یک تکنیک رایج است.

ما در مورد رمزگذاری ایمن در فصل ۱۴ بحث می‌کنیم. فصل بعدی یک پروتکل احراز هویت HTTP پیچیده‌تر، Digest Authentication را توضیح می‌دهد که دارای ویژگی‌های امنیتی قوی‌تری نسبت به Basic Authentication است.





For More Information

<http://www.ietf.org/rfc/rfc2617.txt>

<http://www.ietf.org/rfc/rfc2616.txt>





فصل سیزدهم – Digest Authentication

راحت و منعطف است اما کاملاً نامن است. نامهای کاربری و گذرواژه‌ها به صورت واضح ارسال می‌شوند و هیچ تلاشی برای محافظت از پیام‌ها در برابر دستکاری وجود ندارد. تنها راه برای استفاده ایمن از Basic Authentication استفاده از آن در ارتباط با SSL است.

به عنوان جایگزینی سازگار و امن تر برای احراز هویت اولیه توسعه داده شد. ما این فصل را به Digest Authentication اختصاص می‌دهیم. حتی اگر هنوز به طور گسترده مورد استفاده قرار نگرفته است، مفاهیم هنوز برای هر کسی که تراکنش‌های ایمن را اجرا می‌کند مهم است.

The Improvements of Digest Authentication

یک پروتکل احراز هویت HTTP جایگزین است که سعی می‌کند جدی‌ترین نقص‌های Basic Authentication را برطرف کند. به طور خاص،

- هرگز رمزهای عبور مخفی را در سراسر شبکه به صورت شفاف ارسال نمی‌کند.
- از گرفتن و پخش مجدد Authentication Handshake توسط مهاجمان جلوگیری می‌کند.
- به صورت اختیاری می‌تواند از دستکاری در محتوای پیام محافظت کند.
- در برابر چندین اشکال رایج حملات دیگر نیز موارد محافظتی را اعمال می‌کند.

توجه داشته باشید که Digest Authentication ایمن‌ترین پروتکل ممکن نیست. بسیاری از نیازها برای تراکنش‌های HTTP ایمن با Digest Authentication برآورده نمی‌شوند. برای این نیازها، امنیت لایه حمل و نقل (TLS) و HTTP امن (HTTPS) پروتکلهای مناسب‌تری هستند.

با این حال، Digest Authentication به طور قابل توجهی قوی‌تر از Basic Authentication است، که برای جایگزینی طراحی شده است. Digest Authentication نیز قوی‌تر از بسیاری از طرح‌های محبوب ارائه شده برای سایر سرویس‌های اینترنتی است، مانند CRAM-MD5، که برای استفاده با POP، LDAP و IMAP پیشنهاد شده است.

تا به امروز، Digest Authentication به طور گسترده اجرا نشده است. با این حال، به دلیل خطرات امنیتی ذاتی HTTP، معماران RFC 2617 توصیه می‌کنند که «هر سرویسی که در حال حاضر از Basic استفاده می‌کند باید به محض عملی شدن به Digest تبدیل شود.»

هنوز مشخص نیست که این استاندارد چقدر موفق بوده است.





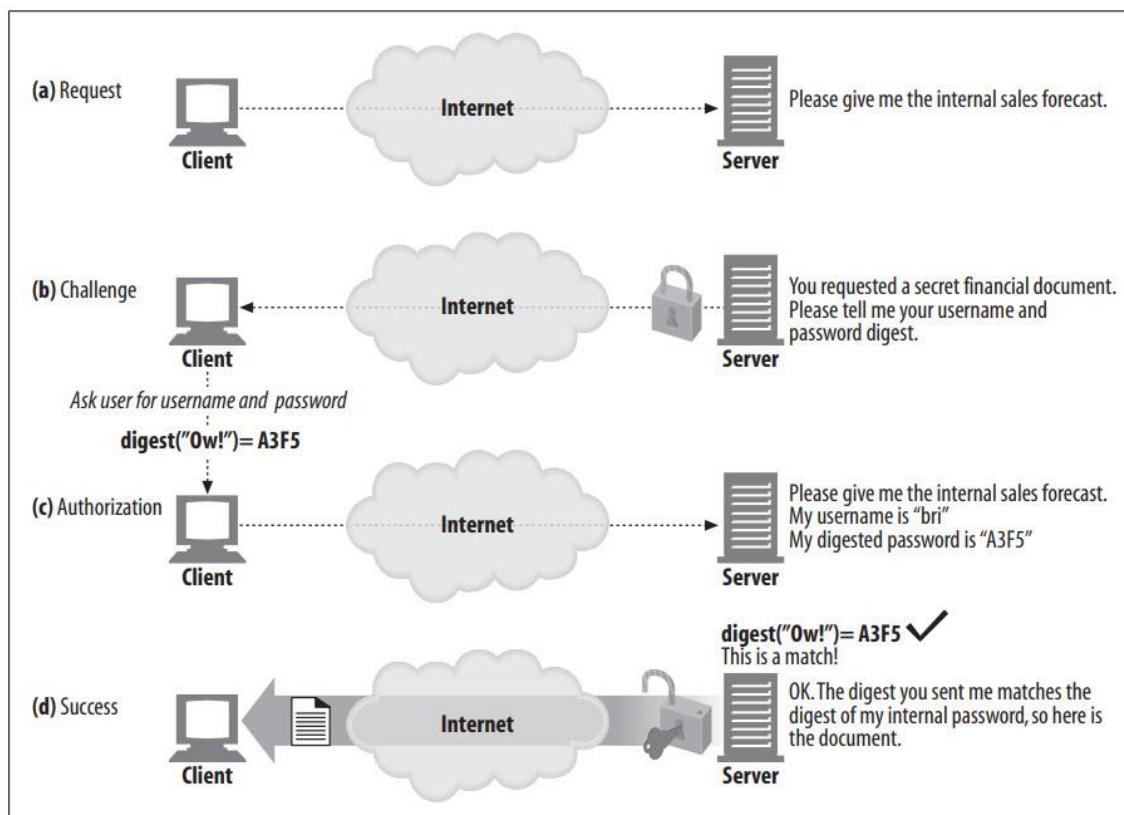
Using Digests to Keep Passwords Secret

شعار "Digest Authentication" هرگز رمز عبور را در سراسر شبکه ارسال نکنید.

به جای ارسال رمز عبور، کلاینت یک "Fingerprint" یا "Digest" رمز عبور را ارسال می‌کند که درهم شکسته غیرقابل برگشت (Irreversible Scrambling) رمز عبور است. کلاینت و سرور هر دو رمز عبور مخفی را می‌دانند، بنابراین سرور می‌تواند تأیید کند که Digest تطابق درستی برای رمز عبور ارائه کرده است. با توجه به این، یک پسر بد راه آسانی برای یافتن رمز عبور آن ندارد، به جز اینکه همه رمزهای عبور در جهان را مورخ کند و هر کدام را امتحان کند!

بیایید ببینیم این مورد چگونه کار می‌کند (این یک نسخه ساده شده است):

در شکل بخش a زیر، کلاینت یک سند محافظت شده را درخواست می‌کند.





در بخش b شکل، سرور از ارائه سند خودداری می‌کند تا زمانی که کلاینت هویت آن را با اثبات اینکه رمز عبور را می‌داند، احراز هویت کند. سرور چالشی را برای کلاینت ارسال می‌کند و نام کاربری و فرم Digest شده رمز عبور را می‌خواهد.

در بخش c شکل، کلاینت با ارسال Digest رمز عبور ثابت می‌کند که رمز عبور را می‌داند. سرور گذرواژه‌های همه کاربران را می‌داند، بنابراین می‌تواند با مقایسه Digest ارائه شده توسط کلاینت با Digest محاسبه شده داخلی سرور، تأیید کند که کاربر رمز عبور را می‌داند. طرف دیگر اگر رمز عبور را نمی‌دانست به راحتی نمی‌توانست Digest درستی را بسازد.

در بخش d شکل، سرور Digest ارائه شده توسط کلاینت را با Digest محاسبه شده داخلی سرور مقایسه می‌کند. اگر مطابقت داشته باشند، نشان می‌دهد که کلاینت رمز عبور را می‌داند (یا حدس واقعاً خوش شانسی انجام داده است!). تابع Digest را می‌توان طوری تنظیم کرد که ارقام زیادی تولید کند که حدس زدن تصادفی به طور مؤثر غیرممکن باشد. هنگامی که سرور مطابقت را تأیید می‌کند، سند به کلاینت ارائه می‌شود - همه این‌ها بدون ارسال رمز عبور از طریق شبکه.

One-Way Digests

«تراکم مجموعه‌ای از اطلاعات» است. Digest ها به عنوان توابع یک طرفه عمل می‌کنند و عموماً تعداد نامحدودی از مقادیر ورودی ممکن را به محدوده محدودی از تراکم تبدیل می‌کنند. دنباله‌ای از بایت‌ها، با هر طول، به یک خلاصه ۱۲۸ بیتی.

آنچه در مورد این Digest ها مهم است این است که اگر رمز عبور مخفی را ندانید، حدس زدن درست برای ارسال به سرور برای شما بسیار سخت است. و به همین ترتیب، اگر Digest را داشته باشید، تشخیص اینکه کدام یک از بی‌نهایت مقادیر ورودی آن را ایجاد کرده‌اند، بسیار سخت خواهد بود.

۱۲۸ بیت خروجی MD5 اغلب به صورت ۳۲ کاراکتر هگزادسیمال نوشته می‌شود که هر کاراکتر نشان دهنده ۴ بیت است. جدول زیر چند نمونه از Digest های MD5 را نشان می‌دهد. توجه داشته باشید که چگونه MD5 ورودی‌های دلخواه را دریافت می‌کند و یک خروجی Digest با طول ثابت ایجاد می‌کند.





Input	MD5 digest
"Hi"	C1A5298F939E87E8F962A5EDFC206918
"bri:Ow!"	BEAAA0E34EBDB072F8627C038AB211F8
"3.1415926535897"	475B977E19ECEE70835BC6DF46F4F6DE
"http://www.http-guide.com/index.htm"	C617C0C7D1D05F66F595E22A4B0EAAA5
"WE hold these Truths to be self-evident, that all Men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the Pursuit of Happiness—That to secure these Rights, Governments are instituted among Men, deriving their just Powers from the Consent of the Governed, that whenever any Form of Government becomes destructive of these Ends, it is the Right of the People to alter or to abolish it, and to institute new Government, laying its Foundation on such Principles, and organizing its Powers in such Form, as to them shall seem most likely to effect their Safety and Happiness."	66C4EF58DA7CB956BD04233FBB64EOA4

تابع گاهی اوقات Checksum های رمزگاری، توابع هش یک طرفه یا توابع Fingerprint نامیده می‌شوند.

Using Nonces to Prevent Replays

یک طرفه ما را از ارسال رمزهای عبور به صورت شفاف نجات می‌دهد. ما فقط می‌توانیم Digest رمز عبور را به جای آن ارسال کنیم و مطمئن باشید که هیچ طرف مخربی نمی‌تواند به راحتی رمز عبور اصلی را از Digest رمزگشایی کند.

متأسفانه، رمزهای عبور ممکن است تا از خطر نجات نمی‌دهد، زیرا یک نفوذگر می‌تواند Digest را ضبط کرده و بارها و بارها آن را برای سرور پخش کند، حتی اگر رمز عبور را نداند. Digest به اندازه رمز عبور خوب است.

برای جلوگیری از چنین حملات تکراری، سرور می‌تواند توکن خاصی به نام *nonce را برای کلاینت ارسال کند که اغلب (شاید در هر میلی ثانیه یا برای هر احراز هویت) تغییر می‌کند. کلاینت قبل از محاسبه Digest توکن nonce را به رمز عبور اضافه می‌کند.

مخلوط کردن nonce با رمز عبور باعث می‌شود که هر بار که Digest تغییر می‌کند، این از حملات تکراری جلوگیری می‌کند، زیرا Digest رمز ثبت شده فقط برای یک مقدار nonce خاص معتبر است و بدون رمز عبور مخفی، مهاجم نمی‌تواند Digest درست را محاسبه کند.



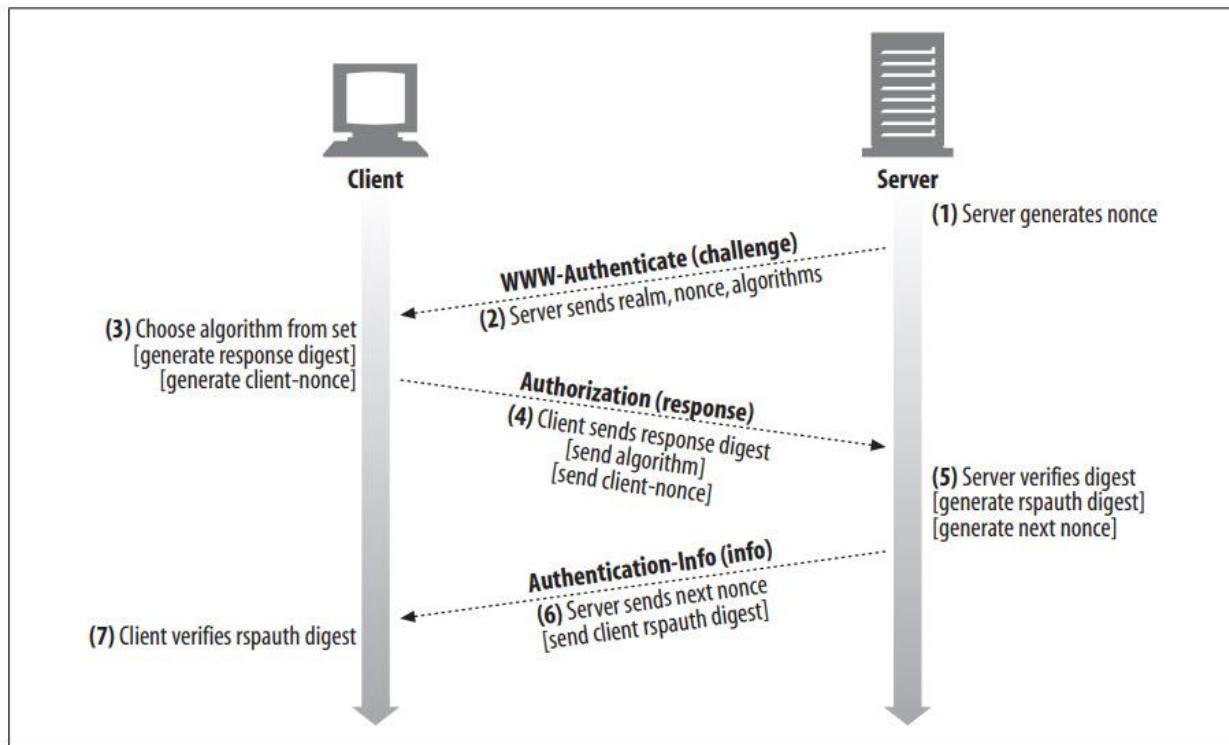


نیاز به استفاده از nonces دارد، زیرا یک ضعف در پخش بی اهمیت باعث می شود Nonces بهاندازه Basic Authentication Un-nonced Digest Authentication چالش WWW از سروری به کلاینت دیگر منتقل می شود.

The Digest Authentication Handshake

پروتکل Digest Authentication یک نسخه پیشرفته از احراز هویت است که از هدرهای مشابه آنچه در استفاده می شود استفاده می کند. برخی از گزینه های جدید به هدرهای سنتی اضافه می شوند و یک هدر اختیاری جدید، Authorization-Info، اضافه می شود.

دست دادن سه مرحله ای ساده شده Digest Authentication در شکل زیر نشان داده شده است.



آنچه در شکل بالا اتفاق می افتد در اینجا آمده است:

- در مرحله ۱، سرور یک مقدار nonce را محاسبه می کند. در مرحله ۲، سرور nonce را در یک پیام چالشی WWW-Authenticate به همراه فهرستی از الگوریتم هایی که سرور پشتیبانی می کند، برای کلاینت ارسال می کند.





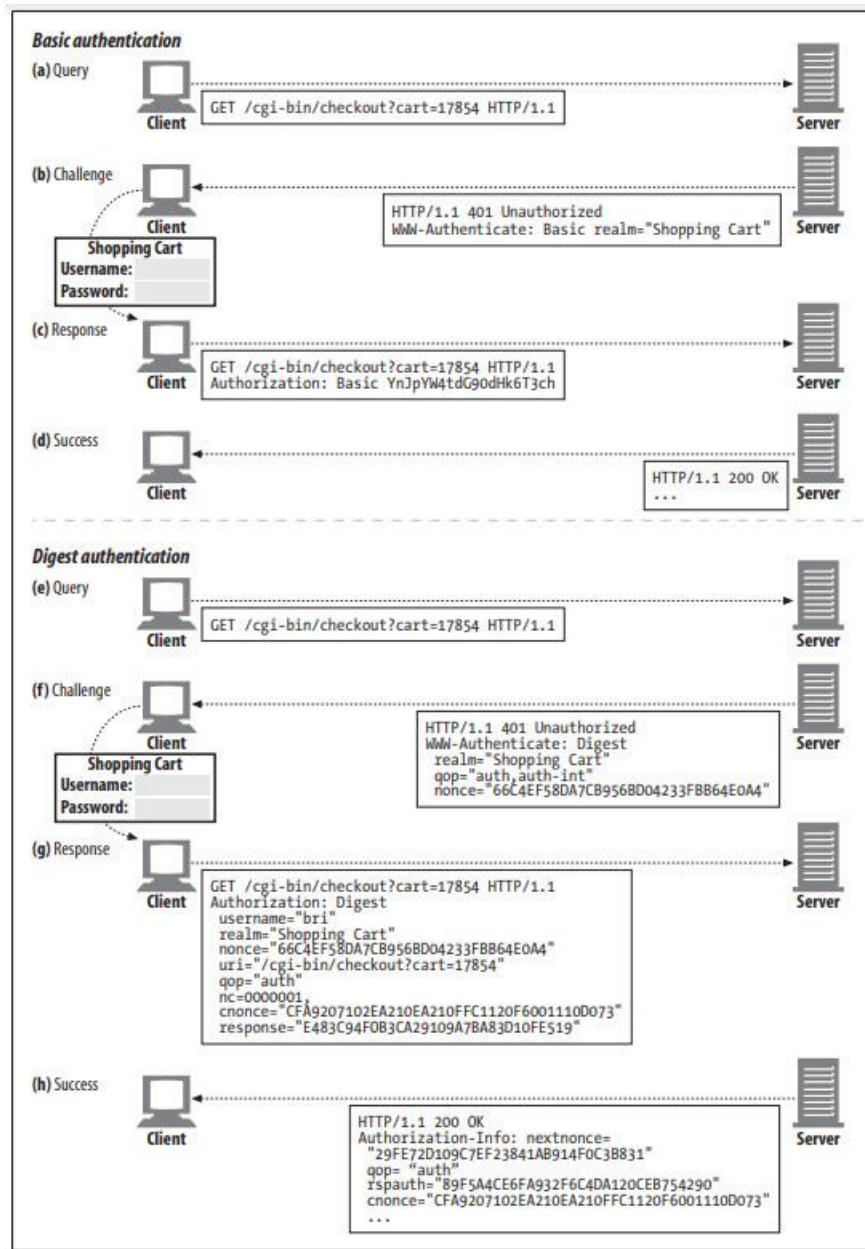
در مرحله ۳، کلاینت الگوریتمی را انتخاب می‌کند و Digest رمز عبور مخفی و سایر داده‌ها را محاسبه می‌کند. در مرحله ۴، Digest را در یک پیام مجوز به سرور ارسال می‌کند. اگر کلاینت بخواهد سرور را احراز هویت کند، می‌تواند یک Client Nonce ارسال کند.

در مرحله ۵، سرور Digest، الگوریتم انتخاب شده، و داده‌های پشتیبانی را دریافت می‌کند و همان ای را که کلاینت انجام می‌دهد محاسبه می‌کند. سپس سرور Digest تولید شده محلی را با Digest ارسال شده از طریق شبکه مقایسه می‌کند و مطابقت آن‌ها را تأیید می‌کند. اگر کلاینت به طور متقارن سرور را با یک کلاینت به چالش بکشد، Digest مشتری ایجاد می‌شود. علاوه بر این، nonce بعدی را می‌توان از قبل محاسبه کرد و از قبل به کلاینت تحویل داد، بنابراین کلاینت می‌تواند دفعه بعد مناسب Digest صادر کند.

بسیاری از این اطلاعات اختیاری هستند و دارای پیش فرض هستند. برای روشن شدن موارد، شکل ۱۳-۳- پیام‌های ارسال شده برای احراز هویت اولیه (a-d) را با یک مثال ساده از احراز هویت خلاصه (شکل e-h) مقایسه می‌کند.

اکنون بباید کمی دقیق‌تر به عملکرد داخلی احراز هویت Digest نگاه کنیم.





Digest Calculations

قلب Digest است که یک طرفه ترکیبی از اطلاعات عمومی، اطلاعات مخفی و یک مقدار time-limited nonce می‌باشد. بیایید اکنون به نحوه محاسبه Digest ها نگاه کنیم. محاسبات Digest به طور کلی ساده است.

Digest Algorithm Input Data

Digest ها به وسیله سه جزء مورد محاسبه قرار می‌گیرند:





- یک جفت توابع متشکل از یکتابع هش یک طرفه $H(d)$ و $KD(s,d)$ Digest، که در آن s مخفف d و $Secret$ داده است.
 - تکهای از داده‌ها حاوی اطلاعات امنیتی، از جمله رمز عبور مخفی، به نام A1
 - تکهای از داده‌ها حاوی ویژگی‌های غیرمخفی پیام درخواست به نام A2
- دو قطعه از داده‌ها، A1 و A2، توسط H و KD پردازش می‌شوند تا یک Digest را ارائه دهند.

The Algorithms $H(d)$ and $KD(s,d)$

احراز هویت Digest از انتخاب انواع الگوریتم‌های Digest پشتیبانی می‌کند. دو الگوریتم پیشنهاد شده در RFC 2617 MD5 و MD5-sess هستند (که در آن "sess" مخفف session است) و اگر الگوریتم دیگری مشخص نشده باشد، الگوریتم به طور پیش فرض روی MD5 قرار می‌گیرد.

اگر از MD5-sess یا MD5 digest استفاده شود، تابع H MD5 داده‌ها را محاسبه می‌کند، و تابع MD5 داده‌های مخفی و غیرمخفی متصل به کولون را محاسبه می‌کند. به عبارت دیگر:

$$H(<\text{data}>) = MD5(<\text{data}>)$$

$$KD(<\text{secret}>, <\text{data}>) = H(\text{concatenate}(<\text{secret}>: <\text{data}>))$$

The Security-Related Data (A1)

تکه‌ای از داده‌ها به نام A1 محصول اطلاعات محرمانه و حفاظتی مانند نام کاربری، رمز عبور، قلمرو حفاظتی (Protection Realm) و nonces است. A1 فقط به اطلاعات امنیتی مربوط می‌شود، نه به خود پیام A1 به همراه H، KD و A2 برای محاسبه Digest‌ها استفاده می‌شود.

RFC 2617 بسته به الگوریتم انتخابی دو روش برای محاسبه A1 تعریف می‌کند:

MD5

هش‌های یک طرفه برای هر درخواست اجرا می‌شوند. A1 سه گانه نام کاربری، Realm و رمز عبور مخفی است که با کولون متصل می‌شود.

MD5-sess

تابع هش فقط یک بار در اولین دست دادن WWW-Authenticate اجرا می‌شود. هش CPU-intensive نام کاربری، Realm و رمز عبور مخفی یک بار انجام می‌شود و به مقادیر nonce و Client nonce (cnonce) فعلی اضافه می‌شود.



تعاریف A1 در جدول زیر نشان داده شده است.

Algorithm	A1
MD5	$A1 = <user>:<realm>:<password>$
MD5-sess	$A1 = MD5(<user>:<realm>:<password>):<nonce>:<cnonce>$

The Message-Related Data (A2)

تکه‌ای از داده‌ها به نام A2 اطلاعات مربوط به خود پیام را نشان می‌دهد، مانند URL، متدهای درخواست و بدنی موجودیت پیام. A2 برای کمک به محافظت در برابر دستکاری روش، منبع یا پیام استفاده می‌شود. A2 به همراه KD، H و A1 برای محاسبه Digest ها استفاده می‌شود.

تعاریف A2 در RFC 2617 بسته به **qop** quality of protection یا انتخاب شده، دو طرح را برای A2 تعریف می‌کند:

- طرح اول فقط شامل روش درخواست HTTP و URL است. این بخش زمانی استفاده می‌شود که **qop="auth"** باشد که حالت پیش فرض است.
- طرح دوم بدنی موجودیت پیام را اضافه می‌کند تا درجه‌ای از بررسی یکپارچگی پیام را فراهم کند. زمانی که **qop="auth-int"** باشد استفاده می‌شود.

تعاریف A2 در جدول زیر نشان داده شده است.

qop	A2
undefined	$<request-method>:<uri-directive-value>$
auth	$<request-method>:<uri-directive-value>$
auth-int	$<request-method>:<uri-directive-value>:<H(<request-entity-body>)>$

روش درخواست request-method URI درخواستی از خط درخواست uri-directive-value است. این ممکن است «*»، «abs_path» یا «URL مطلق» باشد، اما باید با URI درخواست موافق باشد. به ویژه، اگر URI درخواست یک URL مطلق باشد، باید یک URL مطلق ارائه شود.

Overall Digest Algorithm

RFC 2617 با توجه به H، KD، A1 و A2، دو روش را برای محاسبه Digest ها تعریف می‌کند:

- راه اول برای سازگاری با مشخصات قدیمی RFC 2069 در نظر گرفته شده است که در صورت عدم وجود گزینه qop استفاده می‌شود. Digest را با استفاده از هش اطلاعات مخفی و داده‌های پیام nonced محاسبه می‌کند.





- راه دوم رویکرد مدرن و ترجیحی است - شامل پشتیبانی از شمارش nonce و احراز هویت متقارن است. این رویکرد زمانی استفاده می‌شود که "auth-int" یا "qop" "auth" باشد. داده‌های nonce count را به Digest اضافه می‌کند.

تعاریف تابع Digest حاصل در جدول زیر نشان داده شده است. به Digest های حاصل از H، A1، KD و A2 توجه کنید.

qop	Digest algorithm	Notes
undefined	KD(H(A1), <nonce>:H(A2))	Deprecated
auth or auth-int	KD(H(A1), <nonce>:<nc>:<cnonce>:<qop>:H(A2))	Preferred

گم شدن در تمام لایه‌های کپسوله‌سازی مشتق کمی آسان است. این یکی از دلایلی است که برخی از خوانندگان RFC 2617 مشکل دارند. جدول زیر برای آسان‌تر کردن آن، تعاریف H و KD را گسترش می‌دهد و هایی را بر حسب A1 و A2 به جا می‌گذارد.

qop	Algorithm	Unfolded algorithm
undefined	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:MD5(A2))

qop	Algorithm	Unfolded algorithm
auth	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:<nc>:<cnonce>:<qop>:MD5(A2))
auth-int	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:<nc>:<cnonce>:<qop>:MD5(A2))

Digest Authentication Session

پاسخ کلاینت به چالش WWW-Authenticate برای یک فضای حفاظتی، جلسه احراز هویت را با آن فضای حفاظتی شروع می‌کند (Realm) ترکیب شده با ریشه متعارف سروری که به آن دسترسی پیدا می‌شود، «فضای حفاظتی» را تعریف می‌کند).

جلسه احراز هویت تا زمانی که کلاینت چالش WWW-Authenticate دیگری را از هر سروری در فضای حفاظتی دریافت کند، ادامه دارد. یک کلاینت باید نام کاربری، رمز عبور، nonce، تعداد

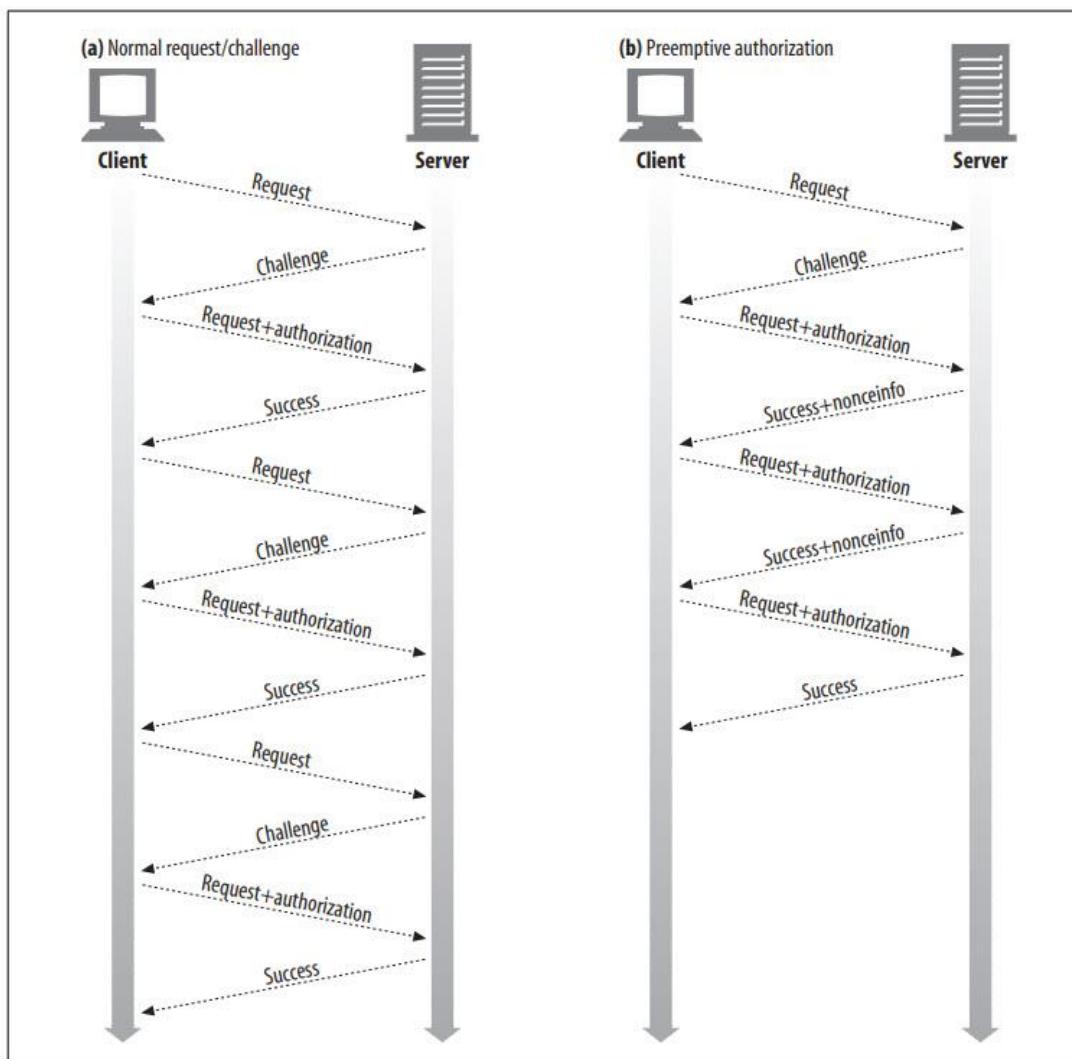




و مقادیر opaque مرتبط با جلسه احراز هویت را به خاطر بسپارد تا از آن برای ساخت هدر Authorization در درخواست‌های آینده در فضای حفاظتی استفاده کند.

وقتی nonce منقضی می‌شود، سرور می‌تواند اطلاعات هدر مجوز قدیمی را بپذیرد، حتی اگر مقدار nonce گنجانده شده تازه نباشد. از طرف دیگر، سرور ممکن است یک پاسخ ۴۰۱ را با یک مقدار nonce جدید برگرداند و باعث شود کلاینت درخواست را دوباره امتحان کند. با مشخص کردن "stale=true" با این پاسخ، سرور به مشتری می‌گوید که بدون درخواست نام کاربری و رمز عبور جدید، با nonce جدید دوباره امتحان کند.

Preemptive Authorization



در احراز هویت عادی، هر درخواست قبل از تکمیل تراکنش نیاز به یک چرخه request/challenge نیاز دارد. این در بخش a شکل بالا نشان داده شده است.





این چرخه request/challenge را می‌توان حذف کرد اگر کلاینت از قبل بداند `nonce` بعدی چیست، بنابراین می‌تواند هدر Authorization صحیح را قبل از درخواست سرور ایجاد کند. اگر کلاینت بتواند هدر Authorization را قبل از درخواست محاسبه کند، کلاینت می‌تواند به طور پیشگیرانه هدر Authorization را به سرور صادر کند، بدون اینکه ابتدا request/challenge را پشت سر بگذارد. تاثیر عملکرد در بخش b شکل بالا نشان داده شده است.

مجوز پیشگیرانه برای Basic Authentication بی‌اهمیت (و رایج) است. مرورگرها معمولاً پایگاه داده‌های client-side ای از نام‌های کاربری و رمز عبور را نگهداری می‌کنند. هنگامی که کاربر به یک سایت احراز هویت می‌کند، مرورگر معمولاً هدر Authorization صحیح را برای درخواست‌های بعدی به آن URL ارسال می‌کند (به فصل ۱۲ مراجعه کنید).

مجوز پیشگیرانه برای Digest Authentication کمی پیچیده‌تر است، زیرا فناوری `nonce` در نظر گرفته شده برای خنثی کردن Replay Attack است. از آنجایی که سرور، `nonces` دلخواه را تولید می‌کند، همیشه راهی برای کلاینت وجود ندارد که تعیین کند تا زمانی که یک چالش را دریافت کند، چه هدر Authorization را ارسال کند.

Digest Authentication چند ابزار برای مجوز پیشگیرانه ارائه می‌دهد در حالی که بسیاری از ویژگی‌های اینمی را حفظ می‌کند. در اینجا سه راه بالقوه وجود دارد که کلاینت می‌تواند بدون انتظار برای چالش جدید `Nonce`, `WWW-Authenticate` صحیح را به دست آورد:

- سرور، `nonce` بعدی را در هدر موقیت `Authentication-Info` از قبل ارسال می‌کند.
- سرور اجازه می‌دهد تا از همان `nonce` برای یک پنجره زمانی کوچک استفاده مجدد شود.
- هم کلاینت و هم سرور از یک الگوریتم `nonce-generation` همگام و قابل پیش‌بینی استفاده می‌کنند.

Next nonce pregeneration

مقدار `nonce` بعدی را می‌توان از قبل توسط هدر موقیت `AuthenticationInfo` در اختیار کلاینت قرار داد. این هدر همراه با پاسخ 200 OK از یک احراز هویت موفق قبلی ارسال می‌شود.

`Authentication-Info: nextnonce=<nonce-value>"`

با توجه به `nonce` بعدی، کلاینت می‌تواند به صورت پیشگیرانه یک هدر Authorization صادر کند.





در حالی که این مجوز پیشگیرانه از چرخه request/challenge جلوگیری می‌کند (سرعت تراکنش را افزایش می‌دهد)، همچنین به طور موثر توانایی Pipeline کردن چندین درخواست به یک سرور را باطل می‌کند، زیرا مقدار nonce بعدی باید قبیل از صدور درخواست بعدی دریافت شود. از آنجا که انتظار می‌رود یک Pipeline فناوری اساسی برای جلوگیری از تأخیر باشد، جریمه عملکرد ممکن است زیاد باشد.

Limited nonce reuse

به جای ایجاد پیش‌تولید دنباله‌ای از nonces، رویکرد دیگر اجازه استفاده مجدد محدود از nonces است. به عنوان مثال، یک سرور ممکن است اجازه دهد یک nonce ۵ بار یا برای ۱۰ ثانیه استفاده مجدد شود.

در این حالت، کلایнт می‌تواند آزادانه درخواست‌هایی را با هدر Authorization صادر کند و می‌تواند آنها را Pipeline کند، زیرا nonce از قبل شناخته شده است. زمانی که nonce در نهایت منقضی شود، انتظار می‌رود سرور یک چالش ۴۰۱ غیر مجاز را با مجموعه دستورات WWW-Authenticate: stale=true برای کلاینت ارسال کند:

WWW-Authenticate: Digest

realm=<realm-value>

nonce=<nonce-value>

stale=true

استفاده مجدد از nonces امنیت را کاهش می‌دهد، زیرا موفقیت مهاجم را در Replay Attack آسان‌تر می‌کند. از آنجایی که طول عمر استفاده مجدد nonce قابل کنترل است، از عدم استفاده مجدد تا استفاده مجدد بالقوه طولانی، می‌توان بین پنجره‌های آسیب پذیری و عملکرد، معاوضه‌هایی ایجاد کرد.

علاوه بر این، از ویژگی‌های دیگری نیز می‌توان برای دشوارتر کردن Replay Attack استفاده کرد، از جمله افزایش شمارنده و آزمایش آدرس IP. با این حال، در حالی که حملات را ناخوشایندتر می‌کنند، این تکنیک‌ها آسیب‌پذیری را از بین نمی‌برند.

Synchronized nonce generation

امکان استفاده از الگوریتم‌های nonce-generation همگام‌سازی شده با زمان وجود دارد، که در آن کلاینت و سرور می‌توانند دنباله‌ای از nonces یکسان را بر اساس یک کلید مخفی مشترک ایجاد کنند، که شخص ثالث به راحتی نمی‌تواند آنها را پیش‌بینی کند (مانند کارت‌های شناسایی امن).





این الگوریتم‌ها فراتر از محدوده مشخصات Digest Authentication هستند.

Nonce Selection

محتویات nonce مبهم (opaque) و وابسته به اجرا هستند. با این حال، کیفیت عملکرد، امنیت و راحتی به انتخاب هوشمندانه بستگی دارد.

RFC 2617 این فرمول فرضی nonce را پیشنهاد می‌کند:

`BASE64(time-stamp H(time-stamp ":" ETag ":" private-key))`

در جایی که time-stamp یک زمان تولید شده توسط سرور یا مقدار غیر تکراری دیگر است، ETag مقدار هدر HTTP ETag مرتبط با موجودیت درخواستی است و کلید خصوصی داده‌هایی است که فقط برای سرور شناخته شده است.

با وجود nonce این فرم، سرور پس از دریافت هدر احراز هویت کلاینت، بخش هش را مجددًا محاسبه می‌کند و در صورتی که با nonce از آن هدر مطابقت نداشته باشد یا مقدار time-stamp به اندازه کافی جدید نباشد، درخواست را رد می‌کند. به این ترتیب سرور می‌تواند مدت اعتبار nonce را محدود کند.

گنجاندن ETag از درخواست مجدد برای نسخه به روز شده منبع جلوگیری می‌کند. (توجه داشته باشید که قرار دادن آدرس IP کلاینت در nonce به نظر می‌رسد به سرور این امکان را می‌دهد که استفاده مجدد از nonce را به همان کلاینتی که در ابتدا آن را دریافت کرده است محدود کند. با این حال، این کار مزروعه‌های پروکسی (Proxy Farms) را که در آن درخواست‌های یک تک کاربر اغلب از طریق پروکسی‌های مختلف استفاده می‌کند. همچنین جعل آدرس IP چندان سخت نیست).

یک پیاده سازی ممکن است برای محافظت در برابر Replay Attack، یک nonce یا Digest استفاده شده قبلی را نپذیرد. یا، یک پیاده سازی ممکن است استفاده از Digest nonces یا Digest های یکباره را برای درخواست‌های GET و PUT یا POST برای درخواست‌های time-stamp انتخاب کند.

Symmetric Authentication

Digest Authentication را گسترش می‌دهد تا به کلاینت امکان تأیید اعتبار سرور را بدهد. این کار را با ارائه یک مقدار nonce کلاینت انجام می‌دهد، که سرور یک Digest پاسخ صحیح را بر اساس دانش صحیح اطلاعات مخفی مشترک ایجاد می‌کند. سپس سرور این Digest را در هدر Authorization-Info به کلاینت برمی‌گرداند.





این احراز هویت متقارن استاندارد RFC 2617 است. برای سازگاری با استاندارد قدیمی‌تر RFC 2069 اختیاری است، اما از آنجایی که پیشرفت‌های امنیتی مهمی را ارائه می‌کند، به همه کلاینت‌ها و سرورهای مدرن قویاً توصیه می‌شود که همه ویژگی‌های RFC 2617 را پیاده‌سازی کنند. به طور خاص، احراز هویت متقارن باید هر زمان که یک دستورالعمل **qop** وجود دارد انجام شود و لازم است زمانی که دستورالعمل **qop** وجود ندارد، انجام نشود.

پاسخ مانند **Digest** درخواست محاسبه می‌شود، با این تفاوت که اطلاعات بدن پیام (A2) متفاوت است، زیرا هیچ متدهای در پاسخ وجود ندارد و داده‌های موجودیت پیام متفاوت است. روش‌های محاسبه A2 برای **Digest** درخواست و پاسخ در جداول زیر مقایسه شده است.

qop	A2
undefined	<request-method>:<uri-directive-value>
auth	<request-method>:<uri-directive-value>
auth-int	<request-method>:<uri-directive-value>:H(<request-entity-body>)

qop	A2
undefined	:<uri-directive-value>
auth	:<uri-directive-value>
auth-int	:<uri-directive-value>:H(<response-entity-body>)

مقدار **cnonce** و مقدار **nc** باید برای درخواست کلاینت باشد که این پیام به آن پاسخ می‌دهد. اگر **nonce count** و **cnonce** مشخص شده باشد، دستورات پاسخ **qop="auth-int"** یا **qop="auth** باید وجود داشته باشند.

Quality of Protection Enhancements

فیلد **qop** ممکن است در هر سه هدر **Digest** وجود داشته باشد: **WWW-Authenticate**, **Authentication-Info** و **Authorization**.

فیلد **qop** به کلاینت‌ها و سرورها اجازه می‌دهد تا برای انواع و کیفیت‌های مختلف حفاظت، مذکوره کنند. برای مثال، برخی از تراکنش‌ها ممکن است بخواهند یکپارچگی متن پیام را بررسی کنند، حتی اگر این امر انتقال را به میزان قابل توجهی کند کند.

سرور ابتدا لیستی از گزینه‌های **qop** جدا شده با کاما را در هدر **WWW-Authenticate** صادر می‌کند. سپس کلاینت یکی از گزینه‌هایی را که پشتیبانی می‌کند و نیازهایش را برآورده می‌کند انتخاب می‌کند و آن را در قسمت **Authorization** **qop** خود به سرور ارسال می‌کند.





استفاده از **qop** اختیاری است، اما فقط برای سازگاری با مشخصات قدیمی RFC 2069. گزینه **qop** باید توسط همه پیاده سازی های **Digest** مدرن پشتیبانی شود.

RFC 2617 دو کیفیت اولیه ارزش حفاظتی را تعریف می کند: "auth" که نشان دهنده احراز هویت است و "aut-int" که نشان دهنده احراز هویت با محافظت از یکپارچگی پیام است.

Message Integrity Protection

اگر حفاظت یکپارچگی اعمال شود (**qop="auth-int"**), H (بدنه موجودیت) هش بدنه موجودیت است، نه بدنه پیام. قبل از اعمال رمزگذاری انتقال توسط فرستنده و پس از حذف آن توسط گیرنده محاسبه می شود. توجه داشته باشید که این شامل مرزهای چند قسمتی و هدرهای تعبیه شده در هر قسمت از هر نوع محتوای چند بخشی است.

Digest Authentication Headers

هر دو پروتکل احراز هویت **Basic** و **Digest** شامل یک چالش مجوز است که توسط هدر WWW-Authenticate و یک پاسخ مجوز که توسط هدر مجوز انجام می شود. احراز هویت **Digest** یک هدر اختیاری Authorization-Info را اضافه می کند که پس از احراز هویت موقتی آمیز ارسال می شود تا یک دست دادن سه مرحله ای تکمیل شود و در امتداد **nonce** بعدی استفاده شود. هدرهای احراز هویت **Basic** و **Digest** در جدول زیر نشان داده شده است.

هدرهای احراز هویت **Digest** کمی پیچیده تر هستند.





Phase	Basic	Digest
Challenge	WWW-Authenticate: Basic realm=<realm-value>	WWW-Authenticate: Digest realm=<realm-value> nonce=<nonce-value> [domain=<list-of-URIs>] [opaque=<opaque-token-value>] [stale=<true-or-false>] [algorithm=<digest-algorithm>] [qop=<list-of-qop-values>] [extension-directive]
Response	Authorization: Basic <base64(user:pass)>	Authorization: Digest username=<username> realm=<realm-value> nonce=<nonce-value> uri=<request-uri> response=<32-hex-digit-digest> [algorithm=<digest-algorithm>] [opaque=<opaque-token-value>] [cnonce=<nonce-value>] [qop=<qop-value>] [nc=<8-hex-digit-nonce-count>] [extension-directive]
Info	n/a	Authentication-Info: nextnonce=<nonce-value> [qop=<list-of-qop-values>] [rspauth=<hex-digest>] [cnonce=<nonce-value>] [nc=<8-hex-digit-nonce-count>]

Practical Considerations

چندین چیز وجود دارد که باید هنگام کار با احراز هویت Digest در نظر گرفته شود. این بخش برخی از این مسائل را مورد بحث قرار می‌دهد.

Multiple Challenges

یک سرور می‌تواند چندین چالش برای یک منبع ایجاد کند. به عنوان مثال، اگر سروی توانایی‌های یک کلاینت را نداند، ممکن است چالش‌های اساسی هم در احراز هویت Basic و هم Digest را فراهم کند. هنگام مواجهه با چالش‌های متعدد، کلاینت باید با قوی‌ترین مکانیزم احراز هویت که پشتیبانی می‌کند، پاسخ دهد.

اگر مقدار فیلد هدر WWW-Authenticate یا ProxyAuthenticate حاوی بیش از یک چالش باشد یا اگر بیش از یک فیلد هدر WWW-Authenticate ارائه شده باشد، باید در تجزیه و تحلیل مقدار فیلد WWW-Authenticate یا ProxyAuthenticate دقت ویژه‌ای داشته باشند، زیرا





چالش ممکن است خود حاوی لیستی از احراز هویت جدا شده با کاما باشد. توجه داشته باشید که بسیاری از مرورگرها فقط احراز هویت Basic را می‌شناسند و نیاز دارند که اولین مکانیزم احراز هویت ارائه شده باشد.

هنگام ارائه طیفی از گزینه‌های احراز هویت، نگرانی‌های امنیتی آشکاری مانند «ضعیفترین لینک» وجود دارد. سرورها باید احراز هویت Basic را تنها در صورتی شامل شوند که حداقل قابل قبول باشد و مدیران باید به کاربران در مورد خطرات اشتراک گذاری رمز عبور یکسان در سیستم‌ها در هنگام استفاده از سطوح مختلف امنیتی هشدار دهند.

Error Handling

در احراز هویت Digest، اگر یک دستورالعمل یا مقدار آن نامناسب باشد، یا اگر دستورالعمل مورد نیاز وجود نداشته باشد، پاسخ مناسب 400 Bad Request است.

اگر Digest درخواست مطابقت نداشته باشد، یک login failure های مکرر از یک کلاینت ممکن است نشان دهنده تلاش مهاجم برای حدس زدن رمزهای عبور باشد.

سرور احراز هویت باید اطمینان حاصل کند که منبع تعیین شده توسط دستورالعمل "uri" با منبع مشخص شده در خط درخواست یکسان است. اگر آن‌ها متفاوت هستند، سرور باید یک خطای Bad Request 400 را برگرداند. (از آنجایی که این ممکن است نشانه‌ای از یک حمله باشد، طراحان سرور ممکن است بخواهند ثبت چنین خطاهایی را در نظر بگیرند.) کپی کردن اطلاعات از URL درخواست در این فیلد با این احتمال که یک پروکسی میانی ممکن است خط درخواست کلاینت را تغییر دهد، سروکار دارد. این درخواست تغییر یافته منجر به خلاصه ای مشابه آنچه توسط کلاینت محاسبه می‌شود، نمی‌شود.

Protection Spaces

مقدار Realm، در ترکیب با URL ریشه متعارف سرور مورد دسترسی، فضای حفاظتی را مشخص می‌کند.

Realm‌ها به منابع محافظت شده روی یک سرور اجازه می‌دهند که به مجموعه‌ای از فضاهای حفاظتی تقسیم شوند که هر کدام دارای طرح احراز هویت و/یا پایگاه داده مجوز خود هستند. مقدار Realm رشته‌ای است که معمولاً توسط سرور مبدأ اختصاص داده می‌شود، که ممکن است معنای خاصی برای طرح احراز هویت داشته باشد. توجه داشته باشید که ممکن است چالش‌های متعددی با طرح مجوز یکسان اما حوزه‌های مختلف وجود داشته باشد.

فضای حفاظتی، دامنه‌ای را تعیین می‌کند که Credential‌ها می‌توانند به طور خودکار روی آن اعمال شوند. اگر درخواست قبلی مجوز داده شده باشد، می‌توان از همان Credential‌ها برای تمام





در خواستهای دیگر در آن فضای حفاظتی برای مدت زمان تعیین شده توسط طرح احراز هویت، پارامترها و یا اولویت کاربر استفاده مجدد کرد. مگر اینکه توسط طرح احراز هویت به گونه دیگری تعریف شده باشد، یک فضای حفاظتی واحد نمی‌تواند خارج از محدوده سرور خود گسترش یابد.

محاسبه خاص فضای حفاظتی به مکانیسم احراز هویت بستگی دارد:

در احراز هویت Basic، کلاینتها فرض می‌کنند که تمام مسیرها در URI درخواست یا پایین‌تر از آن، در همان فضای حفاظتی چالش فعلی قرار دارند. یک کلاینت می‌تواند به طور پیشگیرانه برای منابع در این فضا مجوز دهد بدون اینکه منتظر چالش دیگری از طرف سرور باشد.

در احراز هویت Digest، فیلد دامنه WWW-Authenticate: دقیقاً فضای حفاظتی را تعریف می‌کند. فیلد دامنه یک لیست نقل قول و جدا شده از URI‌ها است. همه URI‌های موجود در لیست دامنه و همه URI‌های منطقی زیر این پیشوندها، در فضای حفاظتی یکسانی قرار دارند. اگر فیلد دامنه گم یا خالی باشد، تمام URI‌ها در سرور Challenging Server در فضای حفاظتی قرار دارند.

Rewriting URIs

پراکسی‌ها ممکن است URI‌ها را به گونه‌ای بازنویسی کنند که Syntax مربوط به URI را تغییر دهد اما منبع واقعی توصیف شده را تغییر ندهد. مثلا:

- نام هاست ممکن است عادی شده یا با آدرس‌های IP جایگزین شوند.
- کarakترهای تعبیه شده (Embedded) ممکن است با فرم‌های Escape "%" جایگزین شوند.
- ویژگی‌های اضافی از نوعی که بر منبع Fetch شده از سرور مبدا خاص تأثیر نمی‌گذارد ممکن است به URI اضافه یا درج شود.

از آنجا که URI‌ها را می‌توان توسط پراکسی‌ها تغییر داد و از آنجایی که sanity احراز هویت Digest یکپارچگی مقدار URI را بررسی می‌کند، در صورت ایجاد هر یک از این تغییرات، احراز هویت Digest خراب می‌شود.

Caches

هنگامی که یک Shared Cache درخواستی حاوی یک هدر Authorization و پاسخی از ارسال آن درخواست دریافت می‌کند، نباید آن پاسخ را به عنوان پاسخ به درخواست دیگری برگرداند، مگر اینکه یکی از دو دستورالعمل Cache-Control در پاسخ وجود داشته باشد:





- اگر پاسخ اولیه شامل دستور العمل Cache-Control باشد، must-revalidate ممکن است از موجودیت آن پاسخ در پاسخ به درخواست بعدی استفاده کند. با این حال، ابتدا باید با استفاده از هدرهای درخواست مربوط به درخواست جدید، آن را با سرور مبدا تأیید مجدد کند تا سرور مبدا بتواند درخواست جدید را تأیید کند.
- اگر پاسخ اصلی شامل دستور العمل Public Cache-Control باشد، موجودیت پاسخ ممکن است در پاسخ به هر درخواست بعدی بازگردانده شود.

Security Considerations

RFC 2617 در خلاصه کردن برخی از خطرات امنیتی ذاتی در طرح های احراز هویت HTTP کار قابل تحسینی انجام می دهد. این بخش برخی از این خطرات را شرح می دهد.

Header Tampering

برای ارائه یک سیستم بدون خطا در برابر دستکاری هدر، به رمزگذاری سرتاسر یا امضای دیجیتال هدرها نیاز دارید (ترجیحاً ترکیبی از هر دو!)

احراز هویت Digest بر ارائه یک طرح احراز هویت ضد دستکاری متمرکز است، اما لزوماً این حفاظت را به داده ها گسترش نمی دهد. تنها هدرهایی که دارای سطحی از حفاظت هستند WWW-Authenticate و Authorization هستند.

Replay Attacks

Replay Attack، در شرایط فعلی، زمانی است که شخصی از مجموعه ای از Credential های احراز هویت پنهان شده از یک تراکنش معین برای تراکنش دیگر استفاده می کند. در حالی که این مشکل در درخواست های GET وجود دارد، بسیار مهم است که یک روش بی خطر برای جلوگیری از Replay Attack برای درخواست های POST و PUT در دسترس باشد. توانایی Replay موفقیت آمیز Credential های استفاده شده قبلی در حین انتقال داده های فرم می تواند باعث ایجاد کابوس های امنیتی شود.

بنابراین، برای اینکه سرور Credential های Replay شده را بپذیرد، مقادیر nonce باید تکرار شوند. یکی از راه های کاهش این مشکل این است که سرور یک nonce تولید کند که حاوی Digest ای از آدرس IP کلاینت، یک مهر زمانی، ETag منبع و یک کلید سرور خصوصی (همانطور که قبلاً توصیه شد). در چنین سناریویی، ترکیب یک آدرس IP و یک مقدار کوتاه مدت ممکن است مانع بزرگی برای مهاجم باشد.





با این حال، این راه حل یک اشکال اساسی دارد. همانطور که قبلاً بحث کردیم، مشکل، استفاده از آدرس IP کلاینت در ایجاد یک انتقال بدون وقفه از طریق Proxy Farms، که در آن درخواست‌های یک کاربر ممکن است از طریق پراکسی‌های مختلف انجام شود، می‌باشد. همچنین جعل IP چندان سخت نیست.

یکی از راه‌های جلوگیری از Replay Attack، استفاده از یک مقدار nonce منحصر به فرد برای هر تراکنش است. در این پیاده سازی، برای هر تراکنش، سرور یک nonce منحصر‌بفرد به همراه مقدار timeout صادر می‌کند. مقدار nonce صادر شده فقط برای تراکنش داده شده معتبر است و فقط برای مدت زمان مقدار وقفه. این حسابداری ممکن است بار روی سرورها را افزایش دهد. با این حال، افزایش باید اندک باشد.

Multiple Authentication Mechanisms

هنگامی که یک سرور از طرح‌های احراز هویت چندگانه (مانند Basic و Digest) پشتیبانی می‌کند، معمولاً انتخاب را در هدرهای WWW-Authenticate ارائه می‌کند. از آنجایی که کلاینت ملزم به انتخاب قوی‌ترین مکانیسم احراز هویت نیست، قدرت احراز هویت حاصله به اندازه ضعیفترین طرح‌های احراز هویت است.

راه‌های واضح برای جلوگیری از این مشکل این است که کلاینت‌ها همیشه قوی‌ترین طرح احراز هویت موجود را انتخاب کنند. اگر این عملی نباشد (چون اکثر ما از کلاینت‌های تجاری موجود استفاده می‌کنیم)، تنها گزینه دیگر استفاده از سرور پروکسی برای حفظ قوی‌ترین طرح احراز هویت است. با این حال، چنین رویکردی تنها در حوزه‌ای امکان‌پذیر است که در آن همه کلاینت‌ها می‌توانند از طرح احراز هویت انتخاب شده پشتیبانی کنند - به عنوان مثال، یک شبکه شرکتی.

Dictionary Attacks

حملات دیکشنری حملات معمولی با حدس زدن رمز عبور هستند. یک کاربر مخرب می‌تواند یک تراکنش را استراق سمع کند و از یک برنامه حدس زدن رمز عبور استاندارد در برابر جفت‌های nonce/response استفاده کند. اگر کاربران از رمزهای عبور نسبتاً ساده استفاده می‌کنند و سرورها از nonces ساده استفاده می‌کنند، یافتن یک مطابقت کاملاً ممکن است. اگر Aging Policy رمز عبور وجود نداشته باشد، با توجه به زمان کافی و هزینه یکباره شکستن رمزهای عبور، جمع آوری رمزهای عبور کافی برای آسیب واقعی آسان است.

واقعاً هیچ راه خوبی برای حل این مشکل وجود ندارد، به جز استفاده از رمزهای عبور نسبتاً پیچیده که به سختی شکسته می‌شوند و یک Aging Policy خوب رمز عبور.





Hostile Proxies and Man-in-the-Middle Attacks

امروزه بسیاری از ترافیک اینترنت از طریق یک پروکسی در یک نقطه دیگر انجام می‌شود. با ظهور تکنیک‌های تغییر مسیر و رهگیری پراکسی‌ها، کاربر ممکن است حتی متوجه نشود که درخواست او از طریق یک پروکسی انجام می‌شود. اگر یکی از آن پراکسی‌ها متخاصل می‌باشد، می‌تواند کلاینت را در برابر حمله آسیب‌پذیر کند.

چنین حمله‌ای می‌تواند به شکل استراق سمع یا تغییر طرح‌های احراز هویت موجود با حذف همه گزینه‌های ارائه شده و جایگزینی آن‌ها با ضعیفترین طرح احراز هویت (مانند احراز هویت Basic) باشد.

یکی از راه‌های به خطر انداختن یک پروکسی قابل اعتماد، استفاده از رابطه‌ای افزونه (Extension Interfaces) آن است. پروکسی‌ها گاهی اوقات رابطه‌ای برنامه نویسی پیچیده‌ای را ارائه می‌دهند و با چنین پراکسی‌هایی ممکن است امکان نوشتن یک افزونه (به عنوان مثال، افزونه) برای رهگیری و تغییر ترافیک وجود داشته باشد.

هیچ راه خوبی برای رفع این مشکل وجود ندارد. راه حل‌های ممکن شامل ارائه سرنخ‌های بصری در مورد قدرت احراز هویت، پیکربندی کلاینت‌ها برای استفاده از قوی‌ترین احراز هویت ممکن، و غیره است، اما حتی زمانی که از قوی‌ترین طرح احراز هویت ممکن استفاده می‌شود، کلاینت‌ها همچنان در برابر استراق سمع آسیب‌پذیر هستند. تنها راه مطمئن برای محافظت در برابر این حملات استفاده از SSL است.

Chosen Plaintext Attacks

کلاینت‌هایی که از احراز هویت Digest استفاده می‌کنند از nonce ارائه شده توسط سرور برای تولید پاسخ استفاده می‌کنند. با این حال، اگر یک پروکسی آسیب‌دیده یا مخرب در وسط وجود داشته باشد که ترافیک را رهگیری می‌کند (یا یک سرور مبدا مخرب)، می‌تواند به راحتی یک nonce را برای محاسبه پاسخ توسط کلاینت ارائه کند. استفاده از کلید شناخته شده برای محاسبه پاسخ ممکن است تحلیل رمزی پاسخ را آسان‌تر کند. این حمله متن ساده انتخاب شده (Chosen Plaintext Attacks) نامیده می‌شود. چند نوع از حملات متن ساده انتخاب شده وجود دارد:

Precomputed dictionary attacks

این ترکیبی از حمله دیکشنری و حمله متن ساده انتخاب شده است. ابتدا، سرور حمله کننده مجموعه‌ای از پاسخ‌ها را با استفاده از تغییرات غیرمعمول از پیش تعیین شده و رمز عبور رایج ایجاد می‌کند و یک دیکشنری ایجاد می‌کند. هنگامی که یک دیکشنری قابل توجه در دسترس است، سرور/پراکسی مهاجم می‌تواند ممنوعیت ترافیک را کامل کند و شروع به ارسال nonces از پیش تعیین شده برای کلاینت‌ها





کند. هنگامی که از یک کلاینت پاسخ دریافت می‌کند، مهاجم دیکشنری تولید شده را برای موارد مشابه جستجو می‌کند. اگر مطابقت وجود داشته باشد، مهاجم رمز عبور آن کاربر خاص را دارد.

Batched brute-force attacks

تفاوت در حمله گروهی **brute-force** در محاسبه رمز عبور است. به جای تلاش برای مطابقت با یک کلاینت محاسبه شده، مجموعه‌ای از ماشین‌ها روی شمارش همه رمزهای عبور ممکن برای یک فضای معین کار می‌کنند. همانطور که ماشین‌ها سریعتر می‌شوند، حمله **brute-force** بیشتر و بیشتر قابل اجرا می‌شود.

به طور کلی، تهدید ناشی از این حملات به راحتی قابل مقابله است. یکی از راههای جلوگیری از آن‌ها، پیکربندی کلاینت‌ها برای استفاده از دستورالعمل **cnonce** اختیاری است، به طوری که پاسخ به صلاحیت کلاینت تولید می‌شود، نه از **nonce** ارائه شده توسط سرور (که می‌تواند توسط مهاجم به خطر بیفت). این، همراه با سیاست‌های اعمال رمزهای عبور نسبتاً قوی و یک مکانیسم خوب **aging** رمز عبور، می‌تواند خطر حملات متن ساده انتخابی را به طور کامل کاهش دهد.

Storing Passwords

mekanizm احراز هویت **Digest**، پاسخ کاربر را با آنچه در داخل سرور ذخیره می‌شود مقایسه می‌کند - معمولاً نام‌های کاربری و تاپل‌های $H(A1)$ از **Digest** نام کاربری، **Realm** و رمز عبور مشتق می‌شود.

بر خلاف فایل رمز عبور سنتی در جعبه یونیکس، اگر فایل رمز عبور احراز هویت **Digest** به خطر بیفت، تمام اسناد موجود در **Realm** فوراً در دسترس مهاجم هستند. نیازی به مرحله رمزگشایی نیست.

برخی از راههای کاهش این مشکل عبارتند از:

- از فایل گذرواژه طوری محافظت کنید که گویی حاوی رمزهای عبور متنی واضح است.
- اطمینان حاصل کنید که نام **Realm** در بین همه **Realm**‌ها منحصر به فرد است، به طوری که اگر یک فایل رمز عبور به خطر بیفت، آسیب به یک **Realm** خاص محلی شود. یک نام **Realm** کاملاً واحد شرایط با میزبان و دامنه باید این نیاز را برآورده کند.

در حالی که احراز هویت **Digest** راه حلی بسیار قوی‌تر و ایمن‌تر از احراز هویت **Basic** ارائه می‌دهد، اما هنوز هیچ گونه حفاظتی برای امنیت محتوا ارائه نمی‌دهد - یک تراکنش واقعاً امن فقط از طریق **SSL** امکان پذیر است که در فصل بعدی توضیح می‌دهیم.



For More Information

<http://www.ietf.org/rfc/rfc2617.txt>





فصل چهاردهم - Secure HTTP

در سه فصل قبل ویژگی‌های HTTP مورد بررسی قرار گرفت که به شناسایی و احراز هویت کاربران کمک می‌کند. این تکنیک‌ها در یک جامعه دوستانه به خوبی کار می‌کنند، اما به اندازه کافی قوی نیستند تا از تراکنش‌های مهم در برابر جامعه متشکل از دشمنان با انگیزه و متخاصل محافظت کنند.

این فصل یک فناوری پیچیده‌تر و تهاجمی‌تر را برای ایمن کردن تراکنش‌های HTTP از شنود و دستکاری با استفاده از رمزگاری دیجیتال ارائه می‌کند.

Making HTTP Safe

مردم از تراکنش‌های وب برای کارهای جدی استفاده می‌کنند. بدون امنیت قوی، آن‌ها در انجام خرید آنلайн و بانکداری احساس راحتی نخواهند کرد. شرکت‌ها نمی‌توانند بدون اینکه بتوانند دسترسی را محدود کنند، اسناد مهم را روی سرورهای وب قرار دهند. وب به فرم امن HTTP نیاز دارد.

فصل‌های قبلی در مورد برخی از روش‌های سبک برای ارائه احراز هویت (تأیید هویت Basic و Digest) و یکپارچگی پیام (digest qop = auth-int) صحبت کردند. این طرح‌ها برای بسیاری از اهداف خوب هستند، اما ممکن است برای خریدهای بزرگ، تراکنش‌های بانکی یا دسترسی به داده‌های محرمانه به اندازه کافی قوی نباشند. برای این تراکنش‌های جدی‌تر، HTTP را با فناوری رمزگذاری دیجیتال ترکیب می‌کنیم.

نسخه ایمن HTTP باید کارآمد، قابل حمل، دارای مدیریت آسان و سازگار با دنیای در حال تغییر باشد. همچنین باید الزامات اجتماعی و دولتی را برآورده کند. ما به یک فناوری برای امنیت HTTP نیاز داریم که موارد زیر را ارائه دهد:

- احراز هویت سرور (کاربران می‌دانند که با سرور واقعی صحبت می‌کنند، نه ساختگی)
- احراز هویت کلاینت (سرورها می‌دانند که با کاربر واقعی صحبت می‌کنند، نه جعلی)
- یکپارچگی (کلاینت‌ها و سرورها از تغییر داده‌های ایشان در امان هستند)
- رمزگذاری (کلاینت‌ها و سرورها به صورت خصوصی بدون ترس از استراق سمع صحبت می‌کنند)
- کارایی (الگوریتمی به اندازه کافی سریع برای استفاده از کلاینت‌ها و سرورهای ارزان قیمت)
- Ubiquity (پروتکل‌ها تقریباً توسط همه کلاینت‌ها و سرورها پشتیبانی می‌شوند)
- مقیاس پذیری اداری (ارتبط امن فوری برای هر کسی، در هر مکان)
- سازگاری (از بهترین روش‌های امنیتی شناخته شده روز پشتیبانی می‌کند)
- بقای اجتماعی (برآورنده نیازهای فرهنگی و سیاسی جامعه)

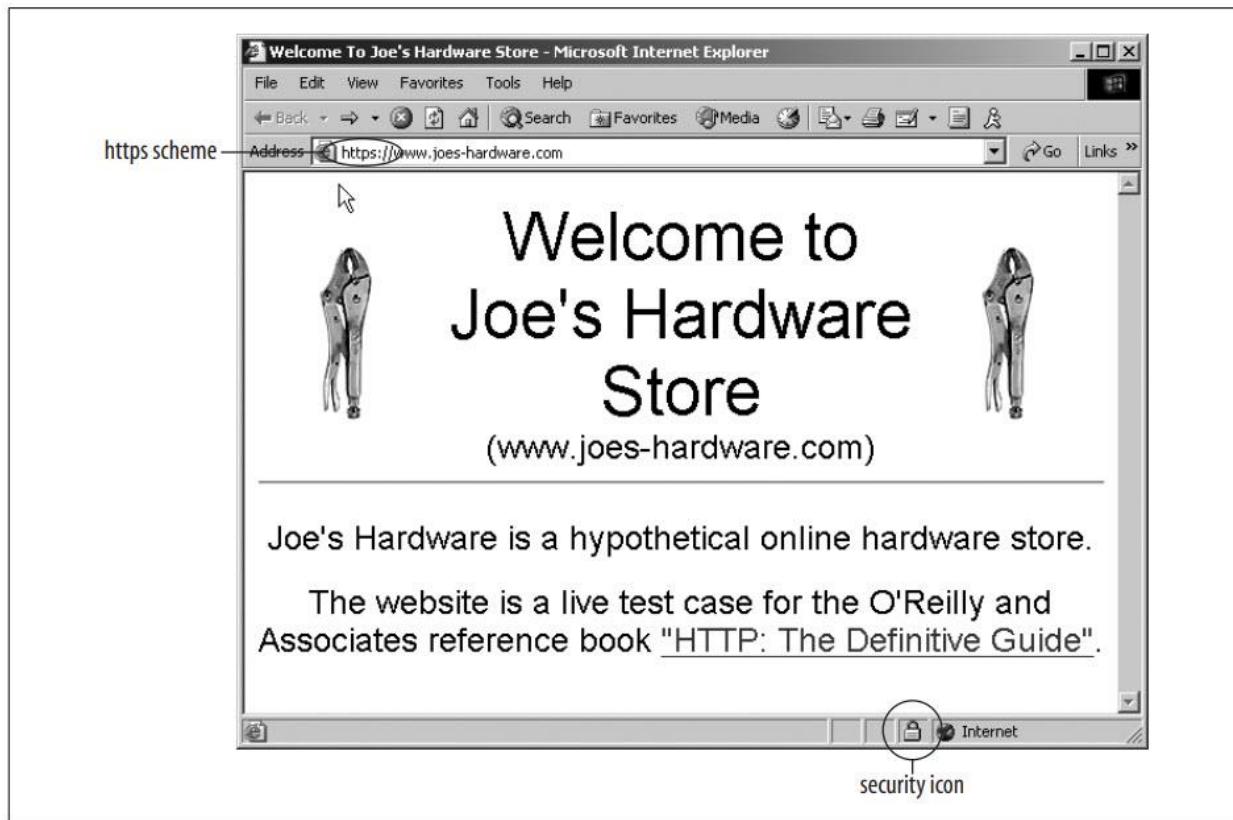




HTTPS

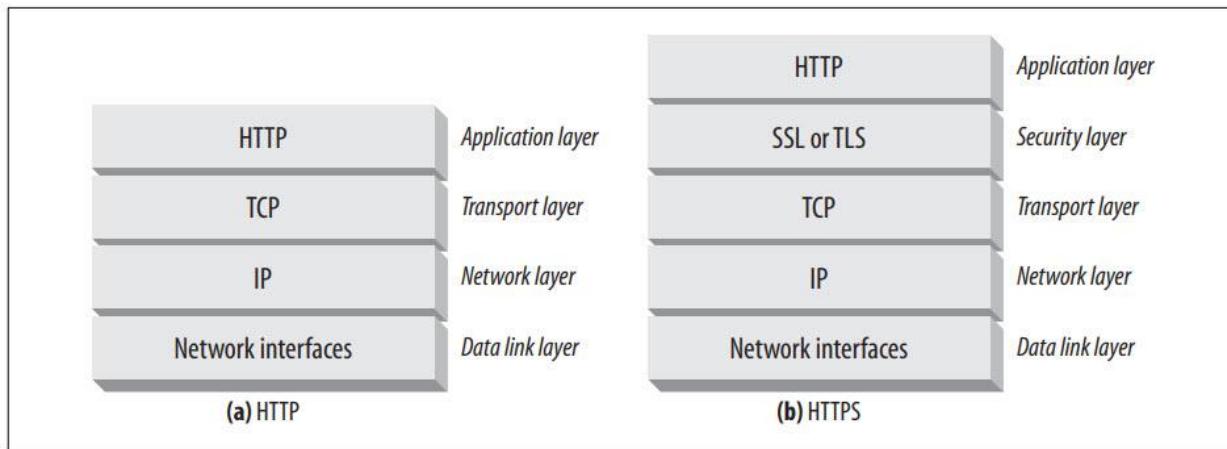
HTTPS محبوب‌ترین شکل امن HTTP است. این توسط Netscape Communications Corporation پیشگام بود و توسط تمام مرورگرها و سرورهای اصلی پشتیبانی می‌شود.

می‌توانید تشخیص دهید که آیا یک صفحه وب از طریق HTTPS به جای HTTP دسترسی داشته است، زیرا URL به جای http:// با طرح https:// شروع می‌شود (برخی از مرورگرها نیز نشانه‌های امنیتی نمادین را نشان می‌دهند، همانطور که در شکل زیر نشان داده است).



هنگام استفاده از HTTPS، تمام داده‌های درخواست و پاسخ HTTP قبل از ارسال در سراسر شبکه رمزگذاری می‌شوند. HTTPS با ارائه یک لایه امنیتی رمزنگاری در سطح انتقال - با استفاده از SSL یا جانشین آن، TLS - در زیر HTTP (شکل زیر) کار می‌کند. از آنجایی که SSL و TLS بسیار شبیه هستند، در این کتاب ما از اصطلاح "SSL" برای نشان دادن SSL و TLS استفاده می‌کنیم.





از آنجایی که بیشتر کارهای سخت رمزگذاری و رمزگشایی در کتابخانه‌های SSL انجام می‌شود، کلاینت‌ها و سرورهای وب برای استفاده از HTTP ایمن نیازی به تغییر منطق پردازش پروتکل خود ندارند.

در بیشتر موارد، آن‌ها به سادگی نیاز دارند که تماس‌های ورودی/خروجی TCP را با تماس‌های SSL جایگزین کنند و چند تماس دیگر را برای پیکربندی و مدیریت اطلاعات امنیتی اضافه کنند.

Digital Cryptography

قبل از اینکه به تفصیل در مورد HTTPS صحبت کنیم، باید پیشینه کمی در مورد تکنیک‌های رمزگذاری رمزگاری شده مورد استفاده توسط SSL و HTTPS ارائه دهیم. در چند بخش بعدی، ما یک آغازگر سریع از ملزومات رمزگاری دیجیتال ارائه خواهیم داد. اگر قبلاً با فن آوری و اصطلاحات رمزگاری دیجیتال آشنا هستید، به راحتی به بخش «HTTPS: The Details» بروید.

Ciphers

الگوریتم‌هایی برای رمزگذاری متن برای غیرقابل خواندن آن برای افراد غیرمجاز

Keys

پارامترهای عددی که رفتار رمزها را تغییر می‌دهند.

Symmetric-key cryptosystems

الگوریتم‌هایی که از یک کلید برای رمزگذاری و رمزگشایی استفاده می‌کنند.

Asymmetric-key cryptosystems

الگوریتم‌هایی که از کلیدهای مختلفی برای رمزگذاری و رمزگشایی استفاده می‌کنند.





Public-key cryptography

سیستمی که ارسال پیام‌های مخفی را برای میلیون‌ها کامپیوتر آسان می‌کند.

Digital signatures

Checksum هایی که تأیید می‌کنند پیام جعلی یا دستکاری نشده است.

Digital certificates

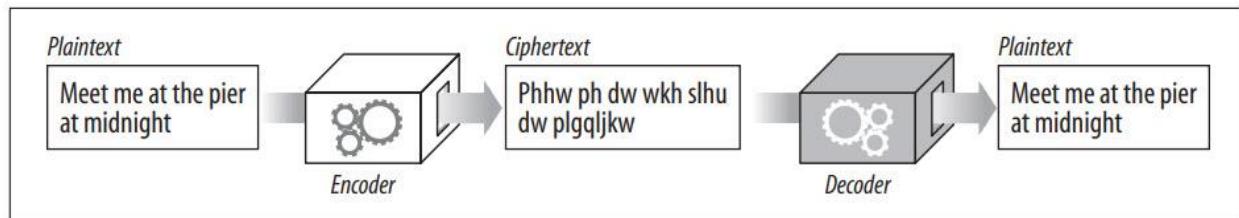
اطلاعات شناسایی، تایید شده و توسط یک سازمان مورد اعتماد امضا شده است.

The Art and Science of Secret Coding

Cryptography هنر و علم رمزگذاری و رمزگشایی پیام‌ها است. هزاران سال است که مردم از روش‌های رمزگاری برای ارسال پیام‌های مخفی استفاده می‌کنند. با این حال، Cryptography می‌تواند بیشتر از رمزگذاری پیام‌ها برای جلوگیری از خوانتن توسط افراد غیرمجاز انجام دهد. همچنین می‌توان از آن برای جلوگیری از دستکاری پیام‌ها استفاده کرد. رمزگاری حتی می‌تواند برای اثبات اینکه شما واقعاً یک پیام یا تراکنش را نوشته‌اید، درست مانند امضای دستنویس شما روی چک یا مهر و موم برجسته روی یک پاکت استفاده شود.

Ciphers

Cryptography مبتنی بر کدهای مخفی به نام Cipher است. یک طرح کدگذاری است - یک راه خاص برای رمزگذاری یک پیام و یک روش همراه برای رمزگشایی Secret. پیام اصلی، قبل از اینکه کدگذاری شود، اغلب Plaintext یا Cleartext نامیده می‌شود. پیام رمزگذاری شده، پس از اعمال رمز، اغلب Ciphertext نامیده می‌شود. شکل زیر یک مثال ساده را در این مورد نشان می‌دهد.

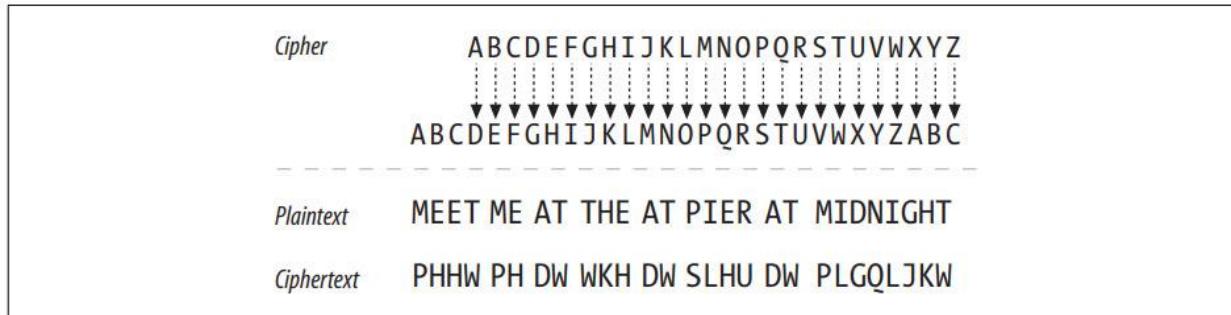


هزاران سال است که از رمزها برای تولید پیام‌های مخفی استفاده می‌شود. افسانه‌ها حاکی از آن است که ژولیوس سزار از رمز چرخشی سه کاراکتری استفاده می‌کرد که در آن هر کاراکتر در پیام با یک کاراکتر در سه موقعیت حروف الفبا به جلو جایگزین می‌شد. در الفبای مدرن ما، A با D، B با E و غیره جایگزین می‌شود.





به عنوان مثال، در شکل زیر، پیام "Meet me at the Pier at Midnight" در متن رمزی "phhw ph" با استفاده از رمز rot3 (چرخش با ۳ کاراکتر) رمزگذاری می‌شود. متن رمز "dw wkh slhu dw plgqljkw" را می‌توان با اعمال کدگذاری معکوس، چرخاندن -۳ کاراکتر در حروف الفبا، به پیام متنی اصلی برگرداند.



برای سادگی مثال، ما علائم نگارشی یا فضای خالی را نمی‌چرخانیم، اما شما می‌توانید این کار را هم انجام دهید.

Cipher Machines

رمزها به عنوان الگوریتم‌های نسبتاً ساده شروع شدند، زیرا انسان‌ها باید خودشان رمزگذاری و رمزگشایی را انجام دهند. از آنجایی که رمزها ساده بودند، مردم می‌توانستند با استفاده از مداد و کاغذ و کتاب‌های کد، کدها را شناسایی کنند. با این حال، برای افراد باهوش نیز این امکان وجود داشت که به راحتی کدها را بشکنند.

با پیشرفت تکنولوژی، مردم شروع به ساخت ماشین‌هایی کردند که می‌توانستند با استفاده از رمزهای بسیار پیچیده‌تر پیام‌ها را به سرعت و با دقت رمزگذاری و رمزگشایی کنند. این ماشین‌های رمزگذاری می‌توانند به جای انجام چرخش‌های ساده، کاراکترها را جایگزین کنند، ترتیب کاراکترها را جابه‌جا کنند، و پیام‌ها را برش دهنند تا شکستن کدها بسیار سخت‌تر شود.

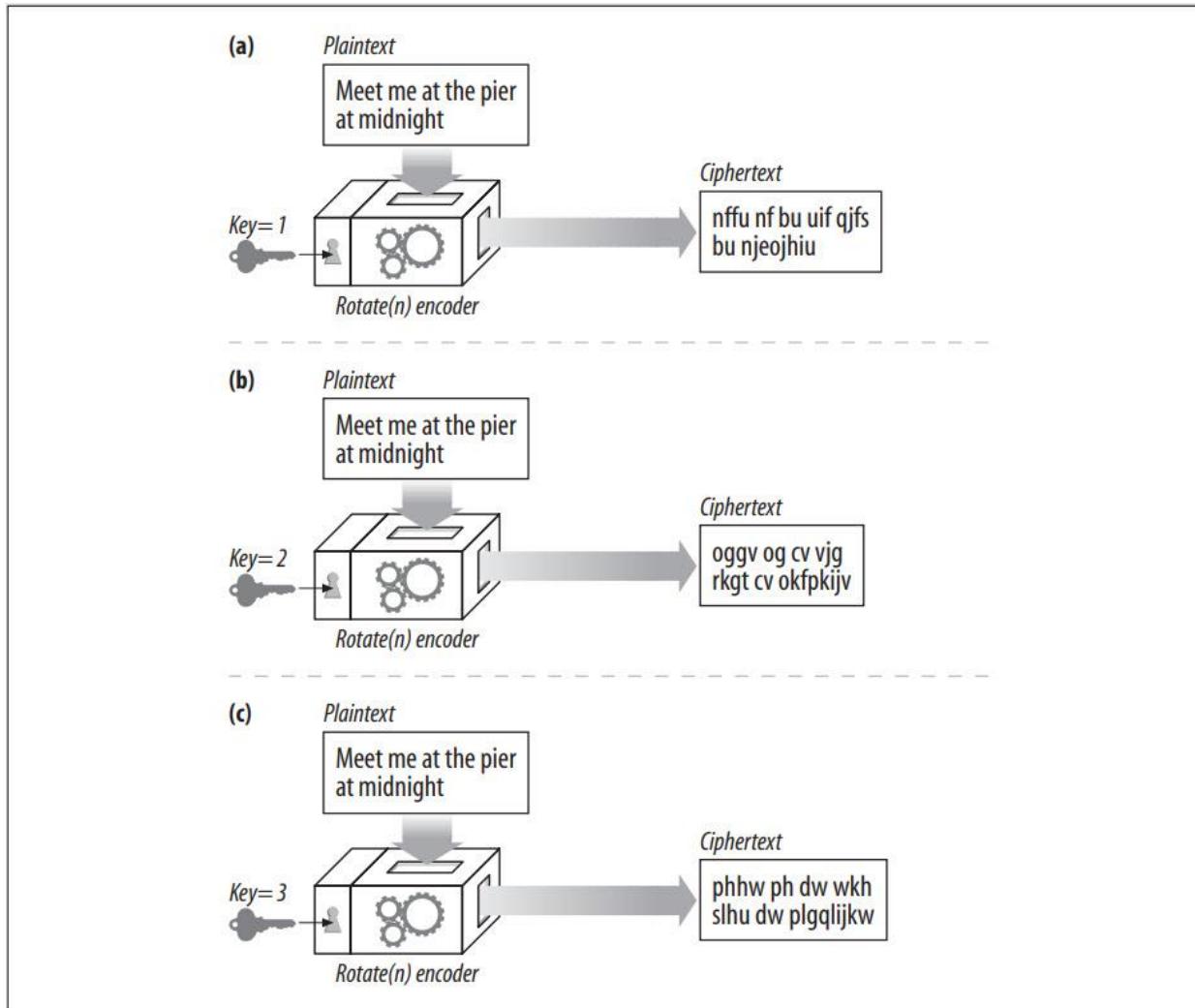
Keyed Ciphers

از آنجایی که الگوریتم‌های کد و ماشین‌ها می‌توانستند به دست دشمن بیفتد، اکثر ماشین‌ها شماره‌گیری داشتند که می‌توان آن‌ها را روی تعداد زیادی مقادیر مختلف تنظیم کرد که نحوه کار رمز را تغییر می‌داد. حتی اگر دستگاه به سرقت رفته باشد، بدون تنظیمات شماره‌گیری مناسب (مقادیر کلید) رمزگشایی نمی‌کند.

این پارامترهای رمز را کلید می‌نامیدند. برای اینکه فرآیند رمزگشایی به درستی کار کند، باید کلید مناسب را در دستگاه رمزگذاری وارد کنید. کلیدهای رمز باعث می‌شوند که یک ماشین رمز مانند مجموعه‌ای از بسیاری از ماشین‌های رمزگذاری مجازی عمل کند که هر کدام به دلیل داشتن مقادیر کلیدی متفاوت، رفتار متفاوتی دارند.



شکل زیر نمونه ای از رمزهای کلیددار را نشان می دهد.



الگوریتم رمز، رمز اولیه N توسط کلید کنترل می شود. همان پیام ورودی، "Meet me at the Pier at Midnight" که از طریق یک دستگاه رمزگذاری ارسال می شود، بسته به مقدار کلید، خروجی های متفاوتی تولید می کند. امروزه تقریباً همه الگوریتم های رمزنگاری از کلیدها استفاده می کنند.

Digital Ciphers

با ظهور محاسبات دیجیتال، دو پیشرفت عمده رخ داد:

- رمزگذاری و الگوریتم های رمزگشایی پیچیده امکان پذیر شد و از محدودیت های سرعت و عملکرد ماشین های مکانیکی رها شد.

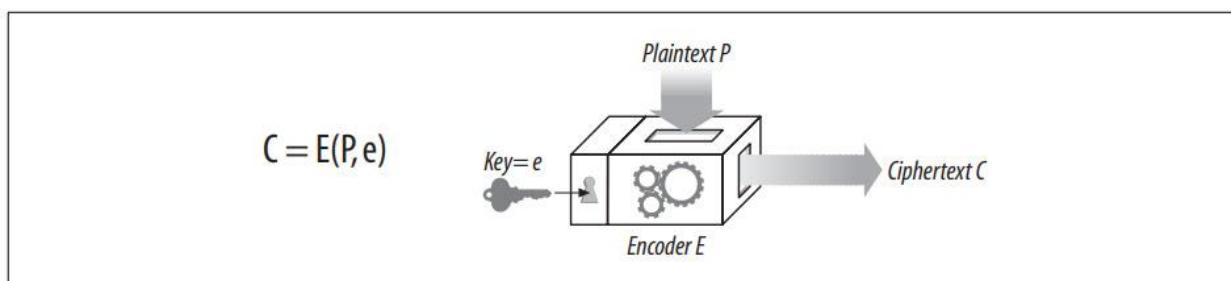




پشتیبانی از کلیدهای بسیار بزرگ امکان پذیر شد، به طوری که یک الگوریتم رمز می‌تواند تریلیون‌ها الگوریتم رمز مجازی را به دست آورد که هر کدام با مقدار کلید متفاوت است. هر چه کلید طولانی‌تر باشد، ترکیب‌های بیشتری از رمزگذاری‌ها امکان‌پذیر است و شکستن کد با حدس زدن تصادفی کلیدها سخت‌تر می‌شود.

برخلاف کلیدهای فلزی فیزیکی یا تنظیمات شماره‌گیری در دستگاه‌های مکانیکی، کلیدهای دیجیتال فقط اعداد هستند. این مقادیر کلید دیجیتال، ورودی به الگوریتم‌های رمزگذاری و رمزگشایی هستند. الگوریتم‌های کدگذاری توابعی هستند که تکه‌ای از داده‌ها را می‌گیرند و بر اساس الگوریتم و مقدار کلید آن را رمزگذاری/رمزگشایی می‌کنند.

با توجه به یک پیام متنی ساده به نام P ، یک تابع رمزگذاری به نام E ، و یک کلید رمزگذاری دیجیتالی به نام e ، می‌توانید یک پیام متن رمز شده C ایجاد کنید (شکل زیر).



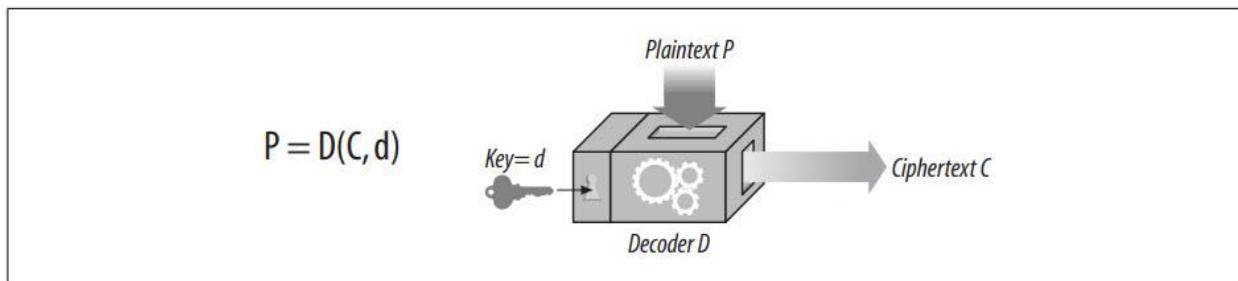
می‌توانید با استفاده از تابع رمزگشایی D و کلید رمزگشایی d ، متن رمز C را به متن اصلی P رمزگشایی کنید. البته، توابع رمزگشایی و رمزگذاری معکوس یکدیگر هستند. رمزگشایی رمزگذاری P پیام اصلی P را برمی‌گرداند.

Symmetric-Key Cryptography

بیایید با جزئیات بیشتری در مورد نحوه کار کلیدها و رمزها با هم صحبت کنیم. بسیاری از الگوریتم‌های رمز دیجیتال رمزهای کلید متقارن نامیده می‌شوند، زیرا از همان مقدار کلید برای رمزگذاری استفاده می‌کنند که برای رمزگشایی استفاده می‌کنند ($d = e$). بیایید کلید را k صدا کنیم.

در رمزگاری کلید متقارن، هم فرستنده و هم گیرنده باید یک کلید مخفی مشترک، k ، برای برقراری ارتباط داشته باشند. فرستنده از کلید مخفی مشترک برای رمزگذاری پیام استفاده می‌کند و متن رمزی حاصل را به گیرنده می‌فرستد. گیرنده متن رمز را می‌گیرد و تابع رمزگشایی را همراه با همان کلید مخفی مشترک برای بازیابی متن اصلی اعمال می‌کند (شکل زیر).





برخی از الگوریتم‌های رمزگاری متقاضی رایج عبارتند از DES، Triple-DES، RC2 و RC4.

Key Length and Enumeration Attacks

بسیار مهم است که کلیدهای مخفی مخفر بمانند. در بیشتر موارد، الگوریتم‌های رمزگذاری و رمزگشایی دانش عمومی هستند، بنابراین کلید تنها چیزی است که مخفی است!

یک الگوریتم رمزگذاری خوب، نفوذگر را مجبور می‌کند تا تک تک مقادیر کلیدی ممکن در جهان را برای شکستن کد امتحان کند. آزمایش تمام مقادیر کلیدی بوسیله Brute Force، حمله Enumeration نامیده می‌شود. اگر فقط چند مقدار کلیدی ممکن وجود داشته باشد، یک فرد بدخواه می‌تواند همه آن‌ها را با Brute Force مرسور کند و در نهایت کد را بشکند. اما اگر مقادیر کلیدی زیادی وجود داشته باشد، ممکن است روزها، سال‌ها یا حتی به اندازه عمر کیهان طول بکشد تا به دنبال کلیدی باشد که رمز را بشکند.

تعداد مقادیر کلید ممکن بستگی به تعداد بیت‌های موجود در کلید و تعداد کلیدهای ممکن معتبر دارد. برای رمزهای کلید متقارن، معمولاً همه مقادیر کلید معتبر هستند. یک کلید ۸ بیتی فقط ۲۵۶ کلید ممکن، یک کلید ۱۶ بیتی دارای ۴۰ کلید ممکن (حدود یک تریلیون کلید) و یک کلید ۳۲ بیتی است. کلید حدود ۴۰ بیتی دارای ۴۰ کلید ممکن را تولید می‌کند.

برای رمزنگاری‌های متقارن، کلیدهای ۴۰ بیتی برای تراکنش‌های کوچک و غیر بحرانی به اندازه کافی این در نظر گرفته می‌شوند. با این حال، آن‌ها توسط ایستگاه‌های کاری پرسرعت امروزی که اکنون می‌توانند میلیارد دها محاسبه در ثانیه انجام دهند، شکستنی هستند.

در مقابل، کلیدهای ۱۲۸ بیتی برای رمزنگاری کلید متقارن بسیار قوی در نظر گرفته می‌شوند. در واقع، کلیدهای بلند چنان تأثیری بر امنیت رمزنگاری دارند که دولت ایالات متحده کنترل صادرات را بر روی نرم افزارهای رمزنگاری که از کلیدهای طولانی استفاده می‌کنند، قرار داده است تا از ایجاد کدهای مخفی توسط سازمان‌های بالقوه متخاصم جلوگیری کند که آژانس امنیت ملی ایالات متحده (NSA) خود قادر به شکستن کدهای مخفی نباشد.





کتاب عالی *Applied Cryptography* (John Wiley & Sons) برگردان Bruce Schneier است که زمان لازم برای شکستن رمز DES را با حدس زدن همه کلیدها، با استفاده از فناوری و اقتصاد ۱۹۹۵ توصیف می‌کند. گزینه‌هایی از این جدول در جدول زیر نشان داده شده است.

Attack cost	40-bit key	56-bit key	64-bit key	80-bit key	128-bit key
\$100,000	2 secs	35 hours	1 year	70,000 years	10^{19} years
\$1,000,000	200 msecs	3.5 hours	37 days	7,000 years	10^{18} years
\$10,000,000	20 msecs	21 mins	4 days	700 years	10^{17} years
\$100,000,000	2 msecs	2 mins	9 hours	70 years	10^{16} years
\$1,000,000,000	200 usecs	13 secs	1 hour	7 years	10^{15} years

با توجه به سرعت ریزپردازندگان ۱۹۹۵، مهاجمی که مایل به خرج کردن ۱۰۰۰۰ دلار در سال ۱۹۹۵ باشد می‌تواند یک کد DES ۴۰ بیتی را در حدود ۲ ثانیه بشکند. کامپیوترهای سال ۲۰۰۲ در حال حاضر ۲۰ برابر سریعتر از سال ۱۹۹۵ هستند. مگر اینکه کاربران به طور مکرر کلیدها را تغییر دهند. توجه داشته باشید که کلیدهای ۴۰ بیتی در برابر مخالفان با انگیزه ایمن نیستند.

اندازه کلید استاندارد DES ۵۶ بیتی امن‌تر است. در اقتصاد سال ۱۹۹۵، یک حمله ۱ میلیون دلاری هنوز چندین ساعت طول می‌کشد تا رمز را بشکند. اما شخصی که به ابرکامپیوترها دسترسی داشته باشد می‌تواند با استفاده از **Brute Force** در عرض چند ثانیه کد را بشکند.

در مقابل، اعتقاد بر این است که کلیدهای ۱۲۸ بیتی Triple-DES، که از نظر اندازه مشابه کلیدهای DES هستند، توسط هر کسی و به هر قیمتی با استفاده از یک حمله **Brute Force** غیرقابل شکستن هستند.

Establishing Shared Keys

یکی از معایب رمزنگاری‌های متقارن این است که فرستنده و گیرنده قبل از اینکه بتوانند با یکدیگر صحبت کنند باید یک کلید مخفی مشترک داشته باشند.

اگر می‌خواهید با فروشگاه سخت‌افزار Joe به طور امن صحبت کنید، شاید بعد از تماشای برنامه‌ای برای بهسازی خانه در تلویزیون عمومی، چند ابزار نجاری سفارش دهید، باید یک کلید مخفی خصوصی بین خود و www.joes-hardware.com ایجاد کنید. اگر می‌خواهید هر چیزی را با خیال راحت سفارش دهید شما به راهی برای تولید کلید مخفی و به خاطر سپردن آن نیاز دارید. هم شما و هم Joe's Hardware و هر کاربر اینترنتی دیگری هزاران کلید برای تولید و به خاطر سپردن دارید.

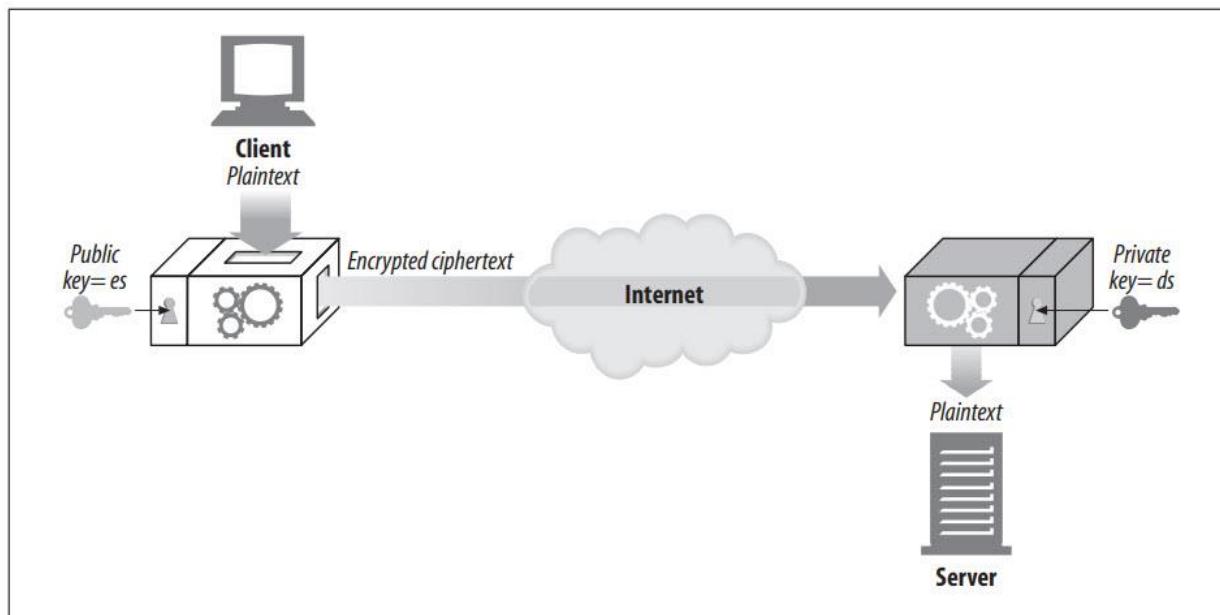




در نظر بگیرید که آلیس (A)، باب (B) و کریس (C) همگی می‌خواستند با (J) صحبت کنند. A، B و C هر کدام باید کلیدهای مخفی خود را با J ایجاد کنند. A به کلید kAJ و C به کلید kCJ و B به کلید kBJ نیاز دارد. هر جفت طرف در ارتباط به کلید خصوصی خود نیاز دارد. اگر N گره وجود داشته باشد و هر گره باید به طور ایمن با تمام گرههای N-1 دیگر صحبت کند، تقریباً N به توان ۲ کل کلیدهای مخفی وجود دارد: یک کابوس مدیریتی.

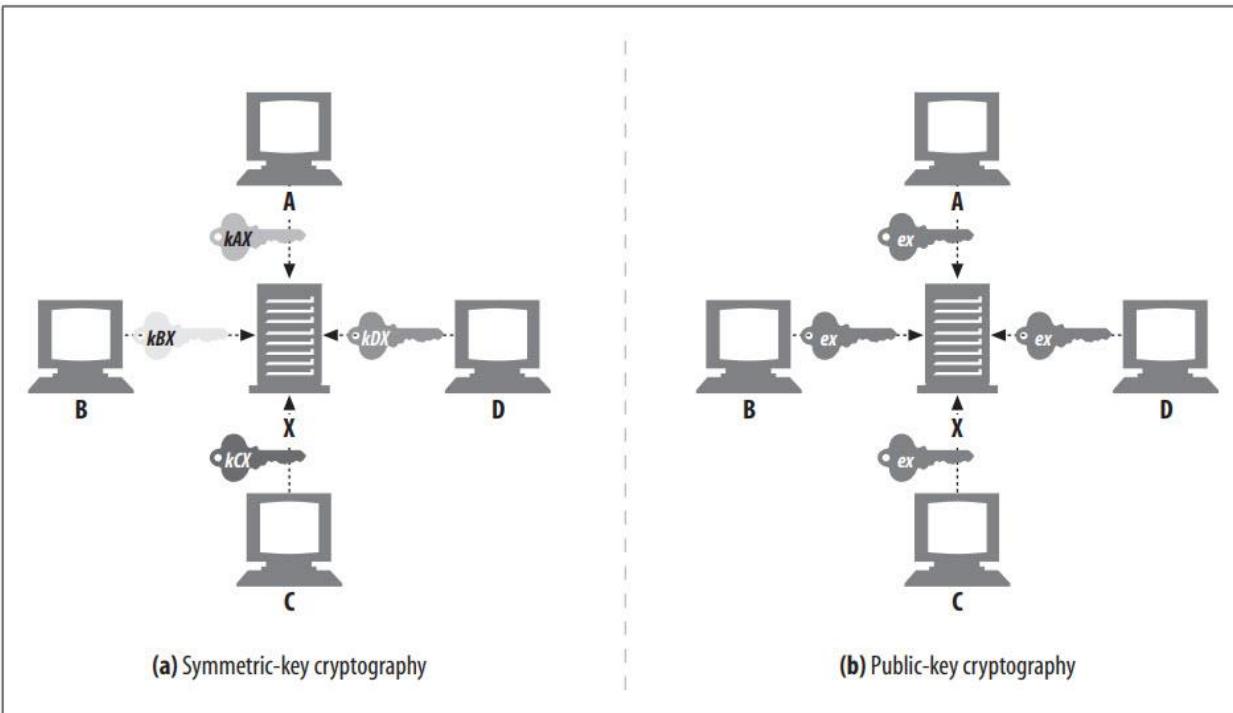
Public-Key Cryptography

به جای یک کلید رمزگذاری/رمزگشایی واحد برای هر جفت میزبان، رمزنگاری کلید عمومی از دو کلید نامتقارن استفاده می‌کند: یکی برای رمزگذاری پیامها برای یک میزبان و دیگری برای رمزگشایی پیامهای میزبان. کلید رمزگذاری به طور عمومی برای جهان شناخته شده است (بنابراین نام رمزنگاری کلید عمومی است)، اما تنها میزبان کلید رمزگشایی خصوصی را می‌داند (شکل زیر را ببینید).



این کار ایجاد کلید را بسیار آسان‌تر می‌کند، زیرا همه می‌توانند کلید عمومی را برای یک میزبان خاص پیدا کنند. اما کلید رمزگشایی مخفی نگه داشته می‌شود، بنابراین فقط گیرنده می‌تواند پیامهای ارسال شده به آن را رمزگشایی کند.





تصویر بالا را در نظر بگیرید. گره X می‌تواند کلید رمزگذاری ex خود را بگیرد و آن را به صورت عمومی منتشر کند. (همانطور که بعداً خواهیم دید، بیشتر جستجوی کلید عمومی در واقع از طریق گواهی‌های دیجیتال انجام می‌شود، اما جزئیات نحوه یافتن کلیدهای عمومی در حال حاضر اهمیت چندانی ندارد - فقط بدانید که آن‌ها در جایی برای عموم در دسترس هستند). اکنون هر کسی که بخواهد پیامی به گره X ارسال کند می‌تواند از همان کلید عمومی معروف استفاده کند.

حتی اگر همه می‌توانند پیام‌های X را با یک کلید رمزگذاری کنند، هیچ کس به جز X نمی‌تواند پیام‌ها را رمزگشایی نماید، زیرا فقط X دارای کلید خصوصی رمزگشایی dx است. تقسیم کلیدها به هر کسی اجازه می‌دهد پیامی را رمزگذاری کند، اما توانایی رمزگشایی پیام‌ها را فقط به مالک محدود می‌کند. این کار ارسال امن پیام‌ها به سرورها را برای گره‌ها آسان‌تر می‌کند، زیرا آن‌ها فقط می‌توانند کلید عمومی سرور را جستجو کنند.

فناوری رمزگذاری کلید عمومی امکان استقرار پروتکلهای امنیتی را برای هر کاربر رایانه در سراسر جهان فراهم می‌کند. به دلیل اهمیت زیاد ساخت یک مجموعه فناوری کلید عمومی استاندارد، یک ابتکار عظیم استانداردهای زیرساخت کلید عمومی (PKI) برای بیش از یک دهه در حال انجام است.

RSA

چالش هر سیستم رمزنگاری نامتقارن با کلید عمومی این است که مطمئن شود هیچ فرد بدخواهی نمی‌تواند کلید خصوصی و مخفی را محاسبه کند - حتی اگر همه سرنخ‌های زیر را داشته باشد:





- کلید عمومی (که هر کسی می‌تواند آن را دریافت کند، زیرا عمومی است)
- یک تکه از متن رمز رهگیری شده (به دست آمده با رديابي شبکه)
- یک پیام و متن رمزی مرتبط با آن (با اجرای Encoder بر روی هر متنی به دست می‌آید)

یکی از محبوب‌ترین سیستم‌های رمزنگاری کلید عمومی که تمام این نیازها را برآورده می‌کند، الگوریتم RSA است که در MIT اختراع شد و متعاقباً توسط RSA Data Security تجاری شد. با توجه به یک کلید عمومی، یک قطعه متن ساده دلخواه، متن رمزی مرتبط از رمزنگاری متن ساده با کلید عمومی، خود الگوریتم RSA و حتی کد منبع اجرای RSA، شکستن کد برای یافتن کلید خصوصی مربوطه مشکلی به اندازه محاسبه اعداد اول بزرگ می‌باشد. اعتقاد بر این است که یکی از سخت‌ترین مسائل در تمام علوم کامپیوتر همین موضوع محاسبه اعداد اول است. بنابراین، اگر بتوانید راهی سریع برای تبدیل اعداد بزرگ به اعداد اول پیدا کنید، نه تنها می‌توانید به حساب‌های بانکی سوئیس نفوذ کنید، بلکه می‌توانید جایزه تورینگ را نیز ببرید.

جزئیات رمزنگاری RSA شامل برخی از ریاضیات پیچیده است، بنابراین ما در اینجا به آن‌ها نخواهیم پرداخت. کتابخانه‌های زیادی وجود دارد که به شما امکان می‌دهد الگوریتم‌های RSA را بدون نیاز به مدرک دکترا در نظریه اعداد انجام دهید.

Hybrid Cryptosystems and Session Keys

رمزنگاری نامتقارن با کلید عمومی بسیار خوب است، زیرا هر کسی می‌تواند پیام‌های ایمن را فقط با دانستن کلید عمومی آن به سرور عمومی ارسال کند. دو گره برای برقراری ارتباط ایمن، ابتدا نیازی به مذکوره با یک کلید خصوصی ندارند.

اما الگوریتم‌های رمزنگاری کلید عمومی از نظر محاسباتی کند هستند. در عمل، مخلوطی از هر دو طرح متقارن و نامتقارن استفاده می‌شود. به عنوان مثال، استفاده از رمزنگاری کلید عمومی برای برقراری ارتباط امن بین گره‌ها معمول است، اما سپس از آن کانال امن برای تولید و برقراری ارتباط یک کلید متقارن موقت و تصادفی برای رمزنگاری بقیه داده‌ها از طریق رمزنگاری متقارن سریع‌تر استفاده می‌شود.

Digital Signatures

تا کنون، ما در مورد انواع رمزهای کلیددار صحبت کردایم که از کلیدهای متقارن و نامتقارن استفاده می‌کنند تا به ما امکان رمزنگاری و رمزگشایی پیام‌های مخفی را بدهند.





علاوه بر رمزگذاری و رمزگشایی پیامها، از سیستم‌های رمزگاری می‌توان برای امضای پیام‌ها استفاده کرد و ثابت کرد که چه کسی پیام را نوشته و ثابت می‌کند پیام دستکاری نشده است. این تکنیک که امضای دیجیتال نامیده می‌شود برای گواهینامه‌های امنیت اینترنت مهم است که در قسمت بعدی به آن می‌پردازیم.

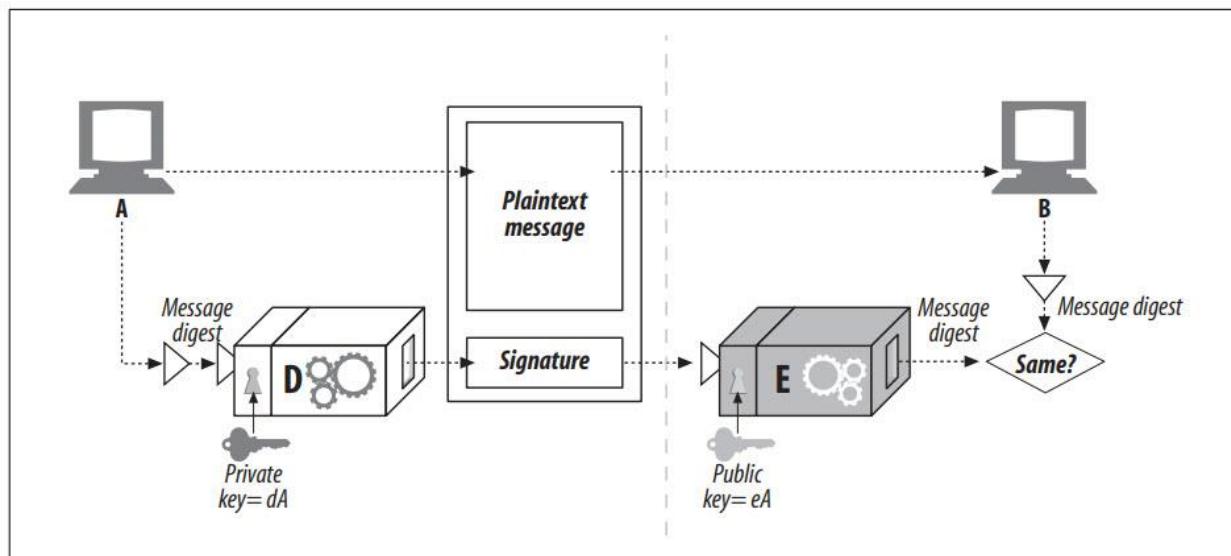
Signatures Are Cryptographic Checksums

امضای دیجیتال Checksum‌های رمزگاری ویژه‌ای هستند که به یک پیام متصل می‌شوند. آن‌ها دو فایده دارند:

- امضاهای ثابت می‌کنند که نویسنده، پیام را نوشته است. از آنجایی که فقط نویسنده دارای کلید خصوصی فوق محروم‌انه نویسنده است، فقط نویسنده می‌تواند این Checksum‌ها را محاسبه کند.
- به عنوان یک "امضای" شخصی نویسنده عمل می‌کند.
- امضاهای از دستکاری پیام جلوگیری می‌کنند. اگر یک مهاجم مخرب پیام را در حین ارسال تغییر دهد، دیگر مطابقت نخواهد داشت و از آنجایی که Checksum شامل کلید خصوصی و مخفی نویسنده است، مهاجم قادر نخواهد بود یک Checksum صحیح برای پیام دستکاری شده بسازد.

امضاهای دیجیتال اغلب با استفاده از فناوری نامتقارن و کلید عمومی تولید می‌شوند. کلید خصوصی نویسنده به عنوان نوعی "اثر انگشت" استفاده می‌شود، زیرا کلید خصوصی فقط توسط مالک شناخته می‌شود.

شکل زیر مثالی را نشان می‌دهد که چگونه گره A می‌تواند پیامی را به گره B ارسال کند و آن را امضا کند:



- گره A پیام با طول متغیر را در یک Digest با اندازه ثابت قرار می‌دهد.
- گره A یک تابع "امضا" را برای Digest اعمال می‌کند که از کلید خصوصی کاربر به عنوان پارامتر استفاده می‌کند. از آنجا که فقط کاربر کلید خصوصی را می‌داند، یک تابع امضای صحیح





نشان می دهد که امضاکننده مالک است. در شکل بالا، ما از تابع رمزگشا D به عنوان تابع امضا استفاده می کنیم، زیرا شامل کلید خصوصی کاربر می شود.

هنگامی که امضا محاسبه شد، گره A آن را به انتهای پیام اضافه می کند و هم پیام و هم امضا را به گره B می فرستد.

در صورت دریافت، اگر گره B بخواهد مطمئن شود که گره A واقعاً پیام را نوشته است و پیام دستکاری نشده است، گره B می تواند امضا را بررسی کند. گره B امضا درهم شده با کلید خصوصی را می گیرد و Tابع معکوس را با استفاده از کلید عمومی اعمال می کند. اگر Digest بدون بسته بندی با نسخه خود گره B مطابقت نداشته باشد، یا پیام در حین ارسال دستکاری شده یا فرستنده کلید خصوصی گره A را نداشته است (و بنابراین گره A نبوده است).

Digital Certificates

در این بخش، در مورد گواهی های دیجیتال، «کارت های شناسایی» اینترنت صحبت می کنیم. گواهی های دیجیتال (که اغلب «certs» نامیده می شوند) حاوی اطلاعاتی درباره یک کاربر یا شرکتی است که توسط یک سازمان مورد اعتماد تضمین شده است.

همه ما انواع مختلفی از هویت را حمل می کنیم. برخی از شناسنامه ها، مانند گذرنامه و گواهینامه رانندگی، به اندازه کافی برای اثبات هویت در بسیاری از موقعیت ها مورد اعتماد هستند. به عنوان مثال، گواهینامه رانندگی ایالات متحده مدرک کافی برای اثبات هویت است که به شما اجازه می دهد برای شب سال نو سوار هواپیما به نیویورک شوید و این مدرک کافی برای سن شما است.

اشکال قابل اعتمادتر شناسایی، مانند گذرنامه، توسط دولت بر روی کاغذ مخصوص امضا و مهر می شود. جعل آنها سخت تر است، بنابراین ذاتاً سطح بالاتری از اعتماد را دارند. برخی از نشان ها و کارت های هوشمند شرکتی شامل لوازم الکترونیکی برای کمک به تقویت هویت شرکت مخابراتی هستند. برخی از سازمان های دولتی فوق محramانه حتی باید قبل از اعتماد به شناسه شما، اثر انگشت یا الگوهای مویرگی شبکیه شما را با شناسه شما مطابقت دهند! سایر اشکال شناسنامه، مانند کارت ویزیت، امکن جعل نسبتاً آسان تری دارند، بنابراین مردم کمتر به این اطلاعات اعتماد دارند. آنها ممکن است برای تعاملات حرفه ای خوب باشند، اما احتمالاً هنگام درخواست وام مسکن، مدرک کافی برای اشتغال ندارند.



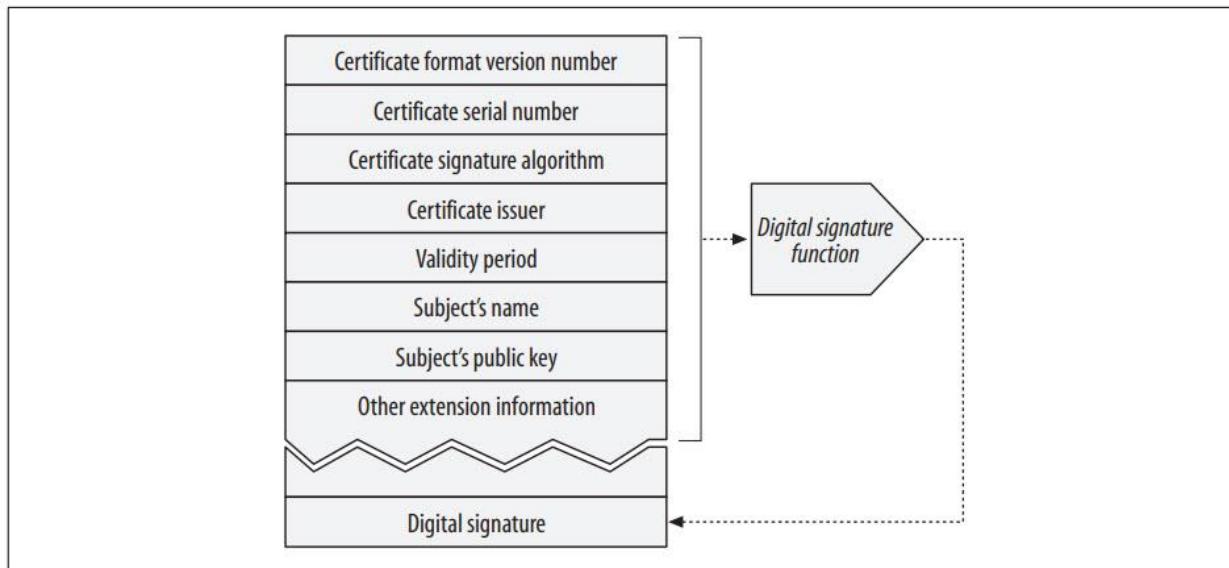


The Guts of a Certificate

گواهی‌های دیجیتال همچنین حاوی مجموعه‌ای از اطلاعات هستند که همه آن‌ها به صورت دیجیتالی توسط مرجع صدور گواهی یا Certificate Authority رسمی امضا شده‌اند. گواهی‌های دیجیتال پایه معمولاً شامل موارد اساسی مشترک برای شناسه‌های چاپی هستند، مانند:

- Subject's name (person, server, organization, etc.)
- Expiration date
- Certificate issuer (who is vouching for the certificate)
- Digital signature from the certificate issuer

علاوه بر این، گواهی‌های دیجیتال اغلب حاوی کلید عمومی Subject و همچنین اطلاعات توصیفی درباره موضوع و الگوریتم امضای مورد استفاده هستند. هر کسی می‌تواند یک گواهی دیجیتال ایجاد کند، اما همه نمی‌توانند از یک مرجع معتبر امضاکننده برای تضمین اطلاعات گواهی و امضای گواهی با کلید خصوصی آن استفاده کنند. یک ساختار گواهی معمولی در شکل زیر نشان داده شده است.



X.509 v3 Certificates

متأسفانه، هیچ استاندارد واحد و جهانی برای گواهی‌های دیجیتال وجود ندارد. انواع مختلفی از گواهی‌های دیجیتال وجود دارد، همانطور که همه کارت‌های شناسایی چاپ شده حاوی اطلاعات یکسانی در یک مکان نیستند. خبر خوب این است که اکثر گواهینامه‌هایی که امروزه مورد استفاده قرار می‌گیرند اطلاعات خود را به شکل استانداردی به نام X.509 v3 ذخیره می‌کنند. گواهینامه‌های X.509 v3 یک روش استاندارد برای ساختاردهی اطلاعات گواهی در فیلدهای قابل تجزیه ارائه می‌کنند. انواع مختلف گواهی‌ها دارای مقادیر





فیلد متفاوتی هستند، اما اکثر آن‌ها از ساختار X.509 v3 پیروی می‌کنند. فیلدهای یک گواهی X.509 در جدول زیر توضیح داده شده است.

Field	Description
Version	The X.509 certificate version number for this certificate. Usually version 3 today.
Serial Number	A unique integer generated by the certification authority. Each certificate from a CA must have a unique serial number.
Signature Algorithm ID	The cryptographic algorithm used for the signature. For example, "MD2 digest with RSA encryption".
Certificate Issuer	The name for the organization that issued and signed this certificate, in X.500 format.
Validity Period	When this certificate is valid, defined by a start date and an end date.
Subject's Name	The entity described in the certificate, such as a person or an organization. The subject name is in X.500 format.
Subject's Public Key Information	The public key for the certificate's subject, the algorithm used for the public key, and any additional parameters.
Issuer Unique ID (optional)	An optional unique identifier for the certificate issuer, to allow the potential reuse of the same issuer name.
Subject Unique ID (optional)	An optional unique identifier for the certificate subject, to allow the potential reuse of the same subject name.
Extensions	An optional set of extension fields (in version 3 and higher). Each extension field is flagged as critical or noncritical. Critical extensions are important and must be understood by the certificate user. If a certificate user doesn't recognize a critical extension field, it must reject the certificate. Common extension fields in use include: <i>Basic Constraints</i> Subject's relationship to certification authority <i>Certificate Policy</i> The policy under which the certificate is granted <i>Key Usage</i> Restricts how the public key can be used
Certification Authority Signature	The certification authority's digital signature of all of the above fields, using the specified signing algorithm.

انواع مختلفی از گواهی‌های مبتنی بر X.509 وجود دارد، از جمله گواهی‌های وب سرور، گواهی‌های ایمیل کلاینت، گواهی‌های امضای کد نرم‌افزار و گواهی‌های Certificate Authority.



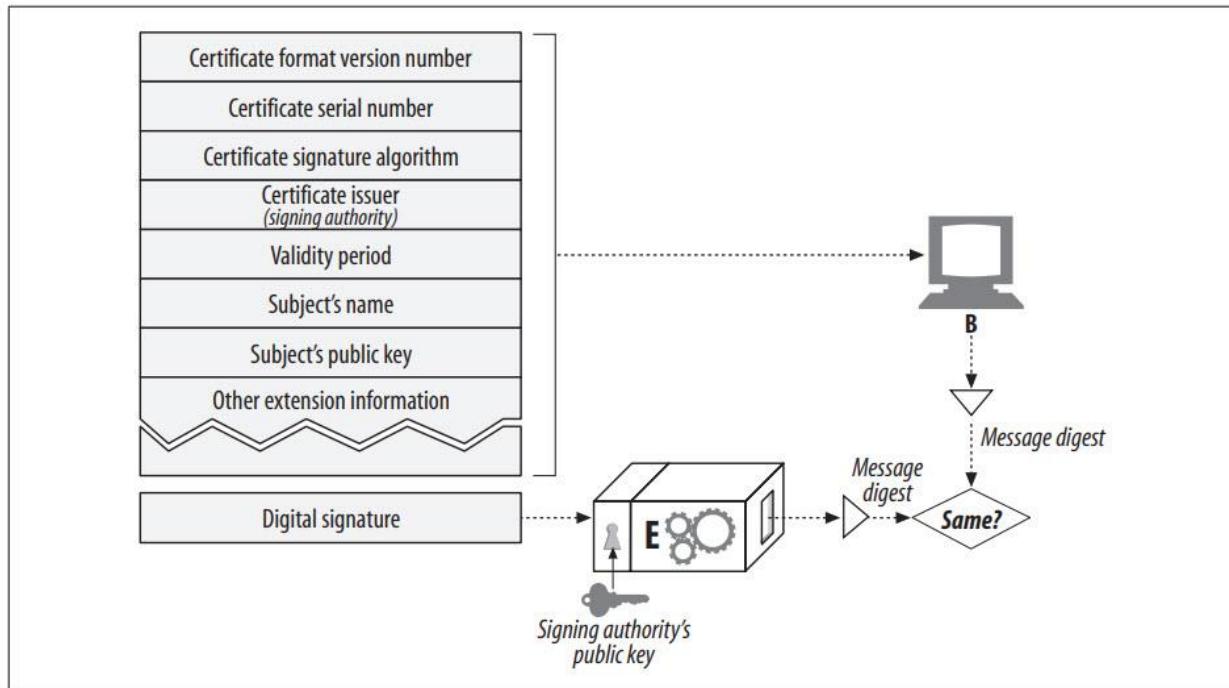


Using Certificates to Authenticate Servers

هنگامی که یک تراکنش وب ایمن از طریق HTTPS ایجاد می‌کنید، مرورگرهای مدرن به طور خودکار گواهی دیجیتال را برای سروری که به آن متصل است واکشی می‌کنند. اگر سرور گواهینامه نداشته باشد، اتصال ایمن با شکست مواجه می‌شود. گواهی سرور حاوی فیلدهای زیادی است، از جمله:

- Name and hostname of the web site
- Public key of the web site
- Name of the signing authority
- Signature from the signing authority

وقتی مرورگر گواهی را دریافت می‌کند، مرجع امضاکننده را بررسی می‌کند. اگر یک مرجع امضاکننده عمومی و معتر باشد، مرورگر از قبل کلید عمومی خود را می‌داند (مرورگرها با گواهی‌های بسیاری از مقامات امضاکننده از قبل نصب شده نصب می‌شوند)، بنابراین می‌تواند امضا را همانطور که در بخش قبلی، «امضای دیجیتالی» بحث کردیم، تأیید کند. شکل زیر نشان می‌دهد که چگونه یکپارچگی یک گواهی با استفاده از امضا دیجیتال آن تأیید می‌شود.



اگر مرجع امضاکننده ناشناخته باشد، مرورگر مطمئن نیست که باید به مرجع امضاکننده اعتماد کند یا خیر و معمولاً یک کادر محاوره‌ای را برای کاربر نمایش می‌دهد تا آن را بخواند و ببیند آیا به امضاکننده اعتماد دارد یا خیر. امضاکننده ممکن است بخش فناوری اطلاعات محلی یا یک فروشنده نرم افزار باشد.





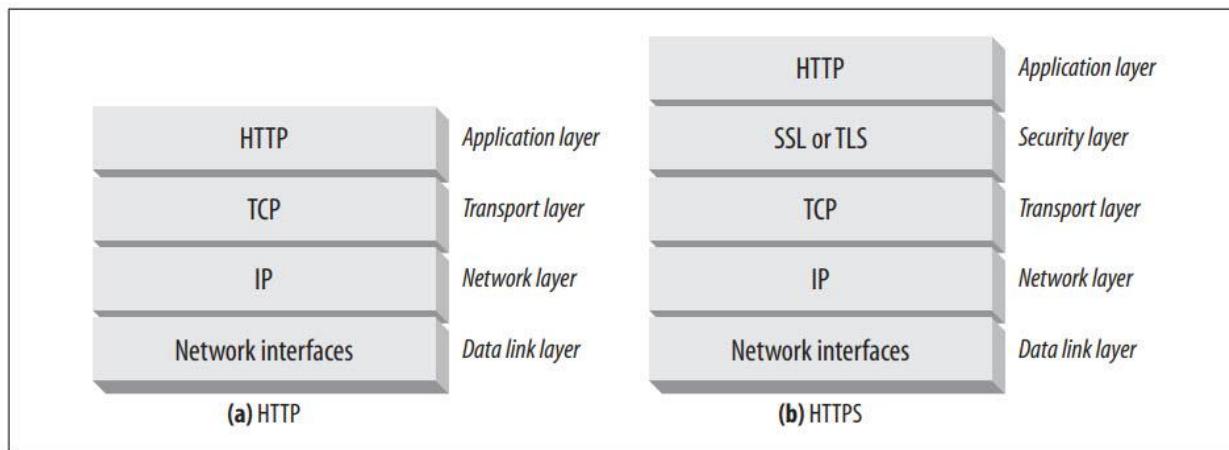
HTTPS: The Details

HTTPS محبوب‌ترین نسخه امن HTTP است. این پروتکل به طور گسترده در تمام مرورگرها و سرورهای تجاری اصلی پیاده سازی و در دسترس است. HTTPS پروتکل HTTP را با مجموعه‌ای قدرتمند از تکنیک‌های رمزگاری مقارن، نامتقارن و مبتنی بر گواهی ترکیب می‌کند، که HTTPS را بسیار امن می‌کند، اما همچنین بسیار انعطاف‌پذیر و آسان برای مدیریت در سراسر اینترنت است.

رشد برنامه‌های کاربردی اینترنتی را سرعت بخشیده است و نیروی اصلی در رشد سریع تجارت الکترونیک مبتنی بر وب بوده است. HTTPS همچنین در مدیریت گسترده و ایمن برنامه‌های کاربردی وب توزیع شده بسیار مهم بوده است.

HTTPS Overview

همان HTTPS است که از طریق یک لایه انتقال امن ارسال می‌شود. HTTPS به جای ارسال پیام‌های HTTP رمزگذاری نشده به TCP و در سراسر اینترنت (بخش a شکل زیر)، پیام‌های HTTP را ابتدا به یک لایه امنیتی ارسال می‌کند که آن‌ها را قبل از ارسال به TCP رمزگذاری می‌کند (بخش b شکل زیر).



امروزه لایه امنیتی SSL و جایگزین مدرن آن یعنی TLS پیاده سازی می‌شود. ما از روش رایج استفاده از اصطلاح "SSL" به معنای SSL یا TLS پیروی می‌کنیم.

HTTPS Schemes

امروزه، HTTP امن اختیاری است. بنابراین، هنگام درخواست از یک وب سرور، به راهی نیاز داریم که به وب سرور بگوییم نسخه پروتکل امن HTTP را انجام دهد. این در طرح URL انجام می‌شود.

در HTTP معمولی و غیر ایمن، پیشوند طرح URL `http` است، مانند:





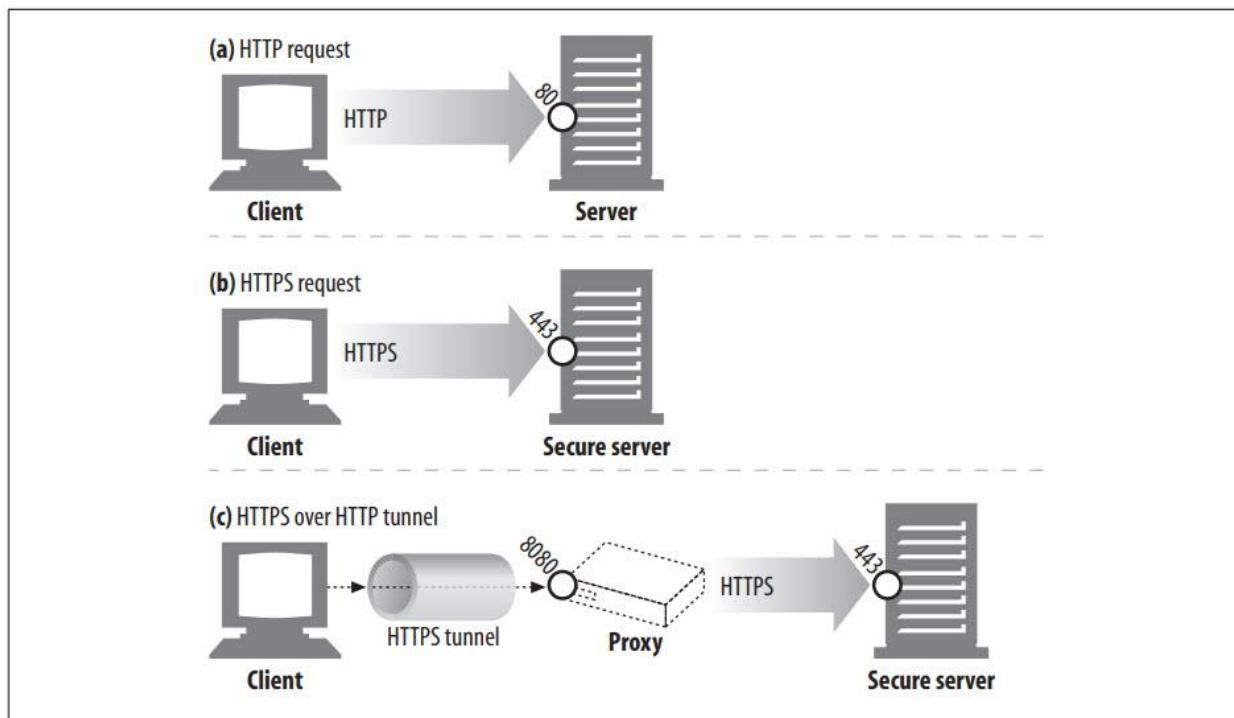
<http://www.joes-hardware.com/index.html>

در پروتکل امن HTTPS، پیشوند طرح https URL است، مانند:

https://cajun-shop.securesites.com/Merchant2/merchant.mv?Store_Code=AGCGS

هنگامی که از یک کلاینت (مانند یک مرورگر وب) خواسته می‌شود تا تراکنشی را روی یک منبع وب انجام دهد، طرح URL را بررسی می‌کند:

- اگر URL دارای طرح http باشد، کلاینت یک اتصال به سرور در پورت ۸۰ (به طور پیش فرض) باز می‌کند و دستورات HTTP قدیمی را برای آن ارسال می‌کند (بخش a شکل زیر).
- اگر URL دارای طرح https باشد، کلاینت یک اتصال به سرور در پورت ۴۴۳ (به طور پیش فرض) باز می‌کند و سپس با سرور «Handshake می‌کند»، برخی از پارامترهای امنیتی SSL را با سرور در فرمت باینری مبادله می‌کند و سپس دستورات HTTP رمزگذاری شده را دنبال می‌کند. (بخش b شکل زیر).



از آنجایی که ترافیک SSL یک پروتکل باینری است که کاملاً با HTTP متفاوت است، ترافیک در پورت‌های مختلف انجام می‌شود (SSL معمولاً روی پورت ۴۴۳ حمل می‌شود). اگر ترافیک SSL و HTTP هر دو به پورت ۸۰ می‌رسید، اکثر وب سرورها ترافیک SSL باینری را به عنوان HTTP اشتباه تفسیر می‌کردند و اتصال را

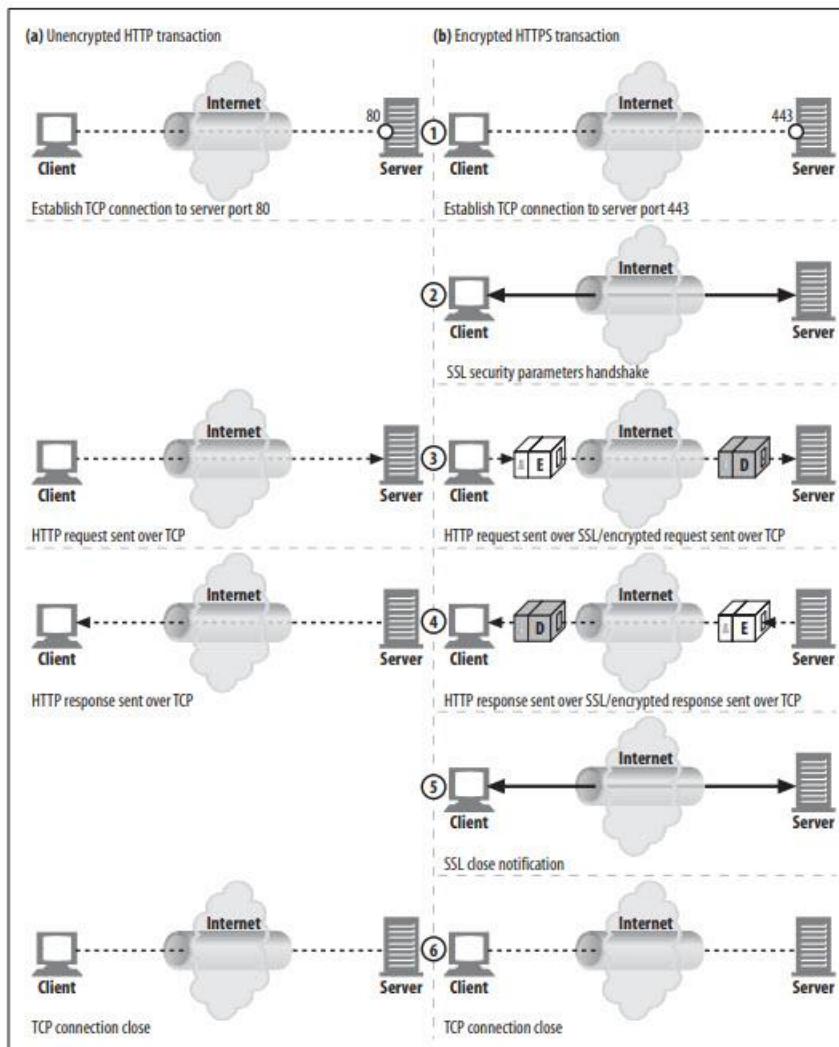


می بستند. لایه‌بندی یکپارچه‌تر سرویس‌های امنیتی در HTTP نیاز به پورت‌های مقصد متعدد را از بین می‌برد، اما در عمل مشکلات جدی ایجاد نمی‌کند.

بیایید کمی دقیق‌تر به نحوه تنظیم اتصالات SSL با سرورهای امن نگاه کنیم.

Secure Transport Setup

در HTTP رمزگذاری نشده، یک کلاینت یک اتصال TCP را به پورت ۸۰ روی سرور وب باز می‌کند، یک پیام درخواست ارسال می‌کند، یک پیام پاسخ دریافت می‌کند و اتصال را می‌بندد. این دنباله در بخش a شکل زیر ترسیم شده است.



این روش در HTTPS به دلیل لایه امنیتی SSL کمی پیچیده‌تر است. در HTTPS، کلاینت ابتدا یک اتصال به پورت ۴۴۳ (درگاه پیش‌فرض برای HTTP اینم) در سرور وب باز می‌کند. هنگامی که اتصال



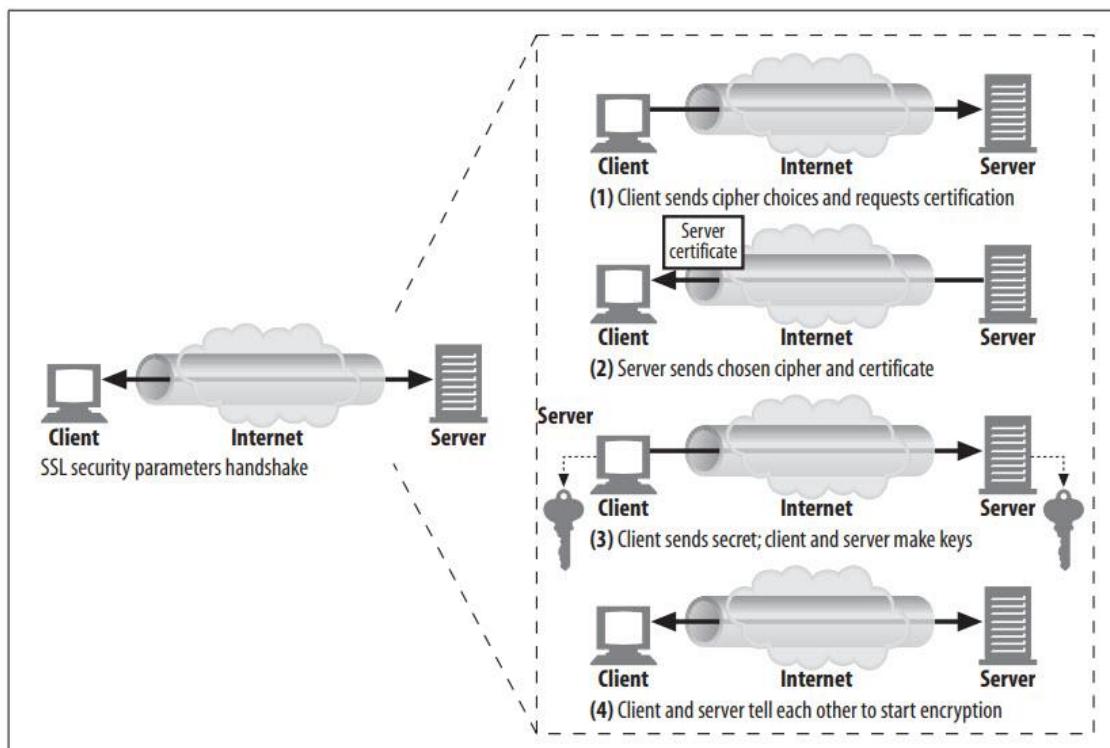
TCP برقرار شد، سرویس گیرنده و سرور لایه SSL را مقداردهی اولیه می‌کنند و در مورد پارامترهای رمزگاری مذاکره نموده و کلیدها را مبادله می‌کنند. هنگامی که دست دادن کامل شد، مقداردهی اولیه SSL انجام می‌شود و کلاینت می‌تواند پیام‌های درخواستی را به لایه امنیتی ارسال کند. این پیام‌ها قبل از ارسال به TCP رمزگذاری می‌شوند. این روش در بخش b شکل بالا نشان داده شده است.

SSL Handshake

قبل از اینکه بتوانید پیام‌های HTTP رمزگذاری شده ارسال کنید، سرویس گیرنده و سرور باید یک Handshake انجام دهند:

- تبادل شماره نسخه پروتکل
- رمزی را انتخاب کنید که هر طرف بداند.
- هویت هر طرف را تأیید کنید.
- کلیدهای جلسه موقت را برای رمزگذاری کانال ایجاد کنید.

قبل از اینکه هر داده HTTP رمزگذاری شده در سراسر شبکه پخش شود، SSL قبلاً یک دسته از داده‌های Handshake را برای برقراری ارتباط ارسال کرده است. ماهیت SSL Handshake در شکل زیر نشان داده شده است.





این یک نسخه ساده شده از SSL Handshake است. بسته به نحوه استفاده از SSL می‌تواند پیچیده‌تر باشد، اما این ایده کلی است.

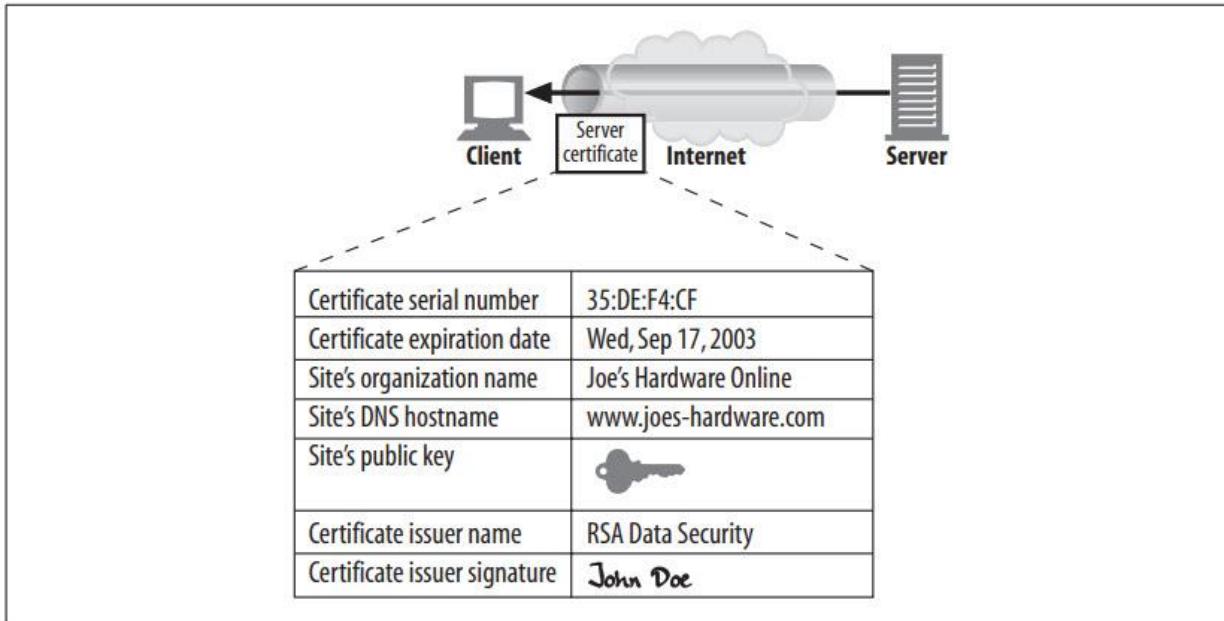
Server Certificates

SSL از احراز هویت متقابل پشتیبانی می‌کند، گواهی‌نامه‌های سرور را به کلاینت‌ها حمل می‌کند و گواهی‌نامه‌های کلاینت را به سرورها باز می‌گرداند. اما امروزه، گواهی کلاینت‌ها معمولاً برای مرور استفاده نمی‌شود. اغلب کاربران حتی گواهی کلاینت شخصی ندارند. یک سرور وب می‌تواند برای دریافت گواهی درخواست نماید، اما این امر به ندرت در عمل رخ می‌دهد.

از سوی دیگر، تراکنش‌های امن HTTPS همیشه نیاز به گواهی سرور دارند. زمانی که شما یک تراکنش ایمن را بر روی یک سرور وب انجام می‌دهید، مانند ارسال اطلاعات کارت اعتباری خود، شما می‌خواهید بدانید که با سازمانی که فکر می‌کنید با آن صحبت می‌کنید، صحبت می‌کنید. گواهی‌های سرور، که توسط یک مقام شناخته‌شده امضا شده‌اند، به شما کمک می‌کنند پیش از ارسال کارت اعتباری یا اطلاعات شخصی خود، میزان اعتماد خود را به سرور ارزیابی کنید.

گواهی سرور یک گواهی مشتق شده از X.509v3 است که نام سازمان، آدرس، نام دامنه سرور DNS و سایر اطلاعات را نشان می‌دهد (شکل زیر را ببینید). شما و نرمافزار کلاینت‌تان می‌توانید گواهی را بررسی کنید تا مطمئن شوید که همه چیز در حال بهروزرسانی است.





Site Certificate Validation

در خود SSL نیازی نیست که شما گواهی سرور وب را بررسی کنید، اما اکثر مرورگرهای مدرن چکهای منطقی ساده‌ای بر روی گواهی‌ها انجام می‌دهند و ابزار انجام چکهای کامل‌تر را در اختیار شما قرار می‌دهند. یک الگوریتم برای تایید گواهی سرور وب، پیشنهاد شده توسط نتسکیپ، اساس اکثر تکنیک‌های تایید مرورگر را تشکیل می‌دهد. مراحل عبارتند از:

Data Check

اول، مرورگر تاریخ شروع و پایان گواهی را بررسی می‌کند تا اطمینان حاصل شود که گواهی هنوز معتبر است. اگر گواهی منقضی شده یا هنوز فعال نشده باشد، اعتبار گواهی با شکست مواجه می‌شود و مرورگر خطای را نشان می‌دهد.

Signer trust check

هر گواهی توسط یک مرجع صدور گواهی یا همان CA که ضامن سرور است، امضا می‌شود. سطوح مختلفی از گواهی وجود دارد که هر کدام نیازمند سطوح متفاوتی از تایید پس‌زمینه هستند. به عنوان مثال، اگر برای گواهی سرور تجارت الکترونیک درخواست می‌کنید، معمولاً باید مدرک قانونی ثبت نام به عنوان یک تجارت را ارائه دهید.

هر کسی می‌تواند گواهی تولید کند، اما برخی از CA سازمان‌های معروفی هستند که رویه‌های کاملاً درک شده برای تأیید هویت و رفتار تجاری خوب متقاضیان گواهی را دارند. به همین





دلیل، مرورگرها فهرستی از مقامات امضاکننده مورد اعتماد را ارسال می کنند. اگر یک مرورگر گواهی امضا شده توسط مقامات ناشناخته (و احتمالاً مخرب) دریافت کند، مرورگر معمولاً یک هشدار نمایش می دهد. مرورگرها همچنین ممکن است هر گواهینامه‌ای را با مسیر امضای معتبر به یک CA قابل اعتماد بپذیرند. به عبارت دیگر، اگر یک CA مورد اعتماد گواهینامه‌ای را برای Sam's Signing Shop یک گواهی سایت را امضا کند، مرورگر ممکن است گواهی را به عنوان منشاء از یک مسیر CA معتبر بپذیرد.

Signature check

هنگامی که مرجع امضا به عنوان قابل اعتماد در نظر گرفته می شود، مرورگر یکپارچگی گواهی را با استفاده از کلید عمومی مرجع امضا برای امضا و مقایسه آن با مبلغ چک بررسی می کند.

Site identity check

اغلب مرورگرها برای جلوگیری از کپی کردن گواهی شخص دیگری یا رهگیری ترافیک خود، سعی می کنند تایید کنند که نام دامنه در گواهی با نام دامنه سرور که با آن صحبت کرده‌اند مطابقت دارد. گواهی‌های سرور معمولاً شامل یک نام دامنه واحد هستند، اما برخی از CA ها گواهی‌هایی را ایجاد می کنند که حاوی فهرست‌هایی از نام سرور یا نام دامنه‌های با حروف عام است، برای کلاسترها یا مجموعه سرورها. اگر نام میزبان با هویت موجود در گواهی مطابقت نداشته باشد، سرویس گیرنده‌گان کاربر محور باید یا به کاربر اطلاع دهند یا اتصال را با یک خطای گواهی بد خاتمه دهند.

Virtual Hosting and Certificates

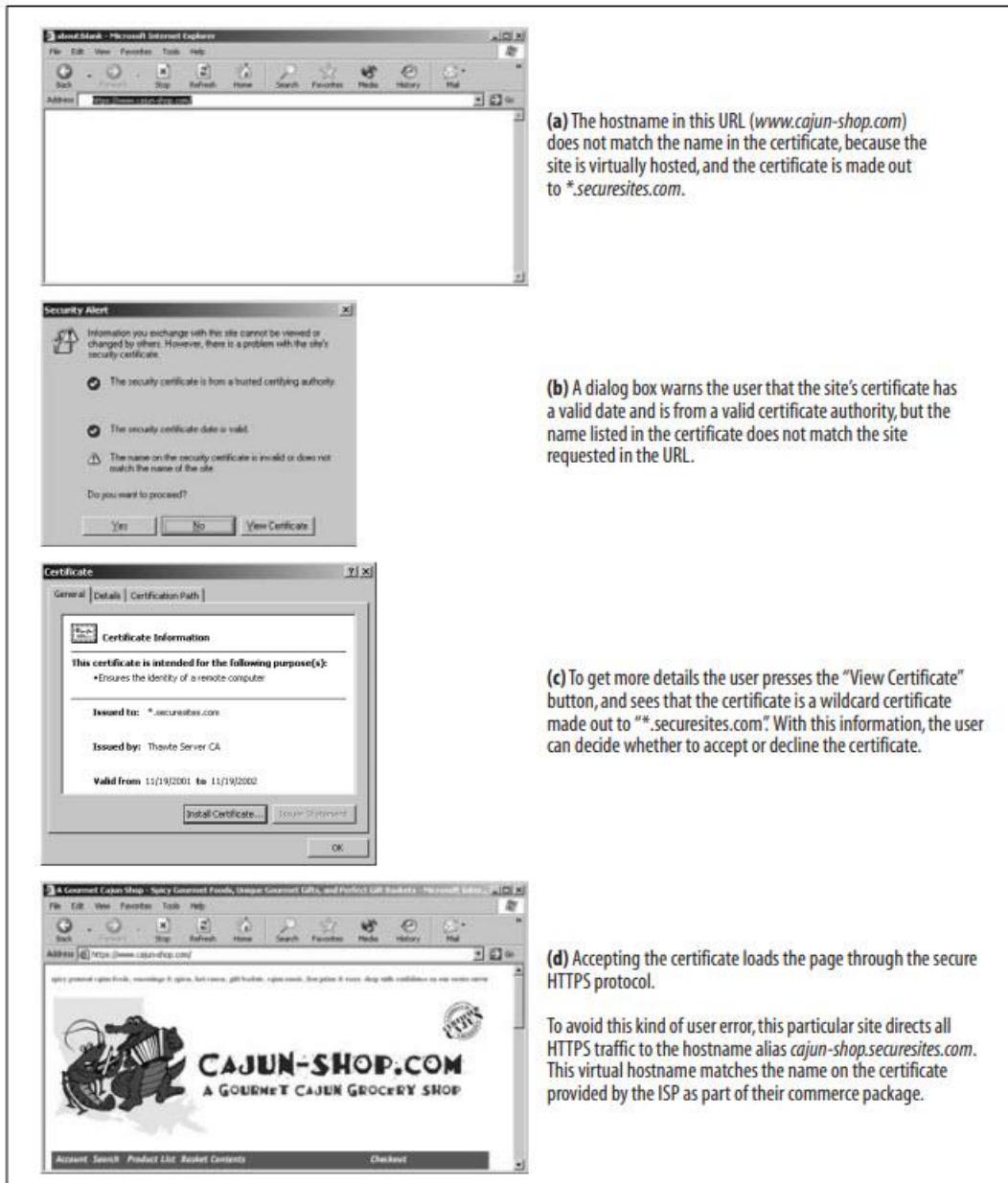
گاهی اوقات رسیدگی به ترافیک ایمن در سایت‌هایی که به صورت مجازی میزبانی می‌شوند دشوار است (چند نام میزبان در یک سرور). برخی از برنامه‌های وب سرور محبوب فقط از یک گواهی پشتیبانی می کنند. اگر کاربری برای نام میزبان مجازی که کاملاً با نام گواهی مطابقت ندارد وارد شود، یک کادر هشدار نمایش داده می شود.

به عنوان مثال، سایت تجارت الکترونیکی Cajun-Shop.com را در نظر بگیرید. ارائه دهنده میزبانی سایت نام رسمی <https://www.cajun-shop.securesites.com> را ارائه کرده است. وقتی کاربران به [shop.com](http://www.cajun-shop.com) مراجعه می کنند، نام میزبان رسمی فهرست شده در گواهی سرور (www.cajun-shop.com) با نام [*.securesites.com](http://www.cajun-shop.securesites.com) مطابقت ندارد و در این حالت هشدار مشابه شکل زیر ظاهر می شود.





برای جلوگیری از این مشکل، صاحبان Cajun-Shop.com همه کاربران را هنگام شروع تراکنش‌های امن به هدایت می‌کنند. مدیریت گواهی برای سایتها میزبان مجازی می‌تواند کمی مشکل باشد.



(a) The hostname in this URL (www.cajun-shop.com) does not match the name in the certificate, because the site is virtually hosted, and the certificate is made out to *.securesites.com.

(b) A dialog box warns the user that the site's certificate has a valid date and is from a valid certificate authority, but the name listed in the certificate does not match the site requested in the URL.

(c) To get more details the user presses the "View Certificate" button, and sees that the certificate is a wildcard certificate made out to "*.securesites.com". With this information, the user can decide whether to accept or decline the certificate.

(d) Accepting the certificate loads the page through the secure HTTPS protocol.

To avoid this kind of user error, this particular site directs all HTTPS traffic to the hostname alias cajun-shop.securesites.com. This virtual hostname matches the name on the certificate provided by the ISP as part of their commerce package.





A Real HTTPS Client

SSL یک پروتکل باینری پیچیده است. مگر اینکه متخصص رمزنگاری باشد، نباید مستقیماً ترافیک خام SSL را ارسال کنید. خوشبختانه چندین کتابخانه تجاری و منبع باز وجود دارد تا برنامه نویسی کلاینتها و سرورهای SSL را آسان‌تر کند.

OpenSSL

OpenSSL محبوب‌ترین پیاده سازی متن باز SSL و TLS است. پروژه OpenSSL یک تلاش داوطلبانه مشترک برای توسعه یک جعبه ابزار قوی، تجاری و با امکانات کامل است که پروتکل‌های SSL و TLS را پیاده‌سازی می‌کند و همچنین یک کتابخانه رمزنگاری همه‌منظوره با قدرت کامل است. می‌توانید اطلاعاتی درباره OpenSSL دریافت کرده و نرم افزار را از <http://www.openssl.org> دانلود کنید.

همچنین ممکن است نام SSLeay (تلفظ S-S-L-e-a-y) OpenSSL جانشین کتابخانه Eric A. Young (the “eay” of SSLeay) در اصل توسط SSLeay را بشنوید. است و رابط کاربری بسیار مشابهی دارد. توسعه داده شد.

A Simple HTTPS Client

در این بخش، از بسته OpenSSL برای نوشتن یک کلاینت HTTPS بسیار ابتدایی استفاده می‌کنیم. این سرویس گیرنده، یک اتصال SSL با یک سرور برقرار می‌کند، برخی از اطلاعات شناسایی را از سرور سایت چاپ می‌کند، یک درخواست HTTP GET را در کانال امن ارسال می‌کند، یک پاسخ HTTP را دریافت می‌کند و پاسخ را چاپ می‌کند.

برنامه C نشان داده شده در زیر یک اجرای OpenSSL از سرویس گیرنده HTTPS ساده است. برای ساده نگه داشتن برنامه، منطق مدیریت خطأ و پردازش گواهی گنجانده نشده است.

از آنجایی که مدیریت خطأ از این برنامه نمونه حذف شده است، باید از آن فقط برای مقدار توضیحی استفاده کنید. نرم افزار در شرایط خطای عادی از کار می‌افتد یا در غیر این صورت بد رفتار می‌کند.





```
*****
* https_client.c --- very simple HTTPS client with no error checking
* usage: https_client servername
*****/
```

```
#include <stdio.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
void main(int argc, char **argv)
{
    SSL *ssl;
    SSL_CTX *ctx;
    SSL_METHOD *client_method;
    X509 *server_cert;
    int sd,err;
```





```
char *str,*hostname,outbuf[4096],inbuf[4096],host_header[512];
struct hostent *host_entry;
struct sockaddr_in server_socket_address;
struct in_addr ip;

/*=====
/* (1) initialize SSL library */
=====*/
SSLeay_add_ssl_algorithms( );
client_method = SSLv2_client_method( );
SSL_load_error_strings( );
ctx = SSL_CTX_new(client_method);
printf("(1) SSL context initialized\n\n");

/*=====
/* (2) convert server hostname into IP address */
=====*/

hostname = argv[1];
host_entry = gethostbyname(hostname);
bcopy(host_entry->h_addr, &(ip.s_addr), host_entry->h_length);
printf("(2) '%s' has IP address '%s'\n\n", hostname, inet_ntoa(ip));
```





```
/*=====
/* (3) open a TCP connection to port 443 on server */
=====*/
sd = socket (AF_INET, SOCK_STREAM, 0);
memset(&server_socket_address, '\0', sizeof(server_socket_address));
server_socket_address.sin_family = AF_INET;
server_socket_address.sin_port = htons(443);
memcpy(&(server_socket_address.sin_addr.s_addr),
host_entry->h_addr, host_entry->h_length);
err = connect(sd, (struct sockaddr*) &server_socket_address,
sizeof(server_socket_address));
if (err < 0) { perror("can't connect to server port"); exit(1); }
printf("(3) TCP connection open to host '%s', port %d\n\n",
hostname, server_socket_address.sin_port);

/*=====
/* (4) initiate the SSL handshake over the TCP connection */
=====*/
ssl = SSL_new(ctx); /* create SSL stack endpoint */
SSL_set_fd(ssl, sd); /* attach SSL stack to socket */
err = SSL_connect(ssl); /* initiate SSL handshake */
printf("(4) SSL endpoint created & handshake completed\n\n");
```





```
/*=====
/* (5) print out the negotiated cipher chosen */
/*=====

printf("(5) SSL connected with cipher: %s\n\n", SSL_get_cipher(ssl));

/*=====
/* (6) print out the server's certificate */
/*=====

server_cert = SSL_get_peer_certificate(ssl);
printf("(6) server's certificate was received:\n\n");
str = X509_NAME_oneline(X509_get_subject_name(server_cert), 0, 0);
printf(" subject: %s\n", str);
str = X509_NAME_oneline(X509_get_issuer_name(server_cert), 0, 0);
printf(" issuer: %s\n\n", str);
/* certificate verification would happen here */
X509_free(server_cert);
```





```
*****  
/* (7) handshake complete --- send HTTP request over SSL */  
*****  
  
sprintf(host_header,"Host: %s:443\r\n",hostname);  
strcpy(outbuf,"GET / HTTP/1.0\r\n");  
strcat(outbuf,host_header);  
strcat(outbuf,"Connection: close\r\n");  
strcat(outbuf,"\r\n");  
err = SSL_write(ssl, outbuf, strlen(outbuf));  
shutdown (sd, 1); /* send EOF to server */  
printf("(7) sent HTTP request over encrypted channel:\n\n%s\n",outbuf);  
  
*****  
/* (8) read back HTTP response from the SSL stack */  
*****  
  
err = SSL_read(ssl, inbuf, sizeof(inbuf) - 1);  
inbuf[err] = '\0';  
printf ("(8) got back %d bytes of HTTP response:\n\n%s\n",err,inbuf);
```





```
*****
/* (9) all done, so close connection & clean up */

*****
SSL_shutdown(ssl);
close (sd);
SSL_free (ssl);
SSL_CTX_free (ctx);
printf("(9) all done, cleaned up and closed connection\n\n");
}
```

این مثال بر روی Sun Solaris کامپایل و اجرا می‌شود، اما نحوه عملکرد برنامه‌های SSL بر روی بسیاری از پلتفرم‌های سیستم عامل را نشان می‌دهد. کل این برنامه، از جمله تمام رمزگذاری و مدیریت کلید و گواهی، به لطف ویژگی‌های قدرتمند ارائه شده توسط OpenSSL، در یک برنامه C سه صفحه‌ای قرار می‌گیرد.

اجازه دهید بخش به بخش برنامه را مرور کنیم:

بالای برنامه شامل فایل‌های پشتیبانی مورد نیاز برای پشتیبانی از شبکه TCP و SSL است.

Section 1 با فراخوانی `SSL_CTX_new` زمینه محلی را ایجاد می‌کند که پارامترهای Handshake و سایر وضعیت‌های اتصال SSL را رديابی می‌کند.

Section 2 نام میزبان ورودی (ارائه شده به عنوان آرگومان خط فرمان) را با استفاده از تابع `gethostbyname` یونیکس به آدرس IP تبدیل می‌کند. پلتفرم‌های دیگر ممکن است راههای دیگری برای ارائه این قابلیت داشته باشند.

Section 3 با ایجاد یک سوکت محلی، تنظیم اطلاعات آدرس راه دور و اتصال به سرور راه دور، اتصال TCP را به پورت ۴۴۳ روی سرور باز می‌کند.

هنگامی که اتصال TCP برقرار شد، لایه SSL را با استفاده از `SSL_new` و `SSL_set_fd` به اتصال TCP متصل می‌کنیم و با فراخوانی `SSL_connect` یا دست دادن SSL را با سرور





انجام می‌دهیم. وقتی **Section 4** انجام شد، ما یک کانال SSL کارآمد ایجاد شده با رمزهای انتخاب شده و گواهی‌های مبادله شده خواهیم داشت.

ارزش رمزگذاری انبوه (Bulk-Encryption Cipher) انتخاب شده را چاپ می‌کند.

Section 6 برخی از اطلاعات موجود در گواهی X.509 را که از سرور ارسال شده است چاپ می‌کند، از جمله اطلاعات مربوط به دارنده گواهی و سازمانی که گواهی را صادر کرده است. کتابخانه OpenSSL کار خاصی با اطلاعات گواهی سرور انجام نمی‌دهد. یک برنامه SSL واقعی، مانند یک مرورگر وب، برخی از بررسی‌های عمومی گواهی را انجام می‌دهد تا مطمئن شود که به درستی امضا شده است و از میزبان مناسب ارائه شده است.

در این مرحله، اتصال SSL ما برای انتقال امن داده آماده استفاده است. در **Section 7**، درخواست ساده HTTP را از طریق کانال SSL با استفاده از `SSL_write` ارسال می‌کنیم، سپس نیمه خروجی اتصال را می‌بندیم.

در **Section 8**، ما پاسخ اتصال را با استفاده از `SSL_read` می‌خوانیم و آن را روی صفحه چاپ می‌کنیم. از آنجایی که لایه SSL تمام رمزگذاری و رمزگشایی را انجام می‌دهد، ما فقط می‌توانیم دستورات HTTP معمولی را بنویسیم و بخوانیم.

در نهایت در **Section 9** پاکسازی را انجام می‌دهیم.

جهت کسب اطلاعات بیشتر در مورد کتابخانه‌های OpenSSL به <http://www.openssl.org> مراجعه کنید.

Executing Our Simple OpenSSL Client

در زیر خروجی سرویس گیرنده HTTP ساده ما را هنگامی که به سمت یک سرور امن اشاره می‌کنیم نشان می‌دهد. در این مورد، ما به کلاینت به صفحه اصلی کارگزاری آنلاین مورگان استنلی اشاره کردیم. لازم به ذکر است که شرکت‌های تجارت آنلاین به طور گسترده از HTTPS استفاده می‌کنند.





```
% https_client clients1.online.msdw.com

(1) SSL context initialized
(2) 'clients1.online.msdw.com' has IP address '63.151.15.11'
(3) TCP connection open to host 'clients1.online.msdw.com', port 443
(4) SSL endpoint created & handshake completed
(5) SSL connected with cipher: DES-CBC3-MD5
(6) server's certificate was received:

subject: /C=US/ST=Utah/L=Salt Lake City/O=Morgan Stanley/OU=Online/CN=clients1.online.msdw.com
issuer: /C=US/O=RSA Data Security, Inc./OU=Secure Server Certification Authority

(7) sent HTTP request over encrypted channel:

GET / HTTP/1.0
Host: clients1.online.msdw.com:443
Connection: close

(8) got back 615 bytes of HTTP response:

HTTP/1.1 302 Found
Date: Sat, 09 Mar 2002 09:43:42 GMT
Server: Stronghold/3.0 Apache/1.3.14 RedHat/3013c (Unix) mod_ssl/2.7.1 OpenSSL/0.9.6
Location: https://clients.online.msdw.com/cgi-bin/ICenter/home
Connection: close
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A HREF="https://clients.online.msdw.com/cgi-bin/ICenter/
home">here</A>.<P>
<HR>
<ADDRESS>Stronghold/3.0 Apache/1.3.14 RedHat/3013c Server at clients1.online.msdw.com
Port 443</ADDRESS>
</BODY></HTML>

(9) all done, cleaned up and closed connection
```

به محض تکمیل چهار بخش اول، کلاینت یک اتصال SSL باز دارد. سپس می‌تواند وضعیت اتصال و پارامترهای انتخابی را جویا شود و گواهی‌های سرور را بررسی کند.

در این مثال، کلاینت و سرور در مورد رمزگذاری انبوه DES-CBC3-MD5 مذاکره کردند. همچنین می‌توانید ببینید که گواهی سایت سرور متعلق به سازمان "Morgan Stanley" در "سالت لیک سیتی، یوتا، ایالات متحده آمریکا" است. این گواهی توسط RSA Data Security اعطا شده است و نام میزبان "clients1.online.msdw.com" است که با درخواست ما مطابقت دارد.

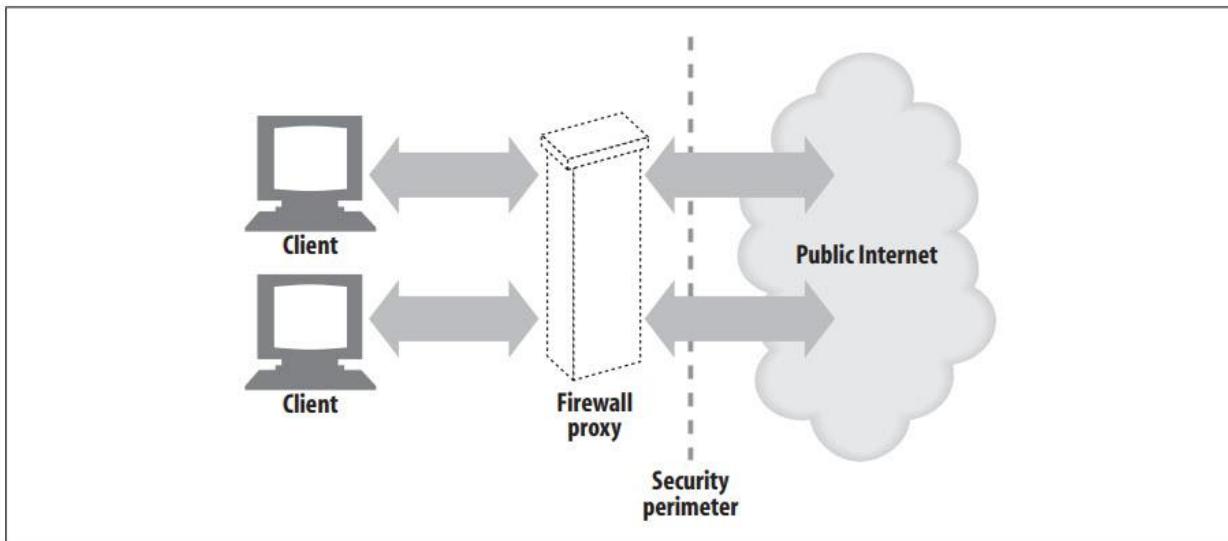
هنگامی که کانال SSL ایجاد شد و کلاینت در مورد گواهی سایت احساس راحتی کرد، درخواست HTTP خود را از طریق کانال امن ارسال می‌کند. در مثال ما، کلاینت یک درخواست HTTP ساده «GET / HTTP/1.0» ارسال می‌کند و یک پاسخ 302 Redirect دریافت می‌کند و از کاربر درخواست می‌کند که یک URL متفاوت واکشی کند.



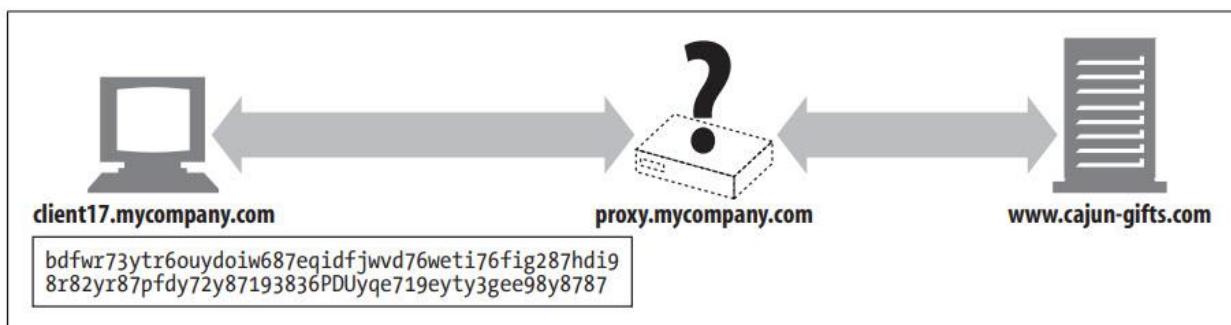


Tunneling Secure Traffic Through Proxies

کلاینت‌ها اغلب از سرورهای پروکسی وب برای دسترسی به سرورهای وب از طرف خود استفاده می‌کنند. به عنوان مثال، بسیاری از شرکت‌ها یک پروکسی را در محیط امنیتی شبکه شرکت و اینترنت عمومی قرار می‌دهند (شکل زیر). پروکسی تنها وسیله‌ای است که توسط روترهای فایرووال، مجاز به تبادل ترافیک HTTP است و ممکن است از بررسی ویروس یا سایر کنترل‌های محتوا استفاده کند.



اما هنگامی که کلاینت با استفاده از کلید عمومی سرور شروع به رمزگذاری داده‌ها در سرور می‌کند، پروکسی دیگر توانایی خواندن هدر HTTP را ندارد! و اگر پروکسی نتواند هدر HTTP را بخواند، نمی‌داند درخواست را کجا ارسال کند (شکل زیر).



برای اینکه HTTPS با پراکسی‌ها کار کند، چند تغییر لازم است تا به پراکسی بگوید کجا باید متصل شود. یکی از تکنیک‌های محبوب پروتکل HTTPS SSL Tunneling است.





با استفاده از پروتکل HTTPS Tunneling، کلاینت ابتدا میزبان امن و پورتی را که می‌خواهد به آن متصل شود را به پروکسی می‌گوید. این کار را قبل از شروع رمزگذاری در متن ساده انجام می‌دهد، بنابراین پروکسی می‌تواند این اطلاعات را بخواند.

برای ارسال اطلاعات نقطه پایانی متن ساده از یک متاد افزودنی جدید به نام CONNECT استفاده می‌شود. متاد CONNECT به پراکسی می‌گوید که یک اتصال به هاست و شماره پورت مورد نظر را باز کند و پس از انجام این کار، داده‌ها را مستقیماً بین کلاینت و سرور تونل کند. متاد CONNECT یک دستور متنی تک خطی است که نام میزبان و پورت سرور مبدا امن را ارائه می‌کند که با یک دو نقطه از هم جدا شده‌اند. host:port با یک فاصله و یک رشته نسخه CRLF و همچنین HTTP دنبال می‌شود. پس از آن یک سری خط هدر درخواست HTTP صفر یا بیشتر بوده و به دنبال آن یک خط خالی وجود دارد. پس از خط خالی، اگر Handshake برقراری اتصال موفقیت آمیز بود، انتقال داده‌های SSL می‌تواند آغاز شود. به عنوان مثال:

CONNECT home.netscape.com:443

HTTP/1.0 User-agent: Mozilla/1.1N

<raw SSL-encrypted data would follow here...>

پس از خط خالی در درخواست، کلاینت منتظر پاسخ از طرف پروکسی خواهد بود. پروکسی درخواست را ارزیابی می‌کند و مطمئن می‌شود که معتبر است و کاربر مجاز به درخواست چنین اتصالی است. اگر همه چیز مرتب باشد، پروکسی به سرور مقصد متصل می‌شود و در صورت موفقیت آمیز بودن، یک پاسخ 200 Connection Established را برای کلاینت ارسال می‌کند.

HTTP/1.0 200 Connection established

Proxy-agent: Netscape-Proxy/1.1





For More Information

امنیت و رمزنگاری موضوعات بسیار مهم و بسیار پیچیده‌ای هستند. اگر می‌خواهید درباره امنیت HTTP، رمزنگاری دیجیتال، گواهی‌های دیجیتال و زیرساخت کلید عمومی اطلاعات بیشتری کسب کنید، در اینجا چند نقطه شروع وجود دارد.

HTTP Security

Web Security, Privacy & Commerce/ Simson Garfinkel, O'Reilly & Associates, Inc

<http://www.ietf.org/rfc/rfc2818.txt>

<http://www.ietf.org/rfc/rfc2817.txt>

SSL and TLS

<http://www.ietf.org/rfc/rfc2246.txt>

<http://developer.netscape.com/docs/manuals/security/sslin/contents.htm>

<http://www.netscape.com/eng/ssl3/draft302.txt>

<http://developer.netscape.com/tech/security/ssl/howitworks.html>

<http://www.openssl.org>

Public-Key Infrastructure

<http://www.ietf.org/html.charters/pkix-charter.html>

<http://www.ietf.org/rfc/rfc2459.txt>

Digital Cryptography

Applied Cryptography/Bruce Schneier, John Wiley & Sons

The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography/Simon Singh, Anchor Books





فصل پانزدهم – Entities and Encodings

HTTP هر روز میلیاردها شیء رسانه‌ای از انواع مختلف را ارسال می‌کند. تصاویر، متن، فیلم‌ها، برنامه‌های نرم‌افزاری و هر مورد دیگری که شما آن را نام ببرید، HTTP آن را ارسال می‌کند. HTTP همچنین مطمئن می‌شود که پیام‌هایش به درستی قابل حمل، شناسایی، استخراج و پردازش هستند. به طور خاص، HTTP تضمین می‌کند که بسته خود:

- می‌تواند به درستی شناسایی شود (با استفاده از فرمتهای رسانه نوع محتوا و هدرهای Content-Header) تا مرورگرها و سایر کلاینت‌ها بتوانند محتوا را به درستی پردازش کنند.
- می‌تواند به درستی بسته بندی شود (با استفاده از هدرهای Content-Length و Content-Encoding)
- تازه است (با استفاده از اعتبار سنجی موجودیت و کنترل‌های Cache-Expiration)
- نیازهای کاربر را برآورده می‌کند (بر اساس هدرهای Content-Negotiation Accept)
- حرکت سریع و کارآمد در شبکه (با استفاده از درخواست‌های محدوده، Encoding دلتا و سایر فشرده سازی داده‌ها)
- کامل و بدون دستکاری (با استفاده از هدرهای Encoding انتقال و Content-MD5 Checksums)

برای تحقق همه این‌ها، HTTP از موجودیت‌های well-labeled برای حمل محتوا استفاده می‌کند.

این فصل در مورد موجودیت‌ها، هدرهای موجودیت مرتبط با آن‌ها و نحوه کار آن‌ها برای حمل محموله وب بحث می‌کند. ما نشان خواهیم داد که چگونه HTTP موارد ضروری اندازه، نوع و رمزگذاری محتوا را فراهم می‌کند. همچنین برخی از ویژگی‌های پیچیده‌تر و قدرتمند موجودیت‌های HTTP، از جمله درخواست‌های محدوده، کدگذاری دلتا، Digestها و رمزگذاری‌های تکه‌ای را توضیح خواهیم داد.

این فصل شامل موارد زیر است:

- قالب و رفتار موجودیت‌های پیام HTTP به عنوان HTTP Data Container
- HTTP چگونه اندازه بدن‌های موجودیت را توصیف می‌کند و HTTP در نحوه اندازه گیری به چه مواردی نیازی دارد.
- هدر موجودیت برای توصیف قالب، الفبا و زبان محتوا استفاده می‌شود تا کلاینت‌ها بتوانند آن را به درستی پردازش کنند.

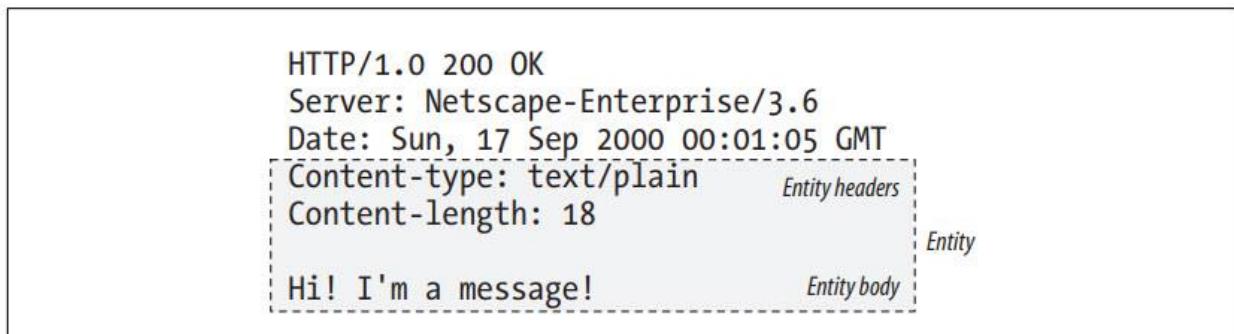




- رمزگذاری‌های محتوای بزرگشتبذیر، که توسط فرستنده‌ها برای تغییر قالب داده‌های محتوا قبل از ارسال استفاده می‌شود تا فضای کمتری اشغال کند یا ایمن‌تر شود.
- رمزگذاری انتقال، که نحوه ارسال HTTP داده‌ها را برای بهبود ارتباطات برخی از انواع محتوا و کدگذاری تکه‌ای، یک رمزگذاری انتقالی که داده‌ها را به چند قسمت تقسیم می‌کند تا محتوایی با طول نامعلوم را به صورت ایمن ارائه کند، تغییر می‌دهد.
- مجموعه‌ای از برچسب‌ها، برچسب‌ها، زمان‌ها و جمع‌بندی‌هایی که به کلاینت‌ها کمک می‌کند آخرین نسخه محتوای درخواستی را دریافت کنند.
- اعتبار سنجی‌هایی که مانند شماره نسخه روی محتوا عمل می‌کنند، بنابراین برنامه‌های کاربردی وب می‌توانند اطمینان حاصل کنند که محتوای تازه دارند و فیلدهای هدر HTTP برای کنترل تازگی اشیا طراحی شده‌اند.
- محدوده‌هایی که برای ادامه دانلودهای لغو شده از جایی که متوقف شده‌اند مفید هستند.
- پسوندهای کدگذاری دلتا HTTP، که به کلاینت‌ها اجازه می‌دهد فقط آن قسمت‌هایی از یک صفحه وب را درخواست کنند که واقعاً از یک ویرایش قبلی تغییر کرده‌اند.
- Checksum های بدنه موجودیت که برای شناسایی تغییرات در محتوای موجودیت در هنگام عبور از پروکسی‌ها استفاده می‌شود.

Messages Are Crates, Entities Are Cargo

اگر پیام‌های HTTP را جعبه‌های سیستم حمل و نقل اینترنتی می‌دانید، موجودیت‌های HTTP محموله واقعی پیام‌ها هستند. شکل زیر یک موجودیت ساده را نشان می‌دهد که در یک پیام پاسخ HTTP حمل می‌شود.



هدرهای موجودیت، یک سند متن ساده (`Content-Type: text/plain`) را نشان می‌دهد که تنها ۱۸ کاراکتر طول دارد (`Content-Length: 18`). مثل همیشه، یک خط خالی (CRLF) فیلدهای هدر را از ابتدای بدنه جدا می‌کند.





هدرهای موجودیت HTTP (که در فصل ۳ پوشش داده شده است) محتویات یک پیام HTTP را توصیف می‌کنند.
HTTP/1.1 10
فیلد هدر موجودیت اصلی را تعریف می‌کند:

Content-Type

نوع شیئی که توسط موجودیت حمل می‌شود.

Content-Length

طول یا اندازه پیام ارسال شده

Content-Language

زبان انسانی که به بهترین وجه با شیء ارسال شده مطابقت دارد.

Content-Encoding

هر تبدیل (فسرده سازی و غیره) که روی داده‌های شی انجام می‌شود.

Content-Location

یک مکان جایگزین برای شی در زمان درخواست.

Content-Range

اگر این یک موجودیت جزئی باشد، این هدر مشخص می‌کند که کدام قطعات از کل گنجانده شده است.

Content-MD5

ای از محتویات بدن موجودیت Checksum

Last-Modified

تاریخی که در آن این محتوای خاص در سرور ایجاد یا اصلاح شده است.

Expires

تاریخ و زمانی که در آن داده‌های موجودیت قدیمی می‌شوند.

Allow

چه متدهای درخواستی در این منبع قانونی هستند. به عنوان مثال، GET و HEAD.





Etag

یک اعتبارسنجی منحصر به فرد برای این نمونه خاص از سند. هدر ETag به طور رسمی به عنوان یک هدر موجودیت تعریف نشده است، اما یک هدر مهم برای بسیاری از عملیات مربوط به موجودیت‌هاست.

Cache-Control

دستورالعمل‌هایی درباره نحوه ذخیره‌سازی این سند. هدر Cache-Control، مانند هدر ETag، به طور رسمی به عنوان هدر موجودیت تعریف نشده است.

Entity Bodies

بدنی موجودیت فقط حاوی محموله خام است (اگر هدر Content-Encoding وجود داشته باشد، محتوا قبلاً توسط الگوریتم رمزگذاری شده است و اولین بایت موجودیت اولین بایت محموله رمزگذاری شده (مثلاً فشرده) است). از آنجایی که محموله بدنی موجودیت فقط داده خام است، هدرهای موجودیت برای توصیف معنای آن داده مورد نیاز است. به عنوان مثال، هدر موجودیت ContentType به ما می‌گوید که چگونه داده‌ها (تصویر، متن، و غیره) را تفسیر کنیم، و هدر موجودیت Content-Encoding به ما می‌گوید که آیا داده‌ها فشرده شده‌اند یا به شکل دیگری کدگذاری مجدد شده‌اند. ما در مورد همه این‌ها و بیشتر در بخش‌های آینده صحبت می‌کنیم.

محتوای خام بلافاصله بعد از خط خالی CRLF که انتهای فیلهای هدر را مشخص می‌کند شروع می‌شود. محتوا هر چه باشد - متن یا باینری، سند یا تصویر، فشرده یا غیر فشرده، انگلیسی یا فرانسوی یا ژاپنی - درست بعد از CRLF قرار می‌گیرد.

شکل زیر دو نمونه از پیام‌های HTTP واقعی را نشان می‌دهد، یکی حاوی یک موجودیت متنی، دیگری حامل یک موجودیت تصویر. مقادیر هگزادسیمال محتوای دقیق پیام را نشان می‌دهد:

- در بخش a شکل، بدنی موجودیت با بایت شماره ۶۵، درست بعد از CRLF هدر انتهایی شروع می‌شود.
بدنی موجودیت حاوی کاراکترهای ASCII برای Hi! I'm a message!
- در بخش b شکل، بدنی موجودیت با بایت شماره ۶۷ شروع می‌شود. بدنی موجودیت حاوی محتوای باینری تصویر GIF است. فایل‌های GIF با امضای نسخه ۶ بایتی، عرض ۱۶ بیتی و ارتفاع ۱۶ بیتی شروع می‌شوند.
شما می‌توانید هر سه مورد را مستقیماً در بدنی موجودیت مشاهده کنید.





(a) Text/plain entity in HTTP response message

HTTP/1.0 200 OK
Content-type: text/plain
Content-length: 18

Hi! I'm a message!

00: 4854 5450 2f31 2e30 2032 3030 204f 4b0d
16: 0a43 6f6e 7465 6e74 2d74 7970 653a 2074
32: 6578 742f 706c 6169 6e0d 0a43 6f6e 7465
48: 6e74 2d6c 656e 6774 683a 2031 380d 0a0d
64: 0a48 6921 2049 276d 2061 206d 6573 7361
80: 6765 210a

final LF (0x0A=<LF>)

start-of-content (0x48="H")

HTTP/1.0 200 OK.
.Content-type: t
ext/plain..Conte
nt-length: 18...
.Hi! I'm a messa
ge!.

(b) Image/gif entity in HTTP response message

HTTP/1.0 200 OK
Content-Type: image/gif
Content-Length: 34867



00: 4854 5450 2f31 2e30 2032 3030 204f 4b0d
16: 0a43 6f6e 7465 6e74 2d74 7970 653a 2069
32: 6d61 6765 2f67 6966 0d0a 436f 6e74 656e
48: 742d 6c65 6e67 7468 3a20 3334 3836 370d
64: 0a0d 0a47 4946 3837 6127 0206 02f7 0000
80: 0402 0404 8204 8482 04fc fe04 8402 0404
96: 42a4 a4ca f484 8284 dce2 dc9c a29c 444a
=85 final LF start-of-content Width Height
(0x0A=<LF>) ("GIF87a") (0x0227=551) (0x0206=518)

HTTP/1.0 200 OK.
.Content-type: i
mage/gif..Conten
t-length: 34867.
...GIF87a'.....
.....DJ

Content-Length: The Entity's Size

هدر Content-Length اندازه بدنی موجود در پیام را بر حسب بایت نشان می‌دهد. اندازه شامل هر گونه رمزگذاری محتوا است (طول محتوا یک فایل متنی فشرده شده gzip اندازه فشرده خواهد بود، نه اندازه اصلی).

هدر Content-Length برای پیام‌های دارای بدنی موجودیت اجباری است، مگر اینکه پیام با استفاده از رمزگذاری تکه‌ای منتقل شود. Content-Length برای تشخیص قطع شدن زودهنگام پیام در هنگام خرابی سرورها و تقسیم بندی صحیح پیام‌هایی که یک اتصال دائمی را به اشتراک می‌گذارند مورد نیاز است.

Detecting Truncation

نسخه‌های قدیمی‌تر HTTP از اتصال نزدیک برای محدود کردن انتهای پیام استفاده می‌کردند.

اما، بدون Content-Length، کلاینت‌ها نمی‌توانند بین بستن موفقیت‌آمیز اتصال در انتهای پیام و بستن اتصال به دلیل خرابی سرور در وسط پیام تمایز قائل شوند. کلاینت‌ها برای تشخیص کوتاه شدن پیام به Content-Length نیاز دارند.

کوتاه شدن پیام مخصوصاً برای سرورهای پراکسی کش¹ شدید است. اگر Cache پیام کوتاه شده² را دریافت کند و کوتاهی را نشناشد، ممکن است محتوای معیوب را ذخیره کرده و بارها آن را ارائه کند. سرورهای

¹ Caching Proxy

² Truncated





پروکسی کش معمولاً بدندهای HTTP را که هدر Content-Length ندارند، ذخیره نمی‌کنند تا خطر ذخیره پیام‌های کوتاه‌شده را کاهش دهند.

Incorrect Content-Length

یک Content-Length نادرست می‌تواند باعث آسیب بیشتر از Content-Length شود. از آنجایی که برخی از کلاینت‌ها و سرورهای اولیه با توجه به محاسبات Content-Length دارای اشکالات شناخته شده‌ای بودند، برخی از کلاینت‌ها، سرورها و پراکسی‌ها حاوی الگوریتم‌هایی هستند که سعی در شناسایی و تصحیح تعامل با سرورهای خراب دارند. User Agent های HTTP/1.1 به طور رسمی قرار است در صورت دریافت و شناسایی طول نامعتبر، آن را به کاربر اطلاع دهند.

Content-Length and Persistent Connections

HTTP برای اتصالات مداوم ضروری است. اگر پاسخ با اتصال دائمی مواجه شود، پاسخ Content-Length می‌تواند بلافضلله پاسخ فعلی را دنبال کند. هدر Content-Length به کلاینت اجازه می‌دهد بداند یک پیام کجا به پایان می‌رسد و پیام بعدی شروع می‌شود. از آنجا که اتصال پایدار است، کلاینت نمی‌تواند از اتصال نزدیک برای شناسایی پایان پیام استفاده کند. بدون هدر Content-Length، برنامه‌های HTTP نمی‌دانند که بدنی یک موجودیت به کجا ختم می‌شود و پیام بعدی شروع می‌شود.

همانطور که در Transfer Encoding and Chunked Encoding خواهیم دید، یک موقعیت وجود دارد که می‌توانید از اتصالات دائمی بدون داشتن هدر Content-Length استفاده کنید. و آن زمانی است که از رمزگذاری تکه‌شده استفاده می‌کنید. رمزگذاری تکه‌ای داده‌ها را در یک سری تکه‌ها ارسال می‌کند که هر کدام اندازه مشخصی دارند. حتی اگر سرور اندازه کل موجودیت را در زمان تولید هدرها نداند (غلب به این دلیل که موجودیت به صورت پویا تولید می‌شود)، سرور می‌تواند از رمزگذاری تکه برای انتقال قطعات با اندازه کاملاً مشخص استفاده کند.

Content Encoding

HTTP به شما امکان می‌دهد محتويات یک موجودیت را رمزگذاری کنید، شاید آن را این‌تر کنید یا آن را فشرده کنید تا فضای کمتری اشغال کند (در ادامه این فصل فشرده سازی را به تفصیل توضیح خواهیم داد). اگر متن به صورت محتوا رمزگذاری شده باشد، هدر Content-Length طول بدنی کدگذاری شده را بر حسب بایت مشخص می‌کند، نه طول متن اصلی و رمزگذاری نشده را.





برخی از برنامه‌های HTTP شناخته شده‌اند که این اشتباه را دریافت می‌کنند و اندازه داده‌ها را قبل از رمزگذاری ارسال می‌کنند که باعث خطاهای جدی، به خصوص با اتصالات مدام می‌شود. متأسفانه، هیچ یک از هدرهای توصیف شده در مشخصات HTTP/1.1 را نمی‌توان برای ارسال طول بدنی اصلی و رمزگذاری نشده استفاده کرد، که این امر باعث می‌شود کلاینت‌ها در تأیید یکپارچگی فرآیندهای رمزگذاری نشده خود مشکل داشته باشند.

Rules for Determining Entity Body Length

قوانين زیر چگونگی تعیین صحیح طول و انتهای بدنی یک موجودیت را در شرایط مختلف توضیح می‌دهد. قوانین باید به ترتیب اعمال شوند؛

۱. اگر یک نوع پیام HTTP خاص مجاز به داشتن متن نیست، هدر Content-Length را برای محاسبات بدنی نادیده بگیرید. هدرهای Content-Length در این مورد Informational هستند و طول واقعی بدنی را توصیف نمی‌کنند. (برنامه‌های ساده HTTP اگر فرض کنند که Content-Length همیشه به معنای وجود یک بدنی است، ممکن است دچار مشکل شوند). مهمترین مثال پاسخ HEAD است. متدهای HEAD از سرور درخواست می‌کند که هدرهایی را که با یک درخواست GET معادل برگردانده می‌شوند، ارسال کند، اما هیچ متنی وجود ندارد. از آنجایی که یک پاسخ GET یک هدر Content-Length را ارسال می‌کند، پاسخ HEAD نیز چنین خواهد بود - اما برخلاف پاسخ GET، پاسخ HEAD بدنی نخواهد داشت. پاسخ‌های 1xx، 204 و 304 نیز می‌توانند دارای هدرهای Informational با طول محتوا باشند، اما بدون بدنی موجودیت. پیام‌هایی که بدنی‌های موجودیت را منع می‌کنند باید در اولین خط خالی بعد از هدرها خاتمه یابند، صرف نظر از اینکه کدام فیلد های هدر موجودیت وجود دارد.

۲. اگر پیامی حاوی هدر Transfer-Encoding باشد (غیر از رمزگذاری پیش‌فرض «هویت» HTTP)، موجودیت با الگوی خاصی به نام «تکه صفر بایت»^۳ خاتمه می‌یابد، مگر اینکه پیام ابتدا با بستن اتصال خاتمه یابد. ما در ادامه این فصل در مورد رمزگذاری‌های انتقال و رمزگذاری‌های تکه‌ای بحث خواهیم کرد.

۳. اگر پیامی دارای هدر Content-Length باشد (و نوع پیام به بدنی‌های موجودیت اجازه می‌دهد)، مقدار Content-Length شامل طول بدنی است، مگر اینکه هدر Transfer-Encoding غیر هویتی وجود داشته باشد. اگر پیامی با یک فیلد هدر Content-Length و یک فیلد هدر Transfer-Encoding غیر هویتی دریافت می‌شود، باید Content-Length را نادیده بگیرید، زیرا رمزگذاری انتقال، نحوه نمایش و انتقال بدنی‌های موجودیت را تغییر می‌دهد (و احتمالاً تعداد بایت‌های ارسال شده).

³ zero-byte chunk





۴. اگر پیام از نوع رسانه "چند بخشی/بایترنج"^۴ استفاده می‌کند و طول موجودیت به طور دیگری مشخص نشده است (در هدر Content-Length)، هر قسمت از پیام چند قسمتی اندازه خود را مشخص می‌کند. این نوع چند قسمتی تنها نوع بدن موجودیتی است که اندازه خود را به صورت مستقل تعیین می‌کند، بنابراین این نوع رسانه نباید ارسال شود مگر اینکه فرستنده بداند گیرنده می‌تواند آن را تجزیه کند.

۵. اگر هیچ یک از قوانین بالا مطابقت نداشته باشد، موجودیت با بسته شدن اتصال به پایان می‌رسد. در عمل، فقط سرورها می‌توانند از Connection Close برای نشان دادن پایان یک پیام استفاده کنند. کلاینت‌ها نمی‌توانند اتصال را بینندن تا پیام‌های سرویس گیرنده پایان یابد، زیرا این کار هیچ راهی برای سرور برای ارسال پاسخ باقی نمی‌گذارد.

۶. برای سازگاری با برنامه‌های HTTP/1.0، هر درخواست HTTP/1.1 که دارای یک نهاد است نیز باید دارای یک فیلد هدر Content-Length معتبر باشد (مگر اینکه سرور شناخته شده باشد که با HTTP/1.1 سازگار است). مشخصات HTTP/1.1 توصیه می‌کند که اگر درخواستی حاوی بدن باشد و Content-Length نداشته باشد، اگر نمی‌تواند طول پیام را تعیین کند، سرور باید یک پاسخ 400 Bad Request ارسال نموده، یا اگر بخواهد برای دریافت یک Content-Length معتبر اصرار داشته باشد، یک پاسخ 411 Length Required ارسال نماید.

Entity Digests

اگرچه HTTP معمولاً بر روی یک پروتکل انتقال قابل اعتماد مانند TCP/IP پیاده‌سازی می‌شود، ممکن است قسمت‌هایی از پیام‌ها به دلایل مختلفی مانند پروکسی‌های transcoding ناسازگار^۵ یا پراکسی‌های واسطه باگ^۶ در حین انتقال اصلاح شوند. برای تشخیص اصلاح ناخواسته داده‌های بدن موجودیت، فرستنده می‌تواند Checksum ای از داده‌ها را زمانی که موجودیت اولیه تولید می‌شود، تولید کند و گیرنده می‌تواند را بررسی کند تا هرگونه تغییر ناخواسته موجودیت را شناسایی نماید.

هدر Content-MD5 توسط سرورها برای ارسال نتیجه اجرای الگوریتم MD5 بر روی بدن موجودیت استفاده می‌شود. فقط سروری که پاسخ از آنجا نشأت می‌گیرد می‌تواند هدر Content-MD5 را محاسبه و ارسال کند. پروکسی‌های میانی و Cache‌ها ممکن است هدر را تغییر داده یا اضافه نکنند - این امر کل هدف تأیید یکپارچگی انتها به انتها را نقض می‌کند. هدر Content-MD5 حاوی MD5 محتوا پس از اعمال همه رمزگذاری‌های محتوا به بدن موجودیت و قبل از اعمال هر گونه رمزگذاری انتقال به آن است. کلاینت‌هایی که به دنبال تأیید صحت پیام هستند باید ابتدا رمزگذاری‌های انتقال را رمزگشایی کنند، سپس MD5 بدن موجودیت رمزگذاری

⁴ multipart/byteranges

⁵ noncompliant transcoding proxies

⁶ buggy intermediary proxies





نشده حاصل را محاسبه کنند. به عنوان مثال، اگر سندی با استفاده از الگوریتم gzip فشرده شود، سپس با رمزگذاری تکه‌ای ارسال شود، الگوریتم MD5 روی بدنے کامل اجرا می‌شود.

علاوه بر بررسی یکپارچگی پیام، MD5 می‌تواند به عنوان کلیدی در جدول هش برای مکان یابی سریع اسناد و کاهش ذخیره سازی تکراری محتوا استفاده شود. علیرغم این کاربردهای احتمالی، هدر Content-MD5 اغلب ارسال نمی‌شود.

برنامه‌های افزودنی به HTTP الگوریتم‌های Digest دیگری را در پیش نویس‌های IETF پیشنهاد کرده‌اند. این برنامه‌های افزودنی یک هدر جدید به نام Want-Digest پیشنهاد کرده‌اند که به کلاینت‌ها اجازه می‌دهد نوع Digest مورد انتظار خود را با پاسخ مشخص کنند. مقادیر کیفیت را می‌توان برای پیشنهاد الگوریتم‌های Digest چندگانه و نشان دادن اولویت استفاده کرد.

Media Type and Charset

فیلد هدر Content-Type نوع MIME بدنی موجودیت را توصیف می‌کند. نوع MIME یک نام استاندارد است که نوع اصلی رسانه حمل شده به عنوان محموله⁷ را توصیف می‌کند (فایل HTML، سند Microsoft Word ویدئو MPEG و غیره). برنامه‌های کلاینت از نوع MIME برای رمزگشایی و پردازش صحیح محتوا استفاده می‌کنند.

مقادیر Content-Type از انواع استاندارد MIME هستند که در IANA⁸ ثبت شده‌اند. انواع MIME شامل یک نوع رسانه اصلی (به عنوان مثال، متن، تصویر، صدا) و به دنبال آن یک اسلش و به دنبال آن یک نوع فرعی است که نوع رسانه را بیشتر مشخص می‌کند. جدول زیر چند نوع MIME متدائل را برای هدر ContentType فهرست می‌کند.

⁷ cargo

⁸ Internet Assigned Numbers Authority





Media type	Description
text/html	Entity body is an HTML document
text/plain	Entity body is a document in plain text
image/gif	Entity body is an image of type GIF
image/jpeg	Entity body is an image of type JPEG
audio/x-wav	Entity body contains WAV sound data
model/vrml	Entity body is a three-dimensional VRML model
application/vnd.ms-powerpoint	Entity body is a Microsoft PowerPoint presentation
multipart/byteranges	Entity body has multiple parts, each containing a different range (in bytes) of the full document
message/http	Entity body contains a complete HTTP message (see TRACE)

توجه به این نکته مهم است که هدر Content-Type نوع رسانه بدن موجودیت اصلی را مشخص می‌کند. برای مثال، اگر موجودیت از رمزگذاری محتوا عبور کرده باشد، هدر Content-Type همچنان نوع بدن موجودیت را قبل از رمزگذاری مشخص می‌کند.

Character Encodings for Text Media

هدر Content-Type همچنین از پارامترهای اختیاری برای تعیین بیشتر نوع محتوا پشتیبانی می‌کند. پارامتر "charset" مثال اصلی است که مکانیسم تبدیل بیت‌ها از موجودیت به کاراکتر در یک فایل متنی را مشخص می‌کند:

Content-Type: text/html; charset=iso-8859-4

Multipart Media Types

پیام‌های ایمیل «چند بخشی» MIME حاوی پیام‌های متعددی هستند که به هم چسبیده و به عنوان یک پیام واحد و پیچیده ارسال می‌شوند. هر جزء مستقل است، با مجموعه هدرهای خاص خود که محتوای آن را توصیف می‌کند. اجزای مختلف به هم متصل شده و توسط یک رشته محدود می‌شوند.

HTTP همچنین از بدن‌های چند بخشی پشتیبانی می‌کند. با این حال، آن‌ها عموماً تنها در یکی از این دو حالت ارسال می‌شوند: در ارسال‌های فرم تکمیل شده و در پاسخ‌های محدوده‌ای که قطعات یک سند را حمل می‌کنند.

Multipart Form Submissions

هنگامی که یک فرم تکمیل‌کننده HTTP ارسال می‌شود، فیلد‌های متنی با طول متغیر و اشیاء آپلود شده به عنوان بخش‌های مجزا از یک بدن چند بخشی ارسال می‌شوند و به فرم‌ها اجازه می‌دهند با





مقداری از انواع و طول‌های مختلف پر شوند. برای مثال، می‌توانید فرمی را پر کنید که نام شما و توضیحاتی را با نام مستعار و یک عکس کوچک می‌پرسد، در حالی که دوست شما ممکن است نام کامل خود و یک مقاله طولانی را در مورد علاقه خود به تعمیر اتوبوس‌های فولکس‌واگن بنویسد.

Content-Type: multipart/form-data یا یک هدر HTTP چنین درخواست‌هایی را با یک Type: multipart/mixed header و یک بدن چند بخشی ارسال می‌کند، مانند این:

Content-Type: multipart/form-data; boundary=[abcdefghijklmnopqrstuvwxyz]

جایی که مرز رشته جداکننده بین قسمت‌های مختلف بدن را مشخص می‌کند.

مثال زیر کدگذاری multipart/form-data را نشان می‌دهد. فرض کنید این فرم را داریم:

```
<FORM action="http://server.com/cgi/handle"
      enctype="multipart/form-data"
      method="post">
<P>
What is your name? <INPUT type="text" name="submit-name"><BR>
What files are you sending? <INPUT type="file" name="files"><BR>
<INPUT type="submit" value="Send"> <INPUT type="reset">
</FORM>
```

اگر کاربر «Sally» را در قسمت ورودی متن وارد کرده و فایل متنی «essayfile.txt» را انتخاب کند، User-Agent ممکن است داده‌های زیر را ارسال کند:

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Sally
--AaB03x
Content-Disposition: form-data; name="files"; filename="essayfile.txt"
Content-Type: text/plain
...contents of essayfile.txt...
--AaB03x--
```

اگر کاربر یک فایل (تصویر) دوم، "imagefile.gif" را انتخاب کند، User-Agent ممکن است قطعات را به صورت زیر بسازد:





```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Sally
--AaB03x
Content-Disposition: form-data; name="files"
Content-Type: multipart/mixed; boundary=BbC04y
--BbC04y
Content-Disposition: file; filename="essayfile.txt"
Content-Type: text/plain
...contents of essayfile.txt...
--BbC04y
Content-Disposition: file; filename="imagefile.gif"
Content-Type: image/gif
Content-Transfer-Encoding: binary
...contents of imagefile.gif...
--BbC04y--
--AaB03x--
```

Multipart Range Responses

پاسخ‌های HTTP به درخواست‌های محدوده نیز می‌تواند چند بخشی باشد. چنین پاسخ‌هایی با هدر-Content-Type: multipart/byteranges و بدنه چند بخشی با محدوده‌های مختلف ارائه می‌شوند. در اینجا نمونه‌ای از پاسخ چند بخشی به درخواست برای محدوده‌های مختلف یک سند آورده شده است:

```
HTTP/1.0 206 Partial content
Server: Microsoft-IIS/5.0
Date: Sun, 10 Dec 2000 19:11:20 GMT
Content-Location: http://www.joes-hardware.com/gettysburg.txt
Content-Type: multipart/x-byteranges; boundary=--[abcdefghijklmnopqrstuvwxyz]--
Last-Modified: Sat, 09 Dec 2000 00:38:47 GMT

--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 0-174/1441
```





Fourscore and seven years ago our fathers brough forth on this continent a new nation, conceived in liberty and dedicated to the proposition that all men are created equal.

--[abcdefghijklmnopqrstuvwxyz]--

Content-Type: text/plain

Content-Range: bytes 552-761/1441

But in a larger sense, we can not dedicate, we can not consecrate, we can not hallow this ground. The brave men, living and dead who struggled here have consecrated it far above our poor power to add or detract.

--[abcdefghijklmnopqrstuvwxyz]--

Content-Type: text/plain

Content-Range: bytes 1344-1441/1441

and that government of the people, by the people, for the people shall not perish from the earth.

--[abcdefghijklmnopqrstuvwxyz]--

درخواست‌های محدوده^۹ در ادامه این فصل با جزئیات بیشتر مورد بحث قرار می‌گیرند.

Content Encoding

برنامه‌های HTTP گاهی اوقات می‌خواهند محتوا را قبل از ارسال آن کدگذاری کنند. به عنوان مثال، یک سرور ممکن است یک سند HTML بزرگ را قبل از ارسال آن به یک کلاینت که از طریق یک اتصال آهسته متصل است، فشرده کند تا به کاهش زمان لازم برای انتقال موجودیت کمک کند. سرور ممکن است محتویات را به گونه‌ای کدگذاری کند که از مشاهده محتوای سند توسط اشخاص ثالث غیرمجاز جلوگیری شود.

این نوع کدگذاری‌ها بر روی محتوا در فرستنده اعمال می‌شود. هنگامی که محتوا به صورت محتوا کدگذاری شد، داده‌های کدگذاری شده طبق معمول به گیرنده موجود در بدنه موجودیت ارسال می‌شود.

The Content-Encoding Process

فرآیند رمزگذاری محتوا به شرح زیر است:

یک وب سرور یک پیام پاسخ اصلی با هدرهای Content-Type و Content-Length تولید می‌کند.

⁹ Range Requests

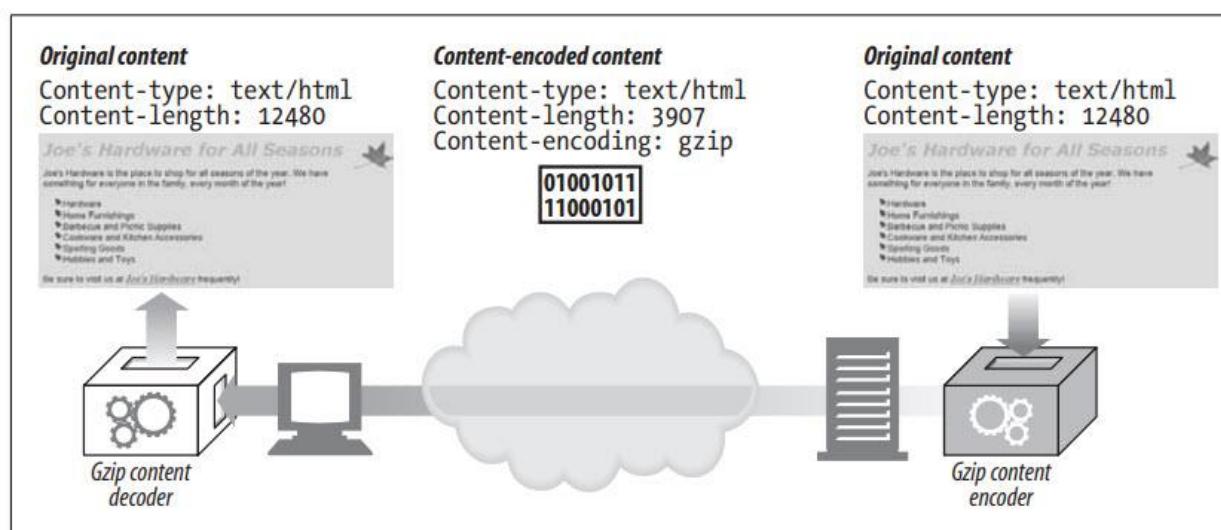




یک سرور کدگذاری محتوا^{۱۰} (شاید سرور مبدا یا یک پیام رمزگذاری شده ایجاد می‌کند. پیام رمزگذاری شده دارای همان Content-Type است اما (مثلاً اگر بدن فشرده شده باشد) Content-Encoding به پیام کدگذاری شده اضافه متفاوتی دارد. سرور کدگذاری محتوا یک هدر Length می‌کند تا برنامه دریافت کننده بتواند آن را رمزگشایی کند.

یک برنامه دریافت کننده، پیام کدگذاری شده را دریافت می‌کند، آن را رمزگشایی می‌کند و نسخه اصلی را دریافت می‌کند.

شکل زیر یک مثال رمزگذاری محتوا را ترسیم می‌کند.



در اینجا، یک صفحه HTML توسط یک تابع کدگذاری محتوای gzip کدگذاری می‌شود تا بدن کوچک‌تر و فشرده‌تری تولید کند. بدن فشرده شده که با رمزگذاری gzip پرچم گذاری شده است، در سراسر شبکه ارسال می‌شود. کلاینت دریافت کننده موجودیت را با استفاده از رمزگشای gzip از حالت فشرده خارج می‌کند.

این قطعه پاسخ، نمونه دیگری از یک پاسخ رمزگذاری شده (یک تصویر فشرده) را نشان می‌دهد:

```
HTTP/1.1 200 OK
Date: Fri, 05 Nov 1999 22:35:15 GMT
Server: Apache/1.2.4
Content-Length: 6096
Content-Type: image/gif
Content-Encoding: gzip
[...]
```

¹⁰ content-encoding



توجه داشته باشید که هدر Content-Type می‌تواند و باید همچنان در پیام وجود داشته باشد. این فرمت اصلی موجودیت را توصیف می‌کند - اطلاعاتی که ممکن است برای نمایش موجودیت پس از رمزگشایی لازم باشد. به یاد داشته باشید که هدر Content-Length اکنون طول بدنه کدگذاری شده را نشان می‌دهد.

Content-Encoding Types

HTTP چند نوع استاندارد کدگذاری محتوا را تعریف می‌کند و اجازه می‌دهد تا کدگذاری‌های اضافی^{۱۱} به عنوان رمزگذاری پسوند^{۱۲} اضافه شوند. کدگذاری‌ها از طریق IANA استاندارد می‌شوند، که یک توکن منحصر به فرد به هر الگوریتم رمزگذاری محتوا اختصاص می‌دهد. هدر Content-Encoding از این مقادیر نشانه استاندارد شده برای توصیف الگوریتم مورد استفاده در کدگذاری استفاده می‌کند.

برخی از نشانه‌های رایج کدگذاری محتوا در جدول زیر فهرست شده‌اند.

Content-encoding value	Description
gzip	Indicates that the GNU zip encoding was applied to the entity. ^a
compress	Indicates that the Unix file compression program has been run on the entity.
deflate	Indicates that the entity has been compressed into the zlib format. ^b
identity	Indicates that no encoding has been performed on the entity. When a Content-Encoding header is not present, this can be assumed.

a RFC 1952 describes the gzip encoding.

b RFCs 1950 and 1951 describe the zlib format and deflate compression.

کدگذاری‌های gzip، compress و deflate الگوریتم‌های فشرده‌سازی بدون تلفات هستند که برای کاهش اندازه پیام‌های ارسالی بدون از دست دادن اطلاعات استفاده می‌شوند. از این میان، gzip به طور معمول موثرترین الگوریتم فشرده سازی است و بیشترین استفاده را دارد.

Accept-Encoding Headers

البته ما نمی‌خواهیم سرورهایی محتوا را به گونه‌ای کدگذاری کنند که کلاینت نتواند فرآیند رمزگشایی را بدرستی انجام دهد. برای جلوگیری از استفاده سرورها از کدگذاری‌هایی که سرویس گیرنده پشتیبانی نمی‌کند، سرویس گیرنده لیستی از کدگذاری‌های محتوای پشتیبانی شده را در هدر درخواست Accept-Encoding ارسال می‌کند.

^{۱۱} Additional Encodings

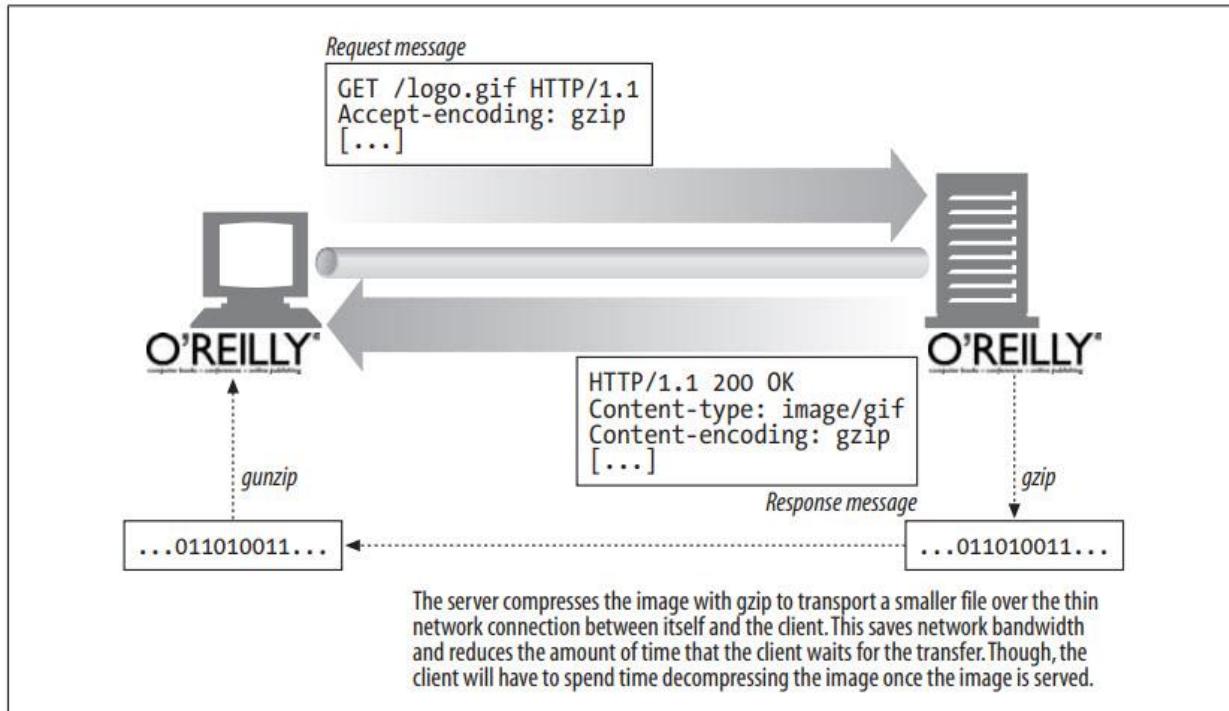
^{۱۲} Extension Encodings





اگر درخواست HTTP حاوی هدر Accept-Encoding نباشد، سرور می‌تواند فرض کند که کلاینت هر کدگذاری را می‌پذیرد (معادل *.(Accept-Encoding: *).

شکل زیر نمونه‌ای از Accept-Encoding را در تراکنش HTTP نشان می‌دهد.



فیلد Accept-Encoding حاوی لیستی از کدهای پشتیبانی شده جدا شده با کاما است. در اینجا چند نمونه از آن قرار داده شده است:

```
Accept-Encoding: compress, gzip
Accept-Encoding:
Accept-Encoding: *
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

کلاینت‌ها می‌توانند با پیوست کردن مقادیر Q (quality) به عنوان پارامتر به هر کدگذاری‌های ترجیحی را نشان دهند. مقادیر Q می‌توانند از ۰.۰ تغییر باشد، که نشان می‌دهد کلاینت کدگذاری مرتبط را نمی‌خواهد، تا ۱.۰، که کدگذاری ترجیحی را نشان می‌دهد. نشانه "*" به معنای "هرچیز دیگری" است. فرآیند انتخاب کدگذاری محتوا برای اعمال، بخشی از فرآیند کلی‌تر تصمیم‌گیری برای ارسال محتوا به کلاینت در پاسخ است.





کد رمزگذاری هویت^{۱۳} فقط می‌تواند در هدر Accept-Encoding وجود داشته باشد و توسط کلاینت‌ها برای تعیین اولویت نسبی نسبت به سایر الگوریتم‌های رمزگذاری محتوا استفاده می‌شود.

Transfer Encoding and Chunked Encoding

در بخش قبلی در مورد رمزگذاری محتوا بحث شد - تبدیل‌های برگشت‌پذیر اعمال شده^{۱۴} در بدن پیام، رمزگذاری محتوا به شدت با جزئیات فرمت محتوا خاص مرتبط است. به عنوان مثال، ممکن است یک فایل متنی را با gzip فشرده کنید، اما یک فایل JPEG را نه، زیرا JPEG‌ها به خوبی با gzip فشرده نمی‌شوند.

این بخش در مورد کدگذاری‌های انتقال بحث می‌کند. کدگذاری‌های انتقال^{۱۵} نیز تبدیل‌های برگشت‌پذیری هستند که روی بدن موجودیت انجام می‌شوند، اما به دلایل معماری اعمال می‌شوند و مستقل از قالب محتوا هستند. شما یک کدگذاری انتقال را برای تغییر روش انتقال داده پیام در سراسر شبکه اعمال می‌کنید (شکل زیر).

Content-encoded response

HTTP/1.0 200 OK
Content-encoding: gzip *Normal header block*
Content-type: text/html
[...]
[encoded message] *Normal entity (just encoded)*

A Content-encoded message just encodes the entity section of the message. With Transfer-encoded messages the encoding is a function of the entire message, changing the structure of the message itself.

Transfer-encoded response

HTTP/1.1 200 OK
Transfer-encoding: chunked *Basic header*

10
abcdefghijklmnopqrstuvwxyz *Encoded blocks*
1
a

Safe Transport

از لحاظ تاریخی، کدگذاری‌های انتقال در پروتکل‌های دیگر وجود دارد تا "انتقال ایمن" پیام‌ها را در یک شبکه فراهم کند. مفهوم حمل و نقل ایمن تمرکز متفاوتی برای HTTP دارد، جایی که زیرساخت حمل و نقل استاندارد

¹³ identity encoding token

¹⁴ reversible transformations applied

¹⁵ transfer encodings





شده و بخشنده‌تر^{۱۶} است. در HTTP، تنها چند دلیل وجود دارد که چرا انتقال متن پیام می‌تواند باعث ایجاد مشکل شود. دو مورد از این‌ها عبارتند از:

Unknown size

برخی از برنامه‌های Gateway و کدگذارهای محتوا قادر به تعیین اندازه نهایی متن پیام بدون تولید محتوا نیستند. اغلب، این سرورها مایلند قبل از مشخص شدن اندازه، ارسال داده را شروع کنند. از آنجایی که هدر HTTP به نیاز دارد تا قبل از داده‌ها قرار گیرد، برخی از سرورها از یک کدگذاری انتقال برای ارسال داده‌ها با یک footer خاص که پایان داده را نشان می‌دهد، استفاده می‌کنند.

Security

شما ممکن است از یک کدگذاری انتقال برای درهم کردن محتوای پیام قبل از ارسال آن در یک شبکه حمل و نقل مشترک استفاده کنید. با این حال، به دلیل محبوبیت طرح‌های امنیتی لایه انتقال مانند SSL، امنیت کدگذاری انتقال چندان رایج نیست.

Transfer-Encoding Headers

فقط دو هدر تعریف شده برای توصیف و کنترل کدگذاری انتقال وجود دارد:

Transfer-Encoding

به گیرنده می‌گوید چه کدگذاری روی پیام انجام شده است تا به طور ایمن منتقل شود.

TE

در هدر درخواست استفاده می‌شود تا به سرور بگوید چه کدگذاری‌های انتقال برنامه افزودنی^{۱۷} برای استفاده مناسب است.

در مثال زیر، درخواست استفاده از هدر TE می‌کند تا به سرور بگوید که کدگذاری تکه‌شده^{۱۸} را می‌پذیرد (که اگر یک برنامه HTTP 1.1 باشد باید آن را بپذیرد) و مایل است Trailer‌های انتهای پیام‌های کدگذاری شده را بپذیرد:

¹⁶ forgiving

¹⁷ extension transfer encoding

¹⁸ chunked encoding





```
GET /new_products.html HTTP/1.1
Host: www.joes-hardware.com
User-Agent: Mozilla/4.61 [en] (WinNT; I)
TE: trailers, chunked
...
```

پاسخ شامل یک هدر Transfer-Encoding است تا به گیرنده بگوید که پیام با کدگذاری تکه‌تکه شده کدگذاری شده است:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Server: Apache/3.0
...

```

بعد از این هدر اولیه، ساختار پیام تغییر می‌کند.

همه مقادیر کدگذاری انتقال به حروف بزرگ و کوچک حساس هستند. HTTP/1.1 از مقادیر کدگذاری انتقال در فیلد هدر TE و در فیلد هدر Transfer-Encoding استفاده می‌کند. آخرین مشخصات HTTP تنها یک کدگذاری انتقال، که کدگذاری تکه‌ای است را تعریف می‌کند.

هدر TE، مانند هدر Accept-Encoding، می‌تواند مقادیر Q را برای توصیف اشکال ترجیحی کدگذاری انتقال داشته باشد. با این حال، مشخصات HTTP/1.1 ارتباط یک مقدار 0.0 Q را با کدگذاری تکه‌ای ممنوع می‌کند.

های آینده به HTTP ممکن است نیاز به کدگذاری‌های انتقال اضافی را ایجاد کند. اگر و زمانی که این اتفاق می‌افتد، کدگذاری انتقال تکه شده باید همیشه در بالای کدگذاری‌های انتقال برنامه افزودنی اعمال شود. این تضمین می‌کند که داده‌ها از طریق برنامه‌های HTTP/1.1 که کدگذاری تکه شده را درک می‌کنند، اما دیگر کدگذاری‌های انتقال را درک نمی‌کنند، «تونل» می‌شوند.

Chunked Encoding

کدگذاری تکه‌ای، پیام‌ها را به تکه‌هایی با اندازه مشخص تقسیم می‌کند. هر تکه یکی پس از دیگری ارسال می‌شود و نیازی به دانستن اندازه کامل پیام قبل از ارسال آن نیست.

توجه داشته باشید که کدگذاری تکه‌ای شکلی از کدگذاری انتقال است و بنابراین یک ویژگی پیام است نه بدنه. کدگذاری چند بخشی که قبلاً در این فصل توضیح داده شد، یک ویژگی بدنه است و کاملاً جدا از کدگذاری تکه‌ای است.





Chunking and persistent connections

هنگامی که ارتباط بین کلاینت و سرور پایدار نباشد، کلاینتها نیازی به دانستن اندازه بدن‌های که می‌خوانند ندارند – آن‌ها انتظار دارند تا زمانی که سرور اتصال را بیندد، بدن را بخوانند.

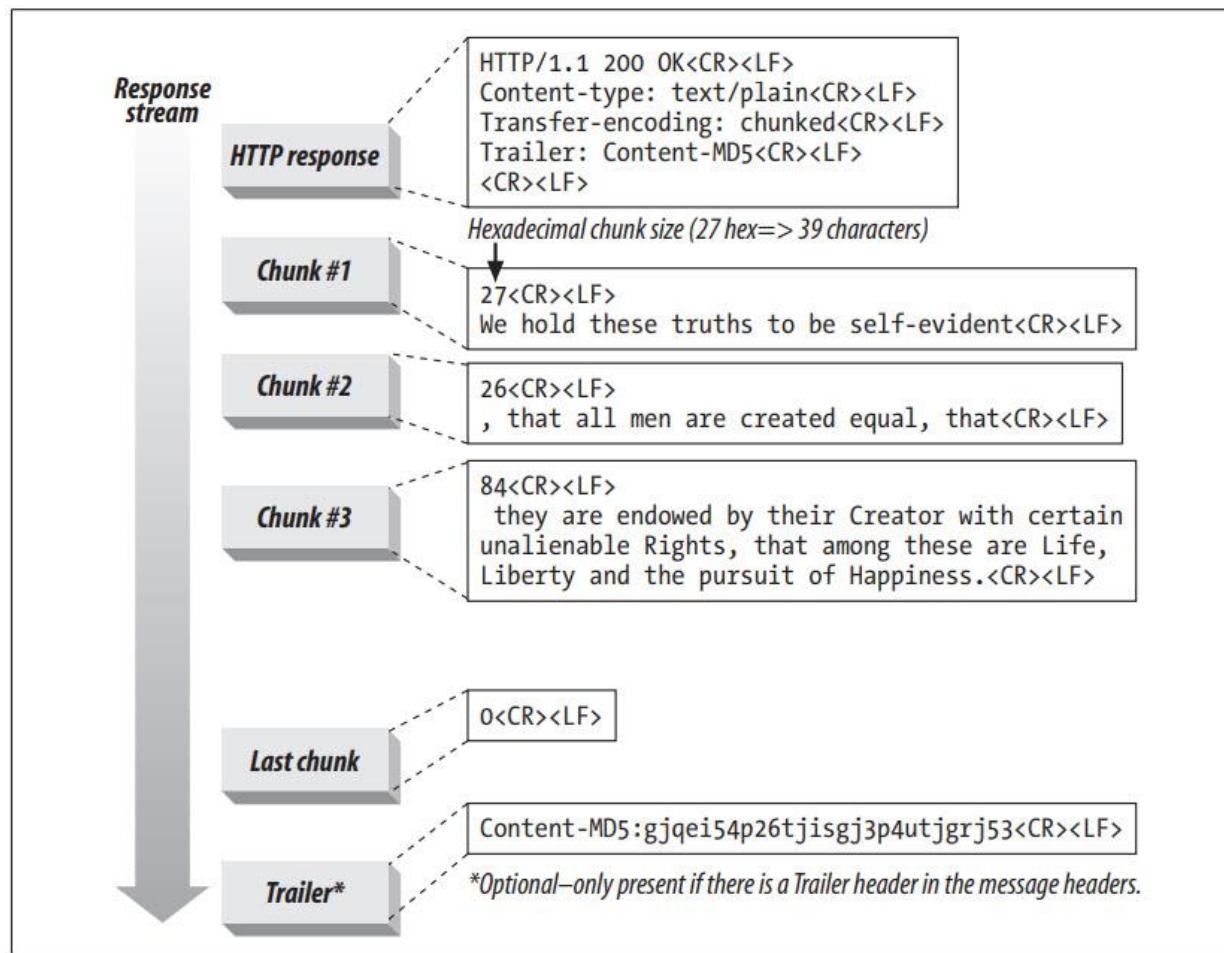
با اتصالات مداوم، اندازه بدن باید شناخته شده و در هدر Content-Length ارسال شود تا بتوان بدن را نوشت. هنگامی که محتوا به صورت پویا در یک سرور ایجاد می‌شود، ممکن است قبل از ارسال، نتوان از طول بدن آن مطلع شد.

کدگذاری تکه‌ای را حلی را برای این معضل فراهم می‌کند و به سرورها اجازه می‌دهد که بدن را به صورت تکه‌ها ارسال کنند و فقط اندازه هر تکه را مشخص کنند. همانطور که بدن به صورت پویا تولید می‌شود، یک سرور می‌تواند بخشی از آن را بافر کند، اندازه و قطعه آن را ارسال کند و سپس این فرآیند را تا زمانی که کل بدن ارسال شود تکرار کند. سرور می‌تواند انتهای بدن را با یک قطعه به اندازه صفر سیگنال دهد و همچنان اتصال را باز و آماده برای پاسخ بعدی نگه دارد.

کدگذاری تکه‌ای نسبتاً ساده است. شکل زیر آناتومی اولیه یک پیام تکه تکه شده را نشان می‌دهد. این مسیر با یک بلوک هدر پاسخ HTTP اولیه و به دنبال آن جریانی از تکه‌ها شروع می‌شود. هر قطعه حاوی مقدار طول و داده‌های آن قطعه است. مقدار طول به شکل هگزادسیمال است و با CRLF از داده‌های تکه جدا می‌شود. اندازه داده‌های تکه شده در بایت اندازه گیری می‌شود و نه دنباله CRLF بین مقدار طول و داده و نه دنباله CRLF در انتهای قطعه را شامل می‌شود. آخرین تکه خاص است – طول آن صفر بوده که به معنای "پایان بدن" است.

یک کلاینت همچنین ممکن است داده‌های تکه تکه شده را به یک سرور ارسال کند. از آنجایی که کلاینت از قبل نمی‌داند که آیا سرور کدگذاری تکه‌ای را می‌پذیرد (سرورها هدرهای TE را در پاسخها به کلاینتها ارسال نمی‌کنند)، باید برای سرور آماده شود تا درخواست تکه تکه شده را با یک پاسخ ۴۱۱ طول مورد نیاز رد کند.





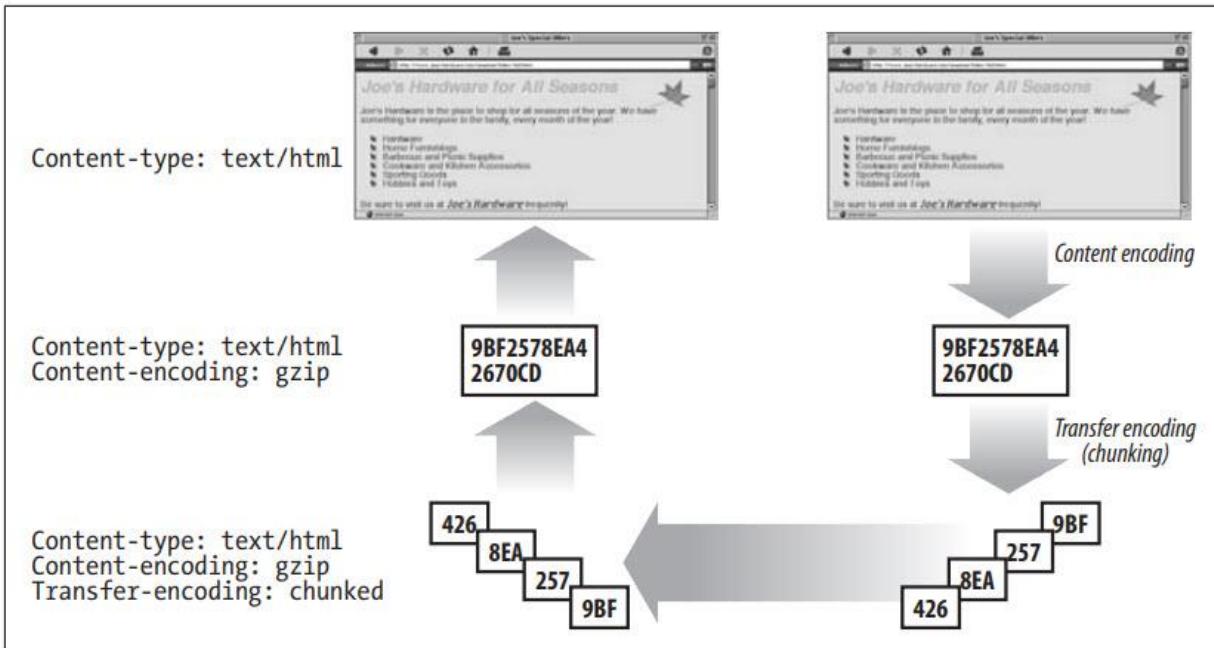
Trailers in chunked messages

هنگامی که یک کدگذاری انتقال به بدن پیام اعمال می‌شود، چند قانون باید رعایت شود:

- مجموعه کدگذاری‌های انتقال باید شامل «chunked» باشد. تنها استثنای در صورتی است که پیام با بستن اتصال خاتمه یابد.
- هنگامی که از کدگذاری انتقال تکه‌ای¹⁹ استفاده می‌شود، لازم است که آخرین کدگذاری انتقال اعمال شده در متن پیام باشد.
- کدگذاری انتقال تکه شده نباید بیش از یک بار روی متن پیام اعمال شود.

¹⁹ chunked transfer encoding





این قوانین به گیرنده اجازه می‌دهد تا طول انتقال پیام را تعیین کند.

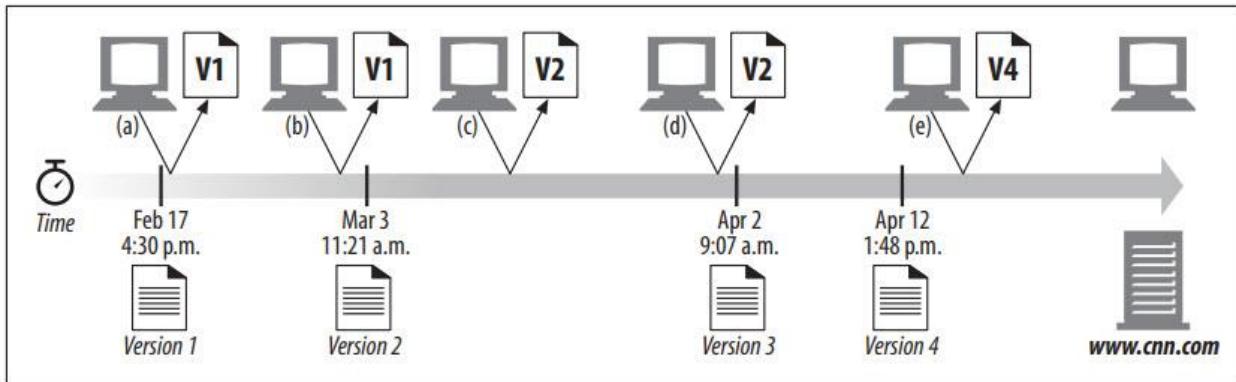
کدگذاری انتقال یک ویژگی نسبتاً جدید HTTP است که در نسخه 1.1 معرفی شده است. سرورهایی که کدگذاری‌های انتقال را پیاده‌سازی می‌کنند باید مراقب باشند که پیام‌های کدگذاری شده با انتقال را به برنامه‌های غیر HTTP/1.1 ارسال نکنند. به همین ترتیب، اگر یک سرور یک پیام کدگذاری شده با انتقال دریافت کند که قادر به درک آن نیست، باید با کد وضعیت 501 Unimplemented پاسخ دهد. با این حال، همه برنامه‌های HTTP/1.1 باید حداقل از کدگذاری تکه‌ای پشتیبانی کنند.

Time-Varying Instances

اشیاء وب ثابت نیستند. URL یکسان می‌تواند در طول زمان به نسخه‌های مختلف یک شی اشاره کند. صفحه اصلی CNN را به عنوان مثال در نظر بگیرید - رفتن به "http://www.cnn.com" چندین بار در روز ممکن است منجر به بازگشت یک صفحه کمی متفاوت در هر بار شود.

صفحه اصلی CNN را به عنوان یک شی و نسخه‌های مختلف آن را به عنوان نمونه‌های مختلف شی در نظر بگیرید (شکل زیر را ببینید).





کلاینت در شکل چندین بار یک منبع (URL) یکسان را درخواست می‌کند، اما با تغییر در طول زمان، نمونه‌های مختلفی از منبع را دریافت می‌کند. در زمان (a) و (b) مصدق یکسانی دارد. در زمان (c) یک نمونه متفاوت دارد.

پروتکل HTTP عملیاتی را برای کلاسی از درخواست‌ها و پاسخ‌ها، به نام **Instance Manipulations**، مشخص می‌کند که بر روی نمونه‌هایی از یک شی عمل می‌کنند. دو روش اصلی دستکاری نمونه، درخواست‌های محدوده و کدگذاری دلتا هستند. هر دوی این روش‌ها به کلاینت‌ها نیاز دارند که بتوانند نسخه دقیق منبعی را که در اختیار دارند (در صورت وجود) شناسایی کرده و نمونه‌های جدید را به صورت مشروط درخواست کنند. این مکانیسم‌ها بعداً در این فصل مورد بحث قرار می‌گیرند.

Validators and Freshness

مجدد به شکل بالا نگاه کنید. کلاینت در ابتدا یک کپی از منبع ندارد، بنابراین درخواستی را به سرور ارسال می‌کند و آن را درخواست می‌کند. سرور با نسخه ۱ منبع پاسخ می‌دهد. کلاینت اکنون می‌تواند این کپی را در Cache نگه دارد، اما برای چه مدت؟

هنگامی که سند در سرویس گیرنده منقضی شد (یعنی زمانی که کلاینت دیگر نمی‌تواند کپی آن را یک نسخه معتبر در نظر بگیرد)، باید یک نسخه جدید از سرور درخواست کند. با این حال، اگر سند در سرور تغییر نکرده باشد، کلاینت نیازی به دریافت مجدد آن ندارد - فقط می‌تواند به استفاده از نسخه Cache خود ادامه دهد.

این درخواست ویژه، که Conditional Request نامیده می‌شود، مستلزم آن است که سرویس گیرنده با استفاده از اعتبارسنجی به سرور بگوید که در حال حاضر کدام نسخه را دارد و فقط در صورتی که نسخه فعلی آن دیگر معتبر نیست، بخواهد کپی ارسال شود. بیایید به سه مفهوم کلیدی - تازگی، اعتبار سنجی و شرطی - با جزئیات بیشتری نگاه کنیم.





Freshness

از سرورها انتظار می‌رود که اطلاعاتی در مورد مدت زمانی که کلاینت‌ها می‌توانند محتوای خود را در Cache نگه دارند و آن را تازه در نظر بگیرند، به کلاینت‌ها ارائه دهند. سرورها می‌توانند این اطلاعات را با استفاده از یکی از دو هدر ارائه دهند: Cache-Control و Expires.

هدر Expires تاریخ و زمان دقیقی را مشخص می‌کند که سند "Expires" می‌شود—زمانی که دیگر نمی‌توان آن را Fresh در نظر گرفت. هدر Syntax Expires به صورت زیر است:

Expires: Sun Mar 18 23:59:59 GMT 2001

برای اینکه یک کلاینت و سرور از هدر Expires به درستی استفاده کنند، ساعت آن‌ها باید همگام باشد. این همیشه آسان نیست، زیرا ممکن است هیچکدام از یک پروتکل همگام سازی ساعت مانند پروتکل زمان شبکه (NTP) را استفاده نکنند. مکانیزمی که انقضا را با استفاده از زمان نسبی تعریف می‌کند مفیدتر است. هدر Cache-Control را می‌توان برای تعیین Maximum Age برای یک سند در ثانیه استفاده کرد - کل مدت زمانی که سند از سرور خارج شده است. Age به همگام سازی ساعت بستگی ندارد و بنابراین احتمالاً نتایج دقیق‌تری به همراه دارد.

هدر Cache-Control در واقع بسیار قدرتمند است. این می‌تواند توسط سرورها و کلاینت‌ها برای توصیف تازگی و با استفاده از دستورالعمل‌های بیشتر به جای تعیین Age یا زمان انقضا استفاده شود. جدول زیر برخی از دستورالعمل‌هایی را که می‌توانند هدر Cache-Control را همراهی کنند، فهرست می‌کند.



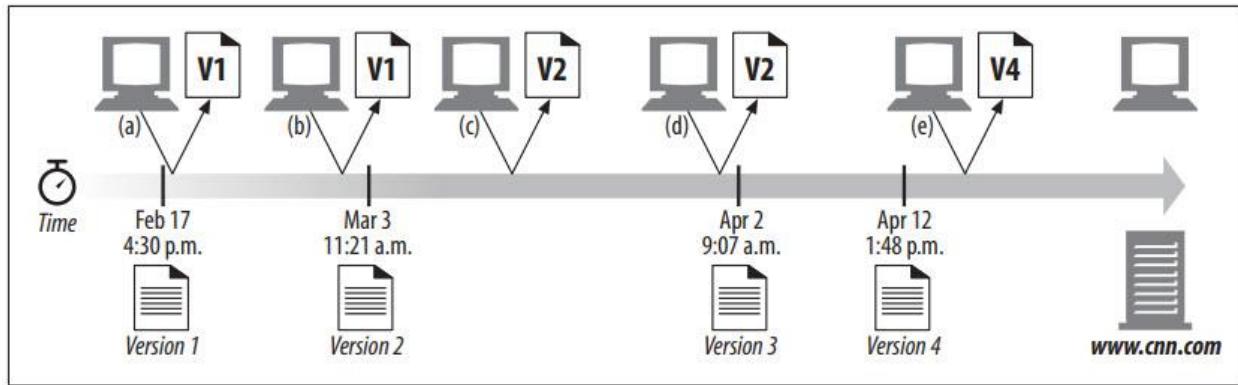


Directive	Message type	Description
no-cache	Request	Do not return a cached copy of the document without first revalidating it with the server.
no-store	Request	Do not return a cached copy of the document. Do not store the response from the server.
max-age	Request	The document in the cache must not be older than the specified age.
max-stale	Request	The document may be stale based on the server-specified expiration information, but it must not have been expired for longer than the value in this directive.
min-fresh	Request	The document's age must not be more than its age plus the specified amount. In other words, the response must be fresh for at least the specified amount of time.
no-transform	Request	The document must not be transformed before being sent.
only-if-cached	Request	Send the document only if it is in the cache, without contacting the origin server.
public	Response	Response may be cached by any cache.
private	Response	Response may be cached such that it can be accessed only by a single client.
no-cache	Response	If the directive is accompanied by a list of header fields, the content may be cached and served to clients, but the listed header fields must first be removed. If no header fields are specified, the cached copy must not be served without revalidation with the server.
no-store	Response	Response must not be cached.
no-transform	Response	Response must not be modified in any way before being served.
must-revalidate	Response	Response must be revalidated with the server before being served.
proxy-revalidate	Response	Shared caches must revalidate the response with the origin server before serving. This directive can be ignored by private caches.
max-age	Response	Specifies the maximum length of time the document can be cached and still considered fresh.
s-max-age	Response	Specifies the maximum age of the document as it applies to shared caches (overriding the max-age directive, if one is present). This directive can be ignored by private caches.

Conditionals and Validators

هنگامی که یک نسخه Cache درخواست می‌شود، و دیگر Cache Fresh نیست، باید مطمئن شود که یک نسخه جدید دارد. Cache می‌تواند کپی فعلی را از سرور مبدا دریافت کند، اما در بسیاری از موارد، سند روی سرور همچنان مانند نسخه قدیمی موجود در Cache است. ما این را در بخش b شکل زیر می‌بینیم (جهت بررسی، تصویر مجدد آورده شده است).





ممکن است کپی ذخیره شده منقضی شده باشد، اما محتوای سرور همچنان مانند محتوای Cache است. اگر یک Cache همیشه سند یک سرور را واکشی می‌کند، حتی اگر همان نسخه Cache منقضی شده باشد، پهنهای باند شبکه را هدر می‌دهد، Load غیرضروری را روی Cache و سرور قرار می‌دهد و همه چیز را کند می‌کند.

برای رفع این مشکل، HTTP با استفاده از درخواست‌های ویژه‌ای که درخواست‌های شرطی^{۲۰} نامیده می‌شوند، راهی را برای کلاینت‌ها فراهم می‌کند که فقط در صورت تغییر منبع، درخواست کپی کنند. درخواست‌های شرطی پیام‌های درخواستی HTTP معمولی هستند، اما تنها در صورتی انجام می‌شوند که یک شرط خاص درست باشد. به عنوان مثال، یک Cache ممکن است پیام GET مشروط زیر را به سرور ارسال کند و از آن بخواهد فایل announce.html را فقط در صورتی که فایل، از ۲۹ ژوئن ۲۰۰۲ اصلاح شده باشد (تاریخ آخرین تغییر سند ذخیره شده توسط نویسنده) ارسال کند:

GET /announce.html HTTP/1.0

If-Modified-Since: Sat, 29 Jun 2002, 14:30:00 GMT

درخواست‌های مشروط توسط هدرهای شرطی که با "If-" شروع می‌شوند، پیاده سازی می‌شوند. در مثال بالا، هدر شرطی If-Modified-Since است. یک هدر شرطی به یک متدازنگار می‌دهد فقط در صورتی اجرا شود که شرط درست باشد. اگر شرط درست نباشد، سرور یک کد خطای HTTP را برمی‌گرداند.

هر شرطی روی یک اعتبارسنجی خاص کار می‌کند. اعتبار سنجی یک ویژگی خاص از نمونه سند است که آزمایش می‌شود. از نظر مفهومی، می‌توانید به اعتبارسنجی مانند شماره سریال، شماره نسخه یا آخرین تاریخ تغییر یک سند فکر کنید. یک کلاینت عاقل در بخش b شکل بالا یک درخواست اعتبار سنجی مشروط به سرور ارسال می‌کند و می‌گوید: «منبع را فقط در صورتی برای من ارسال کنید که دیگر نسخه ۱ نباشد. من

²⁰ conditional requests





نسخه ۱ را دارم.» ما در فصل ۷ در مورد اعتبار سنجی مجدد Cache شرطی بحث کردیم، اما در این فصل جزئیات اعتبار سنجی موجودیت‌ها را با دقت بیشتری مطالعه خواهیم کرد.

هدر شرطی If-Modified-Since آخرین تاریخ اصلاح یک نمونه سند را آزمایش می‌کند، بنابراین می‌گوییم که آخرین تاریخ اصلاح شده اعتبار سنج است. هدر شرطی ETag مقدار If-None-Match یک سند را آزمایش می‌کند، که یک کلمه کلیدی خاص یا برچسب شناسایی نسخه مرتبط با موجودیت است. Last-Modified و ETag دو اعتبار سنجی اولیه هستند که توسط HTTP استفاده می‌شوند. جدول زیر چهار هدر HTTP مورد استفاده برای درخواست‌های شرطی را فهرست می‌کند. در کنار هر هدر شرطی، نوع اعتبار سنجی مورد استفاده با هدر وجود دارد.

Request type	Validator	Description
If-Modified-Since	Last-Modified	Send a copy of the resource if the version that was last modified at the time in your previous Last-Modified response header is no longer the latest one.
If-Unmodified-Since	Last-Modified	Send a copy of the resource only if it is the same as the version that was last modified at the time in your previous Last-Modified response header.
If-Match	ETag	Send a copy of the resource if its entity tag is the same as that of the one in your previous ETag response header.
If-None-Match	ETag	Send a copy of the resource if its entity tag is different from that of the one in your previous ETag response header.

اعتبار سنجی‌ها را به دو دسته تقسیم می‌کند: Weak Validator و Strong Validator. Weak Validator ها ممکن است همیشه نمونه‌ای از یک منبع را به طور منحصر به فرد شناسایی نکنند. Strong Validator ها باید نمونه‌ای از Weak Validator، اندازه شیء بر حسب بایت باشد. محتوای منبع ممکن است تغییر کند حتی اگر اندازه آن ثابت بماند، بنابراین یک Byte-count Validator فرضی تنها به طور ضعیف نشان دهنده تغییر است. با این حال، یک Checksum رمزگاری از محتویات منبع (مانند MD5)، یک Strong Validator است که با تغییر سند تغییر می‌کند.

آخرین زمان اصلاح شده یک Weak Validator در نظر گرفته می‌شود، زیرا اگرچه زمانی را مشخص می‌کند که در آن منبع آخرین اصلاح شده است، اما آن زمان را با دقت حداقل یک ثانیه مشخص می‌کند. از آنجا که یک منبع می‌تواند چندین بار در یک ثانیه تغییر کند، و از آنجا که سرورها می‌توانند هزاران درخواست را در ثانیه ارائه دهند، تاریخ آخرین اصلاح ممکن است همیشه منعکس کننده تغییرات نباشد. هدر ETag یک Strong Validator در نظر گرفته می‌شود، زیرا سرور می‌تواند هر بار که مقداری تغییر می‌کند، یک مقدار متمایز در هدر ETag قرار دهد. اعداد نسخه و Digest Checksum ها کاندیدهای خوبی برای هدر





هستند، اما می‌توانند حاوی هر متن دلخواه باشند. هدرهای ETag انعطاف‌پذیر هستند. آن‌ها مقادیر متن دلخواه (tags) را می‌گیرند و می‌توانند برای ابداع انواع استراتژی‌های اعتبار سنجی کلاینت و سرور استفاده شوند.

ممکن است گاهی اوقات کلاینت‌ها و سرورها بخواهند یک نسخه آزادتر از اعتبار سنجی entity-tag را اتخاذ کنند. برای مثال، ممکن است یک سرور بخواهد تغییرات زیبایی را در یک سند ذخیره شده بزرگ و محبوب بدون ایجاد انتقال انبوه در هنگام اعتبار سنجی مجدد Cache ایجاد کند. در این مورد، سرور ممکن است یک برچسب موجودیت "Weak" را با پیشوند برچسب "W/" اصطلاحاً Advertise کند. یک تگ موجودیت Weak تنها زمانی باید تغییر کند که موجودیت مرتبط به صورت معناداری تغییر کند. هر زمان که مقدار موجودیت مرتبط به هر نحوی تغییر کرد، یک تگ موجودیت Strong باید تغییر کند.

مثال زیر نشان می‌دهد که چگونه یک کلاینت با استفاده از یک تگ موجودیت Weak، می‌تواند با یک سرور فرآیند Revalidate را انجام دهد. سرور فقط در صورتی یک بدنه را برمی‌گرداند که محتوا به روشنی معنی دار از نسخه ۴.۰ سند تغییر کند:

GET /announce.html HTTP/1.1

If-None-Match: W/"v4.0"

به طور خلاصه، هنگامی که کلاینت‌ها بیش از یک بار به یک منبع دسترسی دارند، ابتدا باید تعیین کنند که آیا نسخه فعلی آن‌ها هنوز Fresh هست یا خیر. اگر اینطور نیست، باید آخرین نسخه را از سرور دریافت کنند. برای جلوگیری از دریافت یک کپی مشابه در صورتی که منبع تغییر نکرده باشد، کلاینت‌ها می‌توانند درخواست‌های مشروط را به سرور ارسال کنند و اعتبار سنجی‌هایی را مشخص کنند که به طور منحصر به فرد کپی‌های فعلی آن‌ها را شناسایی می‌کنند. سرورها فقط در صورتی کپی از منبع را ارسال می‌کنند که با نسخه کلاینت متفاوت باشد. برای جزئیات بیشتر در مورد اعتبار سنجی مجدد Cache Processing Steps، لطفاً به فصل ۷ مراجعه کنید.

Range Requests

اکنون می‌دانیم که یک کلاینت تنها در صورتی می‌تواند از سرور بخواهد که منبعی را برای آن ارسال کند که نسخه کلاینت از منبع، دیگر معتبر نباشد. HTTP فراتر می‌رود: به کلاینت‌ها اجازه می‌دهد تا در واقع فقط بخش یا محدوده‌ای از یک سند را درخواست کنند.





تصور کنید که سه چهارم راه را برای دانلود آخرین نرم افزار مورد نظر خود از طریق لینک مودم، آهسته طی کرده‌اید و یک مشکل شبکه باعث قطع شدن اتصال شما می‌شود. شما مدتی منتظر بودید تا دانلود کامل شود و حالا باید همه چیز را از اول شروع کنید، به امید اینکه دوباره همان اتفاق تکرار نشود.

با Range Request ها، یک سرویس گیرنده HTTP می‌تواند بارگیری یک موجودیت را با درخواست محدوده یا بخشی از موجودی که موفق به دریافت آن نشده است از سر بگیرد (مشروط بر اینکه شی در سرور مبدا بین زمانی که کلاینت برای اولین بار آن را درخواست کرد و Range Request بعدی آن تغییر نکرده باشد). مثلا:

```
GET /bigfile.html HTTP/1.1
Host: www.joes-hardware.com
Range: bytes=4000-
User-Agent: Mozilla/4.61 [en] (WinNT; I)
...
...
```

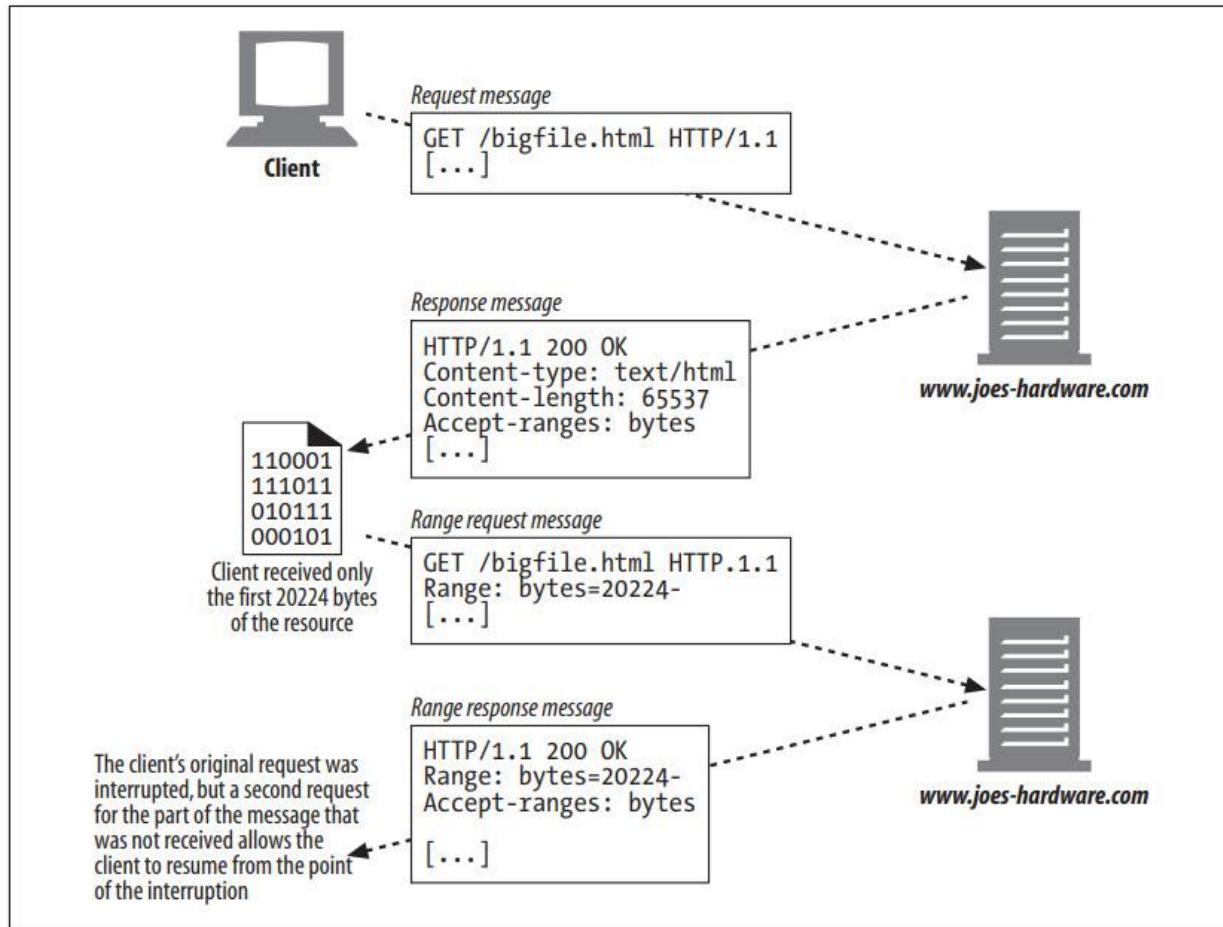
در این مثال، کلاینت باقیمانده سند را پس از ۴۰۰۰ بایت اول درخواست می‌کند (بایتهای پایانی لازم نیست مشخص شوند، زیرا ممکن است اندازه سند برای درخواست کننده مشخص نباشد). Range های این فرم را می‌توان برای یک درخواست ناموفق استفاده کرد که در آن کلاینت ۴۰۰۰ بایت اول را قبل از شکست دریافت کرده است. هدر Range همچنین می‌تواند برای درخواست چندین محدوده استفاده شود (محدوده‌ها می‌توانند به هر ترتیبی مشخص شوند و ممکن است همپوشانی داشته باشند) - برای مثال، تصور کنید یک کلاینت به چندین سرور به طور همزمان متصل می‌شود و محدوده‌های مختلفی از یک سند را از سرورهای مختلف درخواست می‌کند تا سرعت دانلود کلی برای سند در مواردی که کلاینت‌های چندین محدوده را در یک درخواست واحد درخواست می‌کنند، پاسخ‌ها به صورت یک موجودیت واحد، با یک بدنی چند قسمتی و یک هدر Content-Type: multipart/byteranges برمی‌گردند.

همه سرورها Range Request ها را نمی‌پذیرند، اما بسیاری از آن‌ها این کار را می‌کنند. سرورها می‌توانند با گنجاندن هدر Accept-Ranges در پاسخ‌های خود، به کلاینت‌ها اعلام کنند که Range ها را می‌پذیرند. مقدار این هدر یک واحد اندازه گیری است که معمولاً بایت می‌باشد. به عنوان مثال:

```
HTTP/1.1 200 OK
Date: Fri, 05 Nov 1999 22:35:15
Server: Apache/1.2.4
Accept-Ranges: bytes
...
...
```

شکل زیر نمونه‌ای از مجموعه تراکنش‌های HTTP را نشان می‌دهد که Range ها را شامل می‌شوند.





هدرهای Range به طور گسترده توسط نرم افزار مشترک اشتراک گذاری فایل نظیر به نظری برای دانلود همزمان بخش های مختلف فایل های چند رسانه ای از همتایان مختلف استفاده می شوند.

توجه داشته باشید که Range Request ها، یک کلاس از دستکاری های نمونه^{۲۱} هستند، زیرا آنها مبادله بین یک کلاینت و یک سرور برای یک نمونه خاص از یک شی هستند. یعنی Range Request یک کلاینت تنها در صورتی معنا دارد که کلاینت و سرور نسخه یکسانی از یک سند داشته باشند.

Delta Encoding

ما نسخه های مختلف یک صفحه وب را به عنوان نمونه های مختلف یک صفحه توصیف کردیم. اگر کلاینت یک نسخه منقضی شده از یک صفحه داشته باشد، آخرین نمونه صفحه را درخواست می کند. اگر سرور نمونه جدیدی از صفحه داشته باشد، آن را برای کلاینت ارسال نموده و نمونه جدید صفحه را حتی اگر فقط بخش کوچکی از صفحه واقعاً تغییر کرده باشد، ارسال می کند.

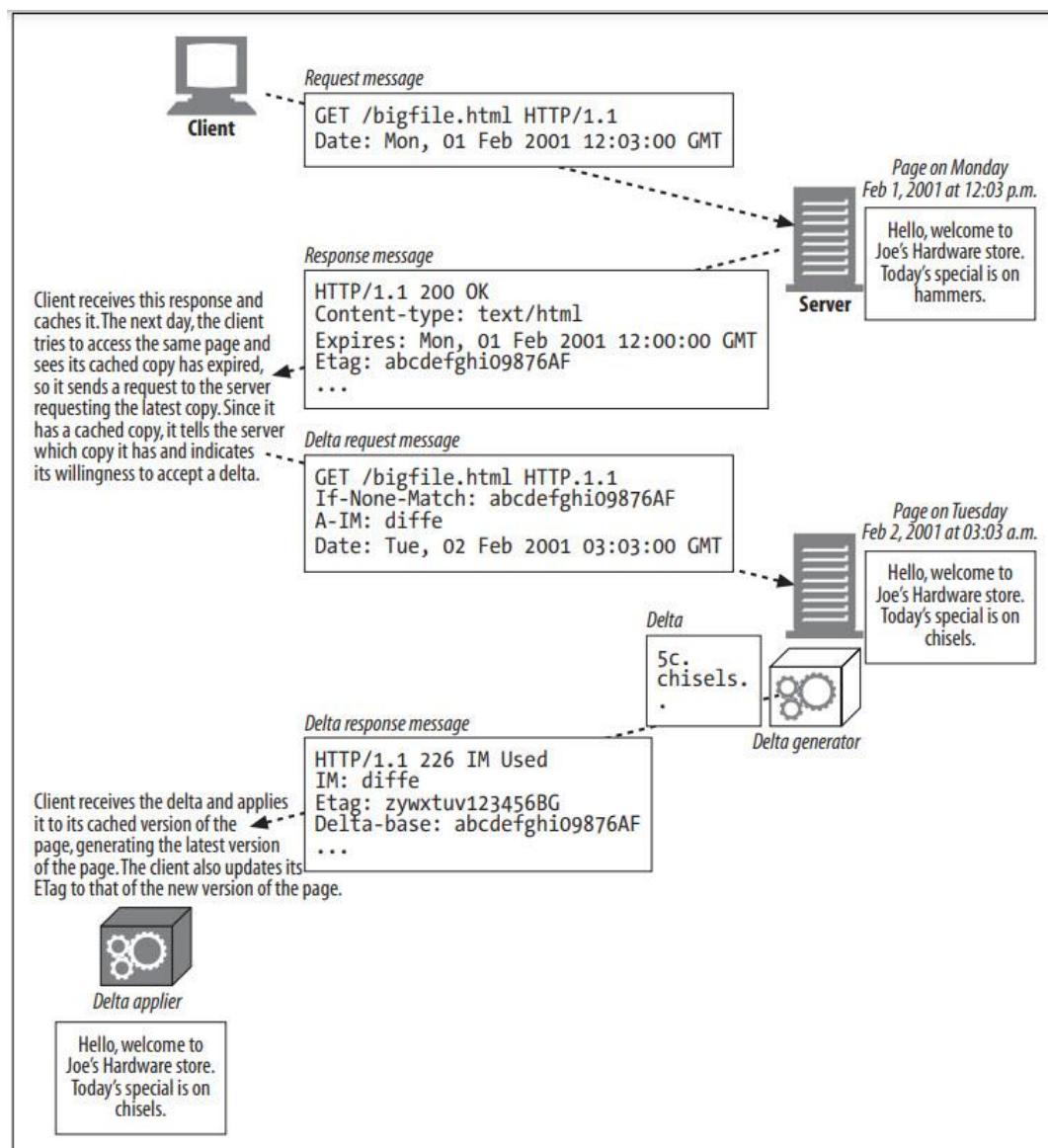
²¹ instance manipulations





به جای ارسال کل صفحه جدید، اگر سرور فقط تغییرات را به نسخه کلاینت از صفحه ارسال کند، کلاینت صفحه را سریعتر دریافت می‌کند (به شرطی که تعداد تغییرات کم باشد). کدگذاری دلتا یک Extension برای پروتکل HTTP است که با انتقال تغییرات به جای کل اشیاء، انتقال‌ها را بهینه می‌کند. کدگذاری دلتا نوعی دستکاری نمونه است، زیرا به کلاینت‌ها و سرورهایی که اطلاعات مربوط به نمونه‌های خاصی از یک شی را مبادله می‌کنند، متکی است. RFC 3229 کدگذاری دلتا را توصیف می‌کند.

شكل زیر مکانیسم درخواست، تولید، دریافت و بکارگیری یک سند با کد دلتا را به وضوح نشان می‌دهد.



کلاینت باید به سرور بگوید که کدام نسخه از صفحه را دارد، که مایل است یک دلتا از آخرین نسخه صفحه را بپذیرد، و چه الگوریتم‌هایی را برای اعمال آن دلتاهای در نسخه فعلی خود می‌شناسد.





سرور باید بررسی کند که آیا نسخه کلاینت از صفحه را دارد و چگونه دلتاها را از آخرین نسخه و نسخه کلاینت محاسبه کند (الگوریتم‌های مختلفی برای محاسبه تفاوت بین دو شی وجود دارد). سپس باید دلتا را محاسبه کند، آن را برای کلاینت ارسال کند، به کلاینت اطلاع دهد که در حال ارسال یک دلتا است و شناسه جدید را برای آخرین نسخه صفحه مشخص کند (زیرا این نسخه ای است که کلاینت پس از اعمال دلتا به نسخه قدیمی خود با آن به پایان می‌رسد).

کلاینت از شناسه منحصر به فرد برای نسخه صفحه خود در یک هدر **If-None-Match** استفاده می‌کند (که توسط سرور در پاسخ قبلی خود به کلاینت در هدر **ETag** ارسال شده است).

این روشی است که کلاینت به سرور می‌گوید: "اگر آخرین نسخه صفحه‌ای که دارید همان **ETag** را ندارد، آخرین نسخه صفحه را برای من ارسال کنید." پس فقط هدر **If-None-Match** باعث می‌شود که سرور آخرین نسخه کامل صفحه را برای کلاینت ارسال کند (اگر با نسخه کلاینت متفاوت بود).

با این حال، کلاینت می‌تواند به سرور بگوید که مایل است یک دلتای صفحه را با ارسال یک هدر **A-IM** بپذیرد. **A-IM** مخفف **Accept-Instance-Manipulation** است ("اوہ، اتفاقاً من برخی از اشکال دستکاری نمونه را قبول دارم، بنابراین اگر یکی از آن‌ها را اعمال کنید، مجبور نخواهید بود که سند کامل را برای من ارسال کنید."). در هدر **A-IM**، کلاینت الگوریتم‌هایی را که می‌داند چگونه اعمال شود، مشخص می‌کند تا آخرین نسخه یک صفحه را با یک نسخه قدیمی و یک دلتا تولید کند. سرور موارد زیر را ارسال می‌کند:

- یک کد پاسخ ویژه (IM ۲۲۶ استفاده شده) که به کلاینت می‌گوید که یک دستکاری نمونه از شی درخواست شده را برای آن ارسال می‌کند، نه خود شی کامل.
- یک هدر **IM** (مخفف **Instance-Manipulation**) که الگوریتم مورد استفاده برای محاسبه دلتا را مشخص می‌کند.
- هدر جدید **ETag**؛
- و یک هدر **Delta-Base**، که سند مورد استفاده به عنوان پایه برای محاسبه دلتا را مشخص می‌کند (در حالت ایده‌آل، همان **ETag** در درخواست **ETag** If-None-Match کلاینت!).

هدرهای استفاده شده در کدگذاری دلتا در جدول زیر خلاصه شده است:





Header	Description
ETag	Unique identifier for each instance of a document. Sent by the server in the response; used by clients in subsequent requests in If-Match and If-None-Match headers.
If-None-Match	Request header sent by the client, asking the server for a document if and only if the client's version of the document is different from the server's.
A-IM	Client request header indicating types of instance manipulations accepted.
IM	Server response header specifying the type of instance manipulation applied to the response. This header is sent when the response code is 226 IM Used.
Delta-Base	Server response header that specifies the ETag of the base document used for generating the delta (should be the same as the ETag in the client request's If-None-Match header).

Instance Manipulations, Delta Generators and Delta Appliers

کلاینت‌ها می‌توانند با استفاده از هدر A-IM انواع دستکاری نمونه‌هایی را که می‌پذیرند مشخص کنند. سرورها نوع دستکاری نمونه مورد استفاده در هدر IM را مشخص می‌کنند.

حالا انواع دستکاری‌های نمونه پذیرفته شده چیست و چه کاری انجام می‌دهند؟ جدول زیر برخی از انواع دستکاری‌های نمونه ثبت شده IANA را فهرست می‌کند.

Type	Description
vcdiff	Delta using the vcdiff algorithm ^a
diffe	Delta using the Unix <i>diff -e</i> command
gdiff	Delta using the gdiff algorithm ^b
gzip	Compression using the gzip algorithm
deflate	Compression using the deflate algorithm
range	Used in a server response to indicate that the response is partial content as the result of a range selection
identity	Used in a client request's A-IM header to indicate that the client is willing to accept an identity instance manipulation

a: Internet draft draft-korn-vcdiff-01 describes the vcdiff algorithm. This specification was approved by the IESG in early 2002 and should be released in RFC form shortly.

b: <http://www.w3.org/TR/NOTE-gdiff-19970901.html> describes the GDIFF algorithm.

یک "Delta Generator" در سرور، مانند شکل پیشین، سند پایه و آخرین نمونه سند را می‌گیرد و با استفاده از الگوریتم مشخص شده توسط کلایند در هدر A-IM، دلتا را بین این دو محاسبه می‌کند. در سمت کلاینت، یک "Delta Applier" دلتا را می‌گیرد و آن را به سند پایه اعمال می‌کند تا آخرین نمونه سند را ایجاد کند. به عنوان مثال، اگر الگوریتم مورد استفاده برای تولید دلتا، دستور یونیکس *diff -e*





باشد، کلاینت می‌تواند دلتا را با استفاده از عملکرد ویرایشگر متن یونیکسی `ed` اعمال کند، زیرا `diff -e <file1>` مجموعه‌ای از دستورات `ed` را ایجاد می‌کند که `<file1>` را به `<file2>` تبدیل می‌کند. `ed` یک ویرایشگر بسیار ساده با چند دستور پشتیبانی شده است. در مثال شکل قبلی، بخش ۵c می‌گوید که خط ۵ را در سند پایه حذف نموده و `chisels.<cr>` را اضافه نمایید. دستور العمل‌های پیچیده‌تر را می‌توان برای تغییرات بزرگ‌تر تولید کرد. الگوریتم یونیکس `diff -e` مقایسه خط به خط فایل‌ها را انجام می‌دهد. این بدیهی است برای فایل‌های متنی مشکلی ندارد اما برای فایل‌های باینری منجر به خرابی می‌شود. الگوریتم `vcdiff` قدرتمندتر است، حتی برای فایل‌های غیر متنی نیز کار می‌کند و به طور کلی دلتاهای کوچکتری نسبت به `diff -e` تولید می‌نماید.

مشخصات کدگذاری دلتا فرمت هدرهای `A-IM` و `IM` را با جزئیات مشخص می‌کند. کافی است بگوییم که چندین دستکاری نمونه را می‌توان در این هدرها (همراه با مقادیر کیفیت مربوطه) مشخص کرد. اسناد می‌توانند قبل از بازگرداندن به کلاینت‌ها، دستکاری‌های متعددی را انجام دهند تا فشرده‌سازی را به حداقل برسانند. به عنوان مثال، دلتاهای تولید شده توسط الگوریتم `vcdiff` ممکن است به نوبه خود با استفاده از الگوریتم `gzip` فشرده شوند. سپس پاسخ سرور حاوی هدر `IM` است: `.gzip .vcdiff`.

کلاینت ابتدا محتوا را `gunzip` می‌کند، سپس نتایج دلتا را در صفحه اصلی خود اعمال می‌کند تا سند نهایی تولید شود.

کدگذاری دلتا می‌تواند زمان انتقال را کاهش دهد، اما اجرای آن می‌تواند دشوار باشد. صفحه‌ای را تصور کنید که مرتبًاً تغییر می‌کند و افراد مختلف به آن دسترسی دارند. سروری که از کدگذاری دلتا پشتیبانی می‌کند، باید تمام نسخه‌های مختلف آن صفحه را با تغییر آن در طول زمان نگه دارد تا بفهمد چه چیزی بین هر کپی کلاینت درخواست کننده و آخرین نسخه تغییر کرده است. (اگر سند به طور مکرر تغییر کند، همانطور که کلاینت‌های مختلف سند را درخواست می‌کنند، نمونه‌های مختلفی از سند را دریافت می‌کنند. هنگامی که درخواست‌های بعدی را به سرور ارائه می‌دهند، بین نمونه سند خود و آخرین نمونه سند درخواست تغییر می‌کنند).

سرور برای اینکه بتواند فقط تغییرات را برای آن‌ها ارسال کند، باید کپی‌هایی از تمام نمونه‌های قبلی که کلاینت‌ها دارند را نگه دارد.)

در ازای کاهش تاخیر در ارائه اسناد، سرورها باید فضای دیسک را افزایش دهند تا نمونه‌های قدیمی اسناد را در اطراف نگه دارند. فضای دیسک اضافی لازم برای انجام این کار ممکن است به سرعت منافع حاصل از مقادیر انتقال کوچکتر را خنثی کند.





For More Information

<http://www.ietf.org/rfc/rfc2616.txt>

<http://www.ietf.org/rfc/rfc3229.txt>

<http://www.ietf.org/rfc/rfc1521.txt>

<http://www.ietf.org/rfc/rfc2045.txt>

<http://www.ietf.org/rfc/rfc1864.txt>

<http://www.ietf.org/rfc/rfc3230.txt>

Introduction to Data Compression - Khalid Sayood, Morgan Kaufmann





فصل شانزدهم – Internationalization

هر روز میلیاردها نفر به صدها زبان، استناد مختلف را تولید می‌کنند. HTTP برای برآورده کردن چشم انداز یک وب واقع‌جهانی، باید از انتقال و پردازش استناد بین المللی به زبان‌ها و الفبای بسیاری پشتیبانی کند.

این فصل دو موضوع اصلی بین المللی سازی وب را پوشش می‌دهد:

- Character Set Encodings
- Language Tags

برنامه‌های HTTP از Character Set Encodings برای درخواست و نمایش متن در الفبای مختلف استفاده نموده و از Language Tags برای توصیف و محدود کردن محتوا به زبان‌هایی استفاده می‌کنند که کاربر می‌فهمد.

این فصل:

- نحوه تعامل Scheme HTTP با استانداردهای الفبای چند زبانه را توضیح می‌دهد.
- مروری سریع از اصطلاحات، فناوری و استانداردها ارائه می‌دهد تا به برنامه نویسان HTTP کمک کند تا کارها را به درستی انجام دهند (خوانندگان آشنا با Character Encodings می‌توانند این بخش را نادیده بگیرند)
- سیستم نامگذاری استاندارد را توضیح می‌دهد. زبان‌ها و نحوه توصیف و انتخاب محتوا توسط Language Tags های استاندارد
- قوانین و احتیاط‌های URI بین المللی را بیان می‌کند.
- قوانین مربوط به تاریخ‌ها و سایر مسائل بین المللی سازی را به طور خلاصه مورد بحث قرار می‌دهد.

HTTP Support for International Content

پیام‌های HTTP می‌توانند محتوا را به هر زبانی منتقل کنند، همانطور که می‌توانند تصاویر، فیلم‌ها یا هر نوع رسانه دیگری را حمل کنند. برای HTTP، بدنه موجودیت فقط یک جعبه بیت^{۲۲} است.

برای پشتیبانی از محتوای بین المللی، سرورها باید الفبا و زبان هر سند را به کلاینت‌ها اطلاع دهند، بنابراین کلاینت می‌تواند به درستی بیت‌های سند را به کاراکتر تبدیل کند و محتوا را به درستی پردازش و به کاربر ارائه دهد.

سرورها با پارامتر Content-Type Charset و هدرهای HTTP Content-Language به کلاینت‌ها درباره الفبای و زبان سند، اطلاعات لازم را ارائه می‌دهند. این هدرها آنچه را در جعبه بیت‌های موجودیت وجود

²² box of bits





دارد، نحوه تبدیل محتویات به Character های مناسبی که می‌توانند روی صفحه نمایش داده شوند و زبان گفتاری که کلمات نشان می‌دهند را توصیف می‌کنند.

در عین حال، کلاینت باید به سرور بگوید که کاربر کدام زبان‌ها را می‌فهمد و کدام الگوریتم‌های کدگذاری الفبایی را مرورگر نصب کرده است. کلاینت هدرهای Accept-Language و Accept-Charset را ارسال می‌کند تا به سرور بگوید که کدام الگوریتم‌های Character Set Encoding و زبان‌هایی را که کلاینت می‌فهمد و کدام یک از آن‌ها ترجیح داده می‌شود.

هدرهای پذیرش HTTP زیر ممکن است توسط فرانسوی‌زبانی ارسال شود که زبان مادری خود را ترجیح می‌دهد (اما کمی انگلیسی صحبت می‌کند) و از مرورگری استفاده می‌کند که از iso-8859-1 West European UTF-8 Unicode Charset Encoding و Charset Encoding پشتیبانی می‌کند:

Accept-Language: fr, en;q=0.8

Accept-Charset: iso-8859-1, utf-8

پارامتر "q=0.8" یک عامل کیفیت است که به انگلیسی (۰.۸) اولویت کمتری نسبت به فرانسوی (۱.۰) به طور پیش فرض) می‌دهد.

Character Sets and HTTP

بنابراین، بایاید مستقیماً به مهم‌ترین (و گیج کننده) جنبه‌های بین‌المللی سازی وب بپردازیم – اسکریپت‌های الفبای بین‌المللی و Character Set Encoding آن‌ها.

استانداردهای مجموعه کاراکترهای وب می‌توانند بسیار گیج کننده باشد. بسیاری از مردم وقتی برای اولین بار سعی می‌کنند نرمافزار وب بین‌المللی بنویسند، به دلیل اصطلاحات پیچیده و متناقض، استناد استانداردی که باید برای خواندن آن‌ها هزینه کنید و ناآشنایی با زبان‌های خارجی، نامید می‌شوند. این بخش و بخش بعدی باید استفاده از Character Set Encoding را برای شما آسان‌تر کند.

Charset Is a Character-to-Bits Encoding

مقادیر HTTP Charset به شما می‌گوید که چگونه بیت‌های محتوای موجودیت، به کاراکترهایی در یک الفبای خاص تبدیل می‌شوند. هر تگ Charset، الگوریتمی را برای ترجمه بیت‌ها به کاراکترها فراهم می‌کند (و بالعکس). تگ‌های Charset در مجموعه کاراکترهای MIME، که توسط IANA نگهداری می‌شود، استاندارد شده‌اند (به شرحی <http://www.iana.org/assignments/character-sets> مراجعه کنید).

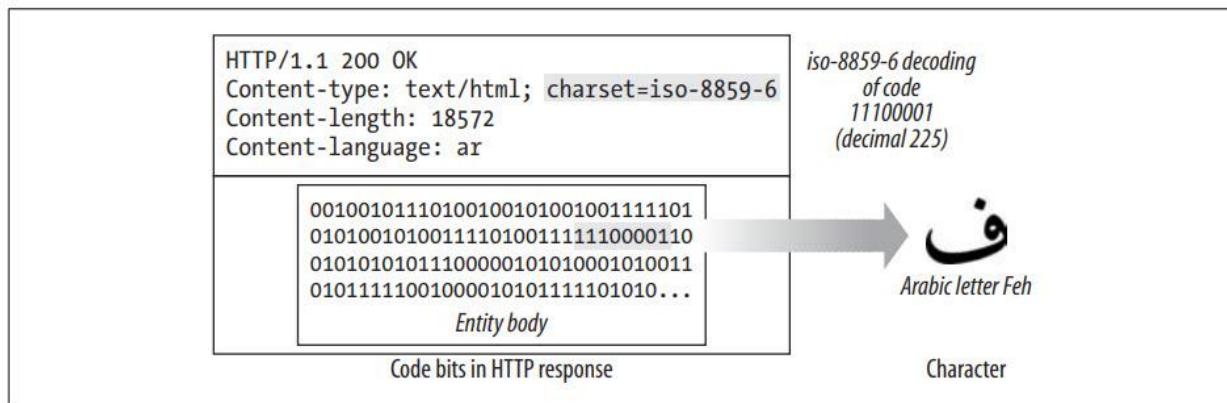




هدر Content-Type زیر به گیرنده می‌گوید که محتوا یک فایل HTML است و پارامتر charset به گیرنده می‌گوید که از طرح Decoding مجموعه حروف عربی iso-8859-6 برای بیت‌های محتوا به کاراکترها استفاده کند:

Content-Type: text/html; charset=iso-8859-6

طرح iso-8859-6 Decoding مقادیر ۸ بیتی را به هر دو الفبای لاتین و عربی، از جمله اعداد، علائم نقطه گذاری و سایر نمادها Map می‌کند. برای مثال، در شکل زیر، الگوی بیت هایلایت شده دارای مقدار کد ۲۲۵ است که (تحت ۶-iso-8859) به حرف عربی "FEH" (صدایی مانند حرف انگلیسی "F") اشاره می‌کند.



برخی از Character Encoding ها (مانند UTF-8 و ISO-2022-JP) کدهای پیچیده‌تر و با طول متغیر هستند که تعداد بیت‌ها در هر کاراکتر متفاوت است. این نوع Encoding به شما امکان می‌دهد از بیت‌های اضافی برای پشتیبانی از حروف با تعداد کاراکترهای زیاد (مانند چینی و ژاپنی) استفاده کنید و این در حالی است که شما از بیت‌هایی کمتری برای پشتیبانی از کاراکترهای استاندارد لاتین استفاده می‌کنید.

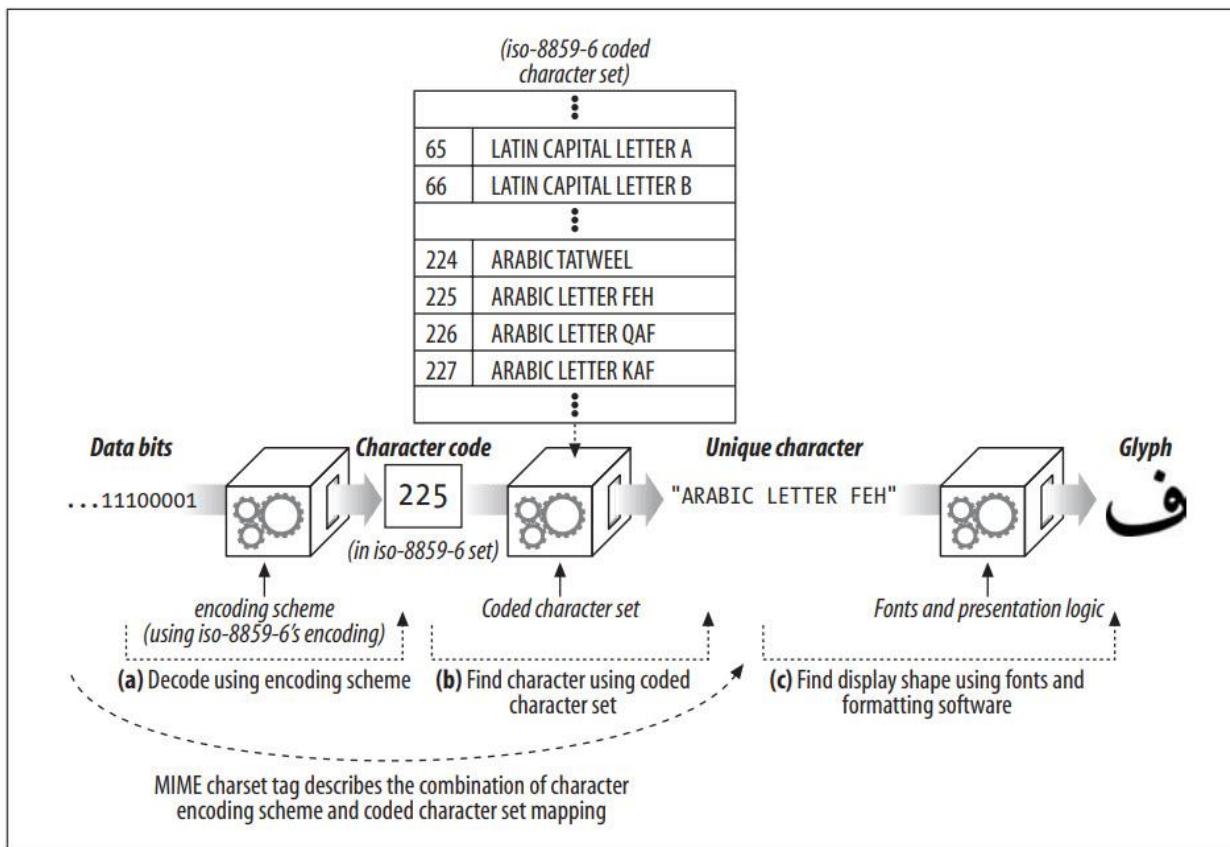
How Character Sets and Encodings Work

باید بینیم Character Set ها و Encoding ها واقعاً چه کار می‌کنند.

ما می‌خواهیم بیت‌های یک سند را به کاراکترهایی تبدیل کنیم که بتوانیم آن‌ها را روی صفحه نمایش دهیم. اما از آنجایی که الفبای متفاوتی وجود دارد و روش‌های مختلفی برای Encoding کاراکترها به بیت وجود دارد (هر کدام مزایا و معایب دارند)، ما به یک روش استاندارد برای توصیف و اعمال الگوریتم Decoding بیت به کاراکتر نیاز داریم.

همانطور که در شکل زیر نشان داده است، تبدیل بیت به کاراکتر در دو مرحله انجام می‌شود:





در بخش a شکل، بیت‌های یک سند به یک کد کاراکتری تبدیل می‌شوند که یک کاراکتر شماره‌دار خاص را در مجموعه کاراکترهای کدگذاری شده خاص مشخص می‌کند. در مثال، کد کاراکتر Decode شده با شماره ۲۲۵ است.

در بخش b شکل، کد کاراکتر برای انتخاب یک عنصر خاص از مجموعه کاراکترهای کدگذاری شده استفاده می‌کند. در iso-8859-6، مقدار ۲۲۵ با "حرف عربی FEH" مطابقت دارد. الگوریتم‌های مورد استفاده در مراحل a و b از تگ MIME Charset تعیین می‌شوند.

یکی از اهداف کلیدی بین المللی نمودن سیستم کاراکترها، جداسازی معناشناسی (حروف) از ارائه (فرم‌های ارائه گرافیکی) است. HTTP فقط به انتقال داده‌های کاراکتر و برچسب‌های Charset و Language مربوط می‌شود.

همانطور که در بخش c شکل نشان داده است، ارائه اشکال کاراکتر توسط نرم افزار نمایش گرافیکی کاربر (مرورگر، سیستم عامل، فونت‌ها) انجام می‌شود.





The Wrong Charset Gives the Wrong Characters

اگر از پارامتر Charset اشتباه استفاده شود، کلاینت کاراکترهای عجیب و غریب و جعلی را مشاهده می‌کند. فرض کنید یک مرورگر مقدار ۲۲۵ (باینری ۱۱۱۰۰۰۱) را از بدنه دریافت کرده است:

اگر مرورگر فکر کند که بدنه با کاراکترهای iso-8859-1 Western European کدگذاری شده است، یک "a" لاتین کوچک با لهجه تند نشان می‌دهد:

á

اگر مرورگر از کدهای عربی iso-8859-6 استفاده می‌کند، "FEH" را نشان می‌دهد:

ف

اگر مرورگر از iso-8859-7 یونانی استفاده می‌کند، یک "آلفا" کوچک نشان می‌دهد:

α

اگر مرورگر از کدهای عبری iso-8859-8 استفاده می‌کند، "BET" را نشان می‌دهد:

ב

Standardized MIME Charset Values

ترکیبی از یک رمزگذاری کاراکتر خاص و یک مجموعه کاراکتر کدگذاری شده خاص، MIME charset نامیده می‌شود. HTTP از تگ‌های MIME استاندارد شده در هدرهای Accept-Charset و Content-Type استفاده می‌کند. مقادیر مجموعه IANA MIME charset با ثبت می‌شوند.





MIME charset value	Description
us-ascii	The famous character encoding standardized in 1968 as ANSI_X3.4-1968. It is also named ASCII, but the "US" prefix is preferred because of several international variants in ISO 646 that modify selected characters. US-ASCII maps 7-bit values into 128 characters. The high bit is unused.
iso-8859-1	iso-8859-1 is an 8-bit extension to ASCII to support Western European languages. It uses the high bit to include many West European characters, while leaving the ASCII codes (0–127) intact. Also called iso-latin-1, or nicknamed "Latin1."
iso-8859-2	Extends ASCII to include characters for Central and Eastern European languages, including Czech, Polish, and Romanian. Also called iso-latin-2.
iso-8859-5	Extends ASCII to include Cyrillic characters, for languages including Russian, Serbian, and Bulgarian.
iso-8859-6	Extends ASCII to include Arabic characters. Because the shapes of Arabic characters change depending on their position in a word, Arabic requires a display engine that analyzes the context and generates the correct shape for each character.
iso-8859-7	Extends ASCII to include modern Greek characters. Formerly known as ELOT-928 or ECMA-118:1986.
iso-8859-8	Extends ASCII to include Hebrew and Yiddish characters.
iso-8859-15	Updates iso-8859-1, replacing some less-needed punctuation and fraction symbols with forgotten French and Finnish letters and replacing the international currency sign with the symbol for the new Euro currency. This character set is nicknamed "Latin0" and may one day replace iso-8859-1 as the preferred default character set in Europe.
iso-2022-jp	iso-2022-jp is a widely used encoding for Japanese email and web content. It is a variable-length encoding scheme that supports ASCII characters with single bytes but uses three-character modal escape sequences to shift into three different Japanese character sets.
euc-jp	euc-jp is an ISO 2022-compliant variable-length encoding that uses explicit bit patterns to identify each character, without requiring modes and escape sequences. It uses 1-byte, 2-byte, and 3-byte sequences of characters to identify characters in multiple Japanese character sets.
Shift_JIS	This encoding was originally developed by Microsoft and sometimes is called SJIS or MS Kanji. It is a bit complicated, for reasons of historic compatibility, and it cannot map all characters, but it still is common.
koi8-r	KOI8-R is a popular 8-bit Internet character set encoding for Russian, defined in IETF RFC 1489. The initials are transliterations of the acronym for "Code for Information Exchange, 8 bit, Russian."
utf-8	UTF-8 is a common variable-length character encoding scheme for representing UCS (Unicode), which is the Universal Character Set of the world's characters. UTF-8 uses a variable-length encoding for character code values, representing each character by from one to six bytes. One of the primary features of UTF-8 is backward compatibility with ordinary 7-bit ASCII text.
windows-1252	Microsoft calls its coded character sets "code pages." Windows code page 1252 (a.k.a. "CP1252" or "WinLatin1") is an extension of iso-8859-1.

Content-Type Charset Header and META Tags

سرورهای وب با استفاده از پارامتر Content-Type را در هدر charset می‌گذارند:

Content-Type: text/html; charset=iso-2022-jp





اگر هیچ Charset ای به صراحت فهرست نشده باشد، گیرنده ممکن است سعی کند مجموعه Charset را از محتوای سند استنتاج کند. برای محتوای HTML، مجموعه کاراکترها ممکن است در تگ‌های <META HTTP-> یافت شوند که مجموعه Charset ها را توصیف می‌کنند.

مثال زیر نشان می‌دهد که چگونه تگ‌های Charset HTML مجموعه HTML META ها را روی کدگذاری ژاپنی-ISO-2022-jp تنظیم می‌کنند. اگر سند HTML نباشد یا تگ META Content-Type وجود نداشته باشد، نرم‌افزار ممکن است سعی کند کدگذاری کاراکتر را با اسکن متن واقعی برای الگوهای رایج نشان دهنده زبان‌ها و کدگذاری‌ها استنتاج کند.

```
<HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-2022-jp">
  <META LANG="jp">
  <TITLE>A Japanese Document</TITLE>
</HEAD>
<BODY>
  ...

```

اگر کلاینت نتواند کدگذاری کاراکتر را استنباط کند، ISO-8859-1 را فرض می‌کند.

The Accept-Charset Header

هزاران روش Encoding و Decoding کاراکتر تعریف شده وجود دارد که در چند دهه گذشته توسعه یافته است. اکثر کلاینت‌ها از همه سیستم‌های کدنویسی و نقشه برداری کاراکترهای مختلف پشتیبانی نمی‌کنند.

کلاینت‌های HTTP می‌توانند با استفاده از هدر درخواست Accept-Charset به سرورها بگویند که دقیقاً کدام سیستم کاراکتری را پشتیبانی می‌کنند. مقدار هدر Accept-Charset فهرستی از طرح‌های کدگذاری کاراکتر را ارائه می‌کند که کلاینت پشتیبانی می‌کند.

برای مثال، هدر درخواست HTTP زیر نشان می‌دهد که کلاینت سیستم کاراکتر ISO-8859-1 اروپای غربی و همچنین سیستم سازگاری Unicode با طول متغیر UTF-8 را می‌پذیرد. سرور برای بازگرداندن محتوا در هر یک از این طرح‌های کدگذاری کاراکتر آزاد است.

Accept-Charset: iso-8859-1, utf-8

توجه داشته باشید که هیچ عنوان پاسخ Content-Charset برای مطابقت با هدر درخواست Accept-Charset وجود ندارد. مجموعه کاراکتر پاسخ توسط پارامتر Charset هدر پاسخ





از سرور بازگردانده می‌شود تا با MIME سازگار باشد. خیلی بد است که این متقاض نیست، اما همه اطلاعات هنوز وجود دارد.

Multilingual Character Encoding Primer

بخش قبل توضیح داد که چگونه هدر Content-Type charset HTTP و پارامتر Accept-Charset اطلاعات کدگذاری کاراکتر را از کلاینت و سرور منتقل می‌کنند. برنامه نویسان HTTP که کارهای زیادی با برنامه‌ها و محتوای بین‌المللی انجام می‌دهند، برای درک مشخصات فنی و پیاده سازی صحیح نرم افزار، باید درک عمیق‌تری از سیستم‌های کاراکتر چند زبانه داشته باشند.

یادگیری سیستم‌های کاراکتر چند زبانه آسان نیست - اصطلاحات پیچیده و ناسازگار است، اغلب برای خواندن اسناد استاندارد باید هزینه پرداخت کنید و ممکن است با زبان‌های دیگری که با آن‌ها کار می‌کنید ناآشنا باشید. این بخش مروری بر سیستم‌ها و استانداردهای کاراکتر است. اگر از قبل با کدگذاری کاراکترها راحت هستید یا به این جزئیات علاقه ندارید، به راحتی به بخش «Language Tags and HTTP» بروید.

Character Set Terminology

در اینجا هشت اصطلاح در مورد سیستم‌های کاراکتر الکترونیکی وجود دارد که باید بدانید:

Character

یک حرف الفبایی، عدد، علامت نقطه گذاری، ایدئوگرام (مانند چینی)، نماد، یا دیگر «اتم» نوشتاری. ابتکار مجموعه کاراکتر جهانی (UCS) که به طور غیررسمی با نام یونیکد شناخته می‌شود، مجموعه‌ای استاندارد از نام‌های متنی برای بسیاری از کاراکترها در بسیاری از زبان‌ها ایجاد کرده است که اغلب برای نام‌گذاری راحت و منحصر به فرد کاراکترها استفاده می‌شود.

Glyph

یک الگوی stroke یا شکل گرافیکی منحصر به فرد که یک کاراکتر را توصیف می‌کند. اگر بتوان یک کاراکتر را به روش‌های مختلف نوشت، ممکن است چندین علامت داشته باشد.

Coded character

یک عدد منحصر به فرد که به یک کاراکتر اختصاص داده شده تا بتوانیم با آن کار کنیم.

Coding space

طیفی از اعداد صحیح که ما قصد داریم از آن‌ها به عنوان مقادیر کد کاراکتر استفاده کنیم.





Code width

تعداد بیت‌ها در هر کد کاراکتری (با اندازه ثابت).

Character repertoire

یک مجموعه کاری خاص از کاراکترها (زیر مجموعه‌ای از همه کاراکترهای جهانی).

Coded character set

مجموعه‌ای از کاراکترهای کدگذاری شده که مجموعه‌ای از کاراکترها (گزیده‌ای از کاراکترهای سراسر جهان) را می‌گیرد و به هر کاراکتر یک کد از یک فضای کدگذاری اختصاص می‌دهد. به عبارت دیگر، کدهای کاراکتر عددی را به کاراکترهای واقعی Map می‌کند.

Character encoding scheme

الگوریتمی برای کدگذاری کدهای کاراکترهای عددی به دنباله‌ای از بیت‌های محتوا (و رمزگشایی آن‌ها). طرح‌های کدگذاری کاراکتر را می‌توان برای کاهش مقدار داده‌های مورد نیاز برای شناسایی کاراکترها (فسرده‌سازی)، کار بر روی محدودیت‌های انتقال و یکپارچه‌سازی مجموعه‌های کاراکتر کدگذاری شده همپوشانی استفاده کرد.

Charset Is Poorly Named

از نظر فنی، تگ Accept-Charset و Content-Type charset (که در پارامتر MIME Charset استفاده می‌شود) به هیچ وجه مجموعه کاراکتری را مشخص نمی‌کند. مقدار MIME Charset یک الگوریتم کل را برای نگاشت بیت‌های داده به کدها به کاراکترهای منحصر به فرد نام می‌برد. این دو مفهوم مجزا از طرح کدگذاری کاراکتر و مجموعه کاراکترهای کدگذاری شده را ترکیب می‌کند.

این اصطلاح نامرتب و گیج کننده است، زیرا قبلاً استانداردهای منتشر شده برای طرح‌های کدگذاری کاراکتر و مجموعه کاراکترهای کدگذاری شده وجود دارد. در اینجا آنچه نویسنده‌گان HTTP/1.1 در مورد استفاده از اصطلاحات خود می‌گویند (در RFC 2616) عبارتست از:

اصطلاح "Charset" در این سند برای اشاره به روشی برای تبدیل دنباله‌ای از Octet‌ها به دنباله‌ای از کاراکترها استفاده می‌شود. توجه: این استفاده از عبارت "Charset" بیشتر به عنوان Character Encoding نامیده می‌شود. با این حال، از آنجایی که HTTP و MIME رجیستری یکسانی دارند، مهم است که اصطلاحات نیز به اشتراک گذاشته شوند.





IETF همچنین از اصطلاحات غیر استاندارد در RFC 2277 استفاده می‌کند:

این سند از اصطلاح "charset" به معنای مجموعه‌ای از قوانین برای نگاشت از یک دنباله هشت‌تایی (Octet) به دنباله‌ای از کاراکترها استفاده می‌کند، مانند ترکیبی از مجموعه کاراکترهای کدگذاری شده و یک طرح رمزگذاری کاراکتر. این همان چیزی است که به عنوان یک شناسه در پارامترهای MIME charset استفاده می‌شود و در رجیستری IANA ثبت می‌شود. (توجه داشته باشید که این اصطلاحی نیست که توسط سایر نهادهای استاندارد مانند ISO استفاده می‌شود).

بنابراین، هنگام خواندن اسناد استاندارد مراقب باشید تا دقیقاً بدانید که چه چیزی تعریف می‌شود. اکنون که اصطلاحات را مرتب کردہایم، بیایید کمی دقیق‌تر به کاراکترها، Glyph‌ها، مجموعه کاراکترها و کدگذاری کاراکترها نگاه کنیم.

Characters

کاراکترها اساسی‌ترین اجزای سازنده نوشتمند هستند. یک کاراکتر نشان دهنده یک حرف الفبایی، عدد، علامت نقطه گذاری، ایدئوگرام (مانند زبان چینی)، نماد ریاضی یا سایر واحدهای اصلی نوشتمند است.

کاراکترها مستقل از فونت و سبک هستند. شکل زی چندین گونه از یک کاراکتر را نشان می‌دهد که "حرف کوچک A" نامیده می‌شود. یک خواننده بومی زبان‌های اروپایی غربی بلافصله هر پنج شکل را به عنوان یک کاراکتر تشخیص می‌دهد، حتی اگر الگوها و سبک‌های Stroke کاملاً متفاوت باشند.

بسیاری از سیستم‌های نوشتاری نیز بسته به موقعیت کاراکتر در کلمه، شکل‌های خطی متفاوتی برای یک کاراکتر دارند. به عنوان مثال، چهار خط در شکل زیر همگی نشان دهنده کاراکتر "عربی AIN" هستند. بخش a شکل زیر نشان می‌دهد که چگونه "AIN" به عنوان یک کاراکتر مستقل نوشته می‌شود. بخش d شکل "AIN" را در ابتدای کلمه نشان می‌دهد، بخش c شکل نیز، "AIN" را در وسط یک کلمه نشان می‌دهد، و بخش b شکل "AIN" را در انتهای یک کلمه نشان می‌دهد.





(a) Standalone (b) Final position (c) Medial position (d) Initial position

ع ع ع ع

(These different glyphs represent the same character, "ARABIC LETTER AIN")

Glyphs, Ligatures, and Presentation Forms

کاراکترها را با **Glyph** اشتباه نگیرید. کاراکترها «اطم‌های» منحصر به فرد و انتزاعی زبان هستند. **Glyph**‌ها روش‌های خاصی هستند که هر کاراکتر را ترسیم می‌کند. هر کاراکتر بسته به سبک هنری، حروف بسیار متفاوتی دارد.

همچنین، کاراکترها را با فرم‌های ارائه^{۲۳} اشتباه نگیرید. برای زیباتر جلوه دادن نوشتار، بسیاری از اسکریپت‌ها و حروف‌های دست‌نویس به شما امکان می‌دهند کاراکترهای مجاور را در لیگاتورهای زیبا بپیوندید، که در آن دو کاراکتر به راحتی به هم متصل می‌شوند.

در زبان انگلیسی حروف‌نویس‌ها اغلب «F» و «I» را به یک «بسته FI» می‌پیوندند (بخش a-b شکل زیر)، و نویسنده‌گان عرب اغلب کاراکترهای «ALIF» و «LAM» را به یک لیگاتور جذاب می‌پیوندند (بخش c-d شکل زیر).

(a) Without FI ligature

(b) With FI ligature

(c) Without LA ligature

(d) With LA ligature

file file

ALIF LAM لام

در اینجا قانون کلی وجود دارد: اگر معنی متن تغییر کند وقتی که یک **Glyph** را با علامت دیگری جایگزین می‌کنید، علامت‌ها کاراکترهای متفاوتی هستند. در غیر این صورت، آن‌ها همان کاراکترها هستند، با ظاهری متفاوت.

²³ presentation forms





Coded Character Sets

مجموعه کاراکترهای کدگذاری شده، تعریف شده در RFC های ۲۲۷۷ و ۲۱۳۰، اعداد صحیح را به کاراکترها می‌کنند. مجموعه کاراکترهای کدگذاری شده اغلب به صورت آرایه پیاده‌سازی می‌شوند که با شماره کد ایندکس می‌شوند (شکل زیر را ببینید). عناصر آرایه کاراکتر هستند.

		US-ASCII coded character set															
		!	"	#	\$	%	&	'	()	*	+	-	/			
		0	1	2	3	4	5	6	7	8	9	:	:	<	=	>	?
		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
		P	Q	R	S	T	U	V	W	X	Y	Z	ı	ı	ı	ı	ı
		ı	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
		ı	p	q	r	s	t	u	v	w	x	y	ı	ı	ı	ı	ı

Code 68 (0x44) "LATIN CAPITAL LETTER D"

بیایید به چند استاندارد مهم مجموعه کاراکترهای کدگذاری شده نگاه کنیم، از جمله مجموعه کاراکترهای تاریخی US-ASCII، پسوندهای iso-8859 به ASCII، مجموعه کاراکترهای JIS X 0201 ژاپنی و مجموعه کاراکترهای جهانی (یونیکد).

US-ASCII: The mother of all character sets

ANSI X3.4 ASCII معروفترین مجموعه کاراکترهای کدگذاری شده است که در سال ۱۹۶۸ به عنوان استاندارد «کد استاندارد آمریکایی برای تبادل اطلاعات» استاندارد شده است. ASCII فقط از مقادیر ۰-۱۲۷ استفاده می‌کند. بنابراین در این بخش تنها ۷ بیت برای پوشش فضای کد مورد نیاز است.

نام ترجیحی "ASCII" است تا آن را از انواع بین‌المللی مجموعه کاراکترهای ۷ بیتی تمایز کند.

پیام‌های HTTP (هدر، URI و غیره) از US-ASCII استفاده می‌کنند.

iso-8859

استانداردهای مجموعه کاراکتر ISO-8859 ابر مجموعه‌های^{۲۴} ۸ بیتی US-ASCII هستند که از بیت بالا برای اضافه کردن کاراکتر برای نوشتن بین‌المللی استفاده می‌کنند. فضای اضافی ارائه شده توسط بیت اضافی (۱۲۸ کد اضافی) به اندازه کافی بزرگ نیست که حتی همه کاراکترهای اروپایی را در خود جای دهد (بدون ذکر کاراکترهای آسیایی)، بنابراین ISO-8859 مجموعه کاراکترهای سفارشی شده را برای مناطق مختلف ارائه می‌دهد:

²⁴ supersets





iso-8859-1	Western European languages (e.g., English, French)
iso-8859-2	Central and Eastern European languages (e.g., Czech, Polish)
iso-8859-3	Southern European languages
iso-8859-4	Northern European languages (e.g., Latvian, Lithuanian, Greenlandic)
iso-8859-5	Cyrillic (e.g., Bulgarian, Russian, Serbian)
iso-8859-6	Arabic
iso-8859-7	Greek
iso-8859-8	Hebrew
iso-8859-9	Turkish
iso-8859-10	Nordic languages (e.g., Icelandic, Inuit)
iso-8859-15	Modification to iso-8859-1 that includes the new Euro currency character

iso-8859-1، همچنین به عنوان Latin1 شناخته می‌شود که مجموعه کاراکترهای پیش فرض برای HTML است. می‌توان از این مجموعه برای نمایش متن در اکثر زبان‌های اروپای غربی استفاده کرد. بحث‌هایی در مورد جایگزینی iso-8859-1 با iso-8859-15 به عنوان مجموعه کاراکترهای کد شده پیش فرض HTTP وجود دارد، زیرا شامل نماد جدید ارز یورو است. با این حال، به دلیل پذیرش گستره 1-iso-8859-15، بعید است که تغییر گستره‌های به iso-8859-15 برای مدتی طولانی به تصویب برسد.

JIS X 0201

JIS X 0201 یک مجموعه کاراکتر بسیار مینیمال است که کاراکترهای ASCII را با کاراکترهای katakana نیم عرض ژاپنی^{۲۵} گسترش می‌دهد. کاراکترهای katakana نیمه عرض در اصل در سیستم تلگراف ژاپنی استفاده می‌شد. JIS X 0201 اغلب "JIS Roman" نامیده می‌شود. JIS مخفف Japanese Industrial Standard است.

JIS X 0208 and JIS X 0212

زبان ژاپنی شامل هزاران کاراکتر از چندین سیستم نوشتاری است. در حالی که می‌توان با استفاده از ۶۳ کاراکتر آواتی پایه در JIS X 0201 (به طرز دردنگی) را فراهم نمود، ولی برای استفاده عملی به مجموعه کاراکترهای بسیار کامل‌تری نیاز است.

مجموعه کاراکترهای JIS X 0208 اولین مجموعه کاراکتر ژاپنی چند بایتی بود. ۶۸۷۹ کاراکتر کدگذاری شده را تعریف کرد که اکثر آن‌ها kanji-based Chinese می‌باشند. مجموعه کاراکترهای 6067 JIS X 0212 کاراکتر اضافی را فراهم می‌کند.

²⁵ half width katakana characters

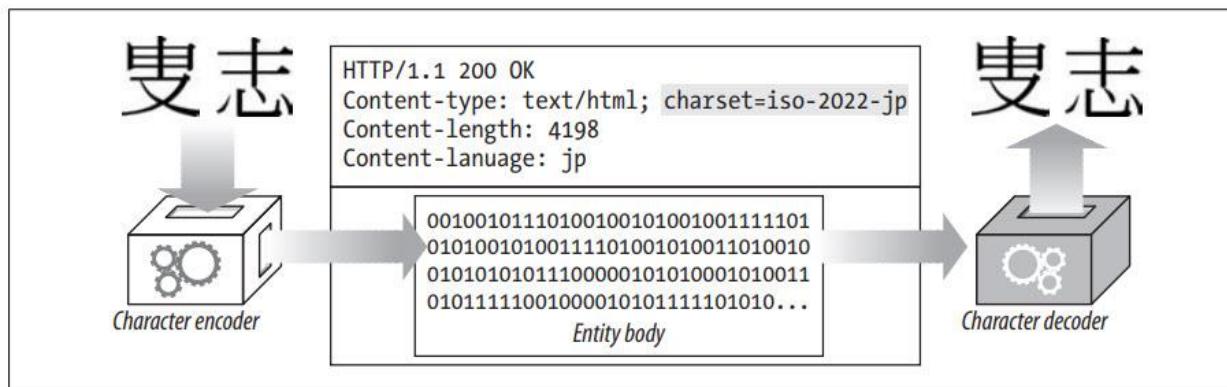


UCS

یک تلاش استاندارد جهانی برای ترکیب همه کاراکترهای جهان در یک مجموعه کاراکتر کدگذاری شده است. ISO 10646 توسط UCS تعریف شده است. یونیکد یک کنسرسیو میکنند. UCS دارای فضای کدگذاری برای میلیون‌ها کاراکتر است، اگرچه مجموعه اصلی فقط از حدود ۵۰۰۰۰ کاراکتر تشکیل شده است.

Character Encoding Schemes

طرح‌های کدگذاری کاراکتر، اعداد کد کاراکتر را در بیت‌های محتوا بسته‌بندی می‌کنند و آن‌ها را دوباره در کدهای کاراکتری در انتهای دیگر باز می‌کنند (شکل زیر).



سه دسته کلی از طرح‌های کدگذاری کاراکتر وجود دارد:

Fixed width

کدگذاری با عرض ثابت هر کاراکتر کدگذاری شده را با تعداد ثابتی از بیت‌ها نشان می‌دهد. آن‌ها سریع پردازش می‌شوند اما می‌توانند فضا را هدر دهند.

Variable width (nonmodal)

کدگذاری‌های با عرض متغیر از تعداد بیت‌های متفاوتی برای شماره‌های کد کاراکتری مختلف استفاده می‌کنند. آن‌ها می‌توانند تعداد بیت‌های مورد نیاز برای کاراکترهای رایج را کاهش دهند و سازگاری با مجموعه کاراکترهای ۸ بیتی قدیمی را حفظ کنند و در عین حال امکان استفاده از چندین بایت را برای کاراکترهای بین‌المللی فراهم نمایند.

Variable width (modal)



کدگذاری‌های Modal از الگوهای «escape» ویژه برای جابجایی بین حالت‌های مختلف استفاده می‌کنند. به عنوان مثال، یک کدگذاری Modal می‌تواند برای جابجایی بین مجموعه کاراکترهای متعدد و همپوشانی در وسط متن استفاده شود. پردازش کدگذاری‌های Modal پیچیده است، اما می‌توانند به طور موثر از سیستم‌های نوشتاری پیچیده پشتیبانی کنند.

بیایید به چند طرح رمزگذاری رایج نگاه کنیم.

8-bit

Identity Encoding ۸ بیتی با عرض ثابت به سادگی هر کد کاراکتری را با مقدار ۸ بیتی مربوطه کدگذاری می‌کند. این مدل فقط از مجموعه کاراکترهایی با محدوده کد ۲۵۶ کاراکتر پشتیبانی می‌کند. خانواده مجموعه کاراکترهای iso-8859 از Identity Encoding ۸ بیتی استفاده می‌کند.

UTF-8

UTF-8 یک طرح کدگذاری کاراکتر محبوب است که برای UCS طراحی شده است (ucs "Transformation Format" است). UTF-8 برای مقادیر کد کاراکتر از یک کدگذاری غیر معین با طول متغیر استفاده می‌کند، جایی که بیت‌های اصلی اولین بایت طول کاراکتر کدگذاری شده را بر حسب بایت نشان می‌دهد و هر بایت بعدی حاوی شش بیت مقدار کد است (جدول زیر).

Character code bits	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
0–7	0ccccccc	-	-	-	-	-
8–11	110cccccc	10cccccc	-	-	-	-
12–16	1110cccccc	10cccccc	10cccccc	-	-	-
17–21	11110ccc	10cccccc	10cccccc	10cccccc	-	-
22–26	111110cc	10cccccc	10cccccc	10cccccc	10cccccc	-
27–31	1111110c	10cccccc	10cccccc	10cccccc	10cccccc	10cccccc

اگر اولین بایت کدگذاری شده دارای بیت بالای صفر باشد، طول آن فقط ۱ بایت است و ۷ بیت باقیمانده حاوی کد کاراکتر است. این نتیجه خوبی از سازگاری ASCII دارد (اما سازگاری با iso-8859 نیست، زیرا از بیت بالا استفاده می‌کند).

به عنوان مثال، کد کاراکتر ASCII "Z" (01011010) به صورت ۱ بایت ۰۱۰۱۱۰۱۰ می‌شود، در حالی که کد ۵۰۷۳ (مقدار باینری ۱۰۰۱۱۱۰۱۰۰۰۱) در ۳ بایت کدگذاری می‌شود:

11100001 10001111 10010001





iso-2022-jp

iso-2022-jp یک کدگذاری پرکاربرد برای اسناد اینترنتی ژاپنی است. Modal iso-2022-jp یک کدگذاری با طول متغیر و با تمام مقادیر کمتر از ۱۲۸ برای جلوگیری از مشکلات نرم افزار غیر ۸ بیتی است.

زمینه رمزگذاری همیشه روی یکی از چهار مجموعه کاراکتر از پیش تعریف شده تنظیم شده است. iso-2022-jp در ابتدا از مجموعه کاراکترهای USASCII استفاده می‌کند، اما می‌تواند به مجموعه کاراکترهای JIS X 0201 (JIS-Roman) یا مجموعه کاراکترهای بسیار بزرگتر JIS X 0208-1978 و JIS X 0208-1983 با استفاده از فرار ۳ بایتی Escape Sequences سوئیچ کند.

Escape Sequences ها در جدول زیر قابل مشاهده هستند. در عمل، متن ژاپنی با "ESC \\$@" یا "ESC (B" or "ESC (J" شروع می‌شود و با "ESC (B" or "ESC (J" پایان می‌یابد.

Escape sequence	Resulting coded character set	Bytes per code
ESC (B	US-ASCII	1
ESC (J	JIS X 0201-1976 (JIS Roman)	1
ESC \\$ @	JIS X 0208-1978	2
ESC \\$ B	JIS X 0208-1983	2

هنگامی که در حالت‌های US-ASCII یا JIS-Roman استفاده می‌شود. هنگام استفاده از مجموعه کاراکترهای بزرگتر JIS X 0208 دو بایت برای هر کد کاراکتر استفاده خواهد شد. کدگذاری بایت‌های ارسالی را بین ۳۳ تا ۱۲۶ محدود می‌کند.

euc-jp

euc-jp یکی دیگر از کدهای محبوب ژاپنی است. EUC مخفف "Extended UNIX Code" است که برای اولین بار برای پشتیبانی از کاراکترهای آسیایی در سیستم عامل‌های یونیکس ایجاد شد.

مانند iso-2022-jp، کدگذاری euc-jp یک کدگذاری با طول متغیر است که امکان استفاده از چندین مجموعه کاراکتر استاندارد ژاپنی را فراهم می‌کند. اما برخلاف iso-2022-jp، کدگذاری euc-jp مودال نبوده و هیچ Escape Sequences ای برای جابجایی بین حالت‌ها وجود ندارد.

euc-jp از چهار مجموعه کاراکتر کدگذاری شده پشتیبانی می‌کند: JIS X 0201 (JIS-Roman) با چند جایگزین ژاپنی)، half-width katakana JIS X 0208 (کاراکتر در سیستم ASCII تلگراف اصلی ژاپنی استفاده می‌شود) و JIS X 0212





یک بایت برای کدگذاری JIS Roman (سازگار با ASCII)، دو بایت برای JIS X 0208 و JIS X 0212 استفاده می‌شود. کدگذاری کمی بیهوده است اما پردازش آن ساده است.

الگوهای کدگذاری در جدول زیر نشان داده شده است.

Which byte	Encoding values
JIS X 0201 (94 coded characters)	
1st byte	33–126
JIS X 0208 (6879 coded characters)	
1st byte	161–254
2nd byte	161–254
Half-width katakana (63 coded characters)	
1st byte	142
2nd byte	161–223
JIS X 0212 (6067 coded characters)	
1st byte	143
2nd byte	161–254
3rd byte	161–254

این بررسی ما از مجموعه کاراکترها و کدگذاری‌ها را به پایان می‌رساند. بخش بعدی تگ‌های Language و نحوه استفاده HTTP از آن را برای هدف قرار دادن محتوا برای مخاطبان توضیح می‌دهد.

Language Tags and HTTP

تگ‌های Language رشته‌های کوتاه و استاندارد شده‌ای هستند که زبان‌های گفتاری را نام گذاری می‌کنند.

ما به نام‌های استاندارد نیاز داریم، یا برخی از افراد اسناد فرانسوی را به عنوان French برچسب‌گذاری می‌کنند، دیگران از «Français» استفاده می‌کنند، دیگران ممکن است همچنان از French استفاده کنند، و افراد تنبل ممکن است فقط از «Fra» یا «F» استفاده کنند. تگ‌های Language استاندارد از این سردرگمی جلوگیری می‌کنند.

تگ‌های Language برای انگلیسی (en)، آلمانی (de)، کره‌ای (ko) و برای بسیاری از زبان‌های دیگر نیز بدین شکل وجود دارد. تگ‌های Language می‌توانند انواع و گویش‌های منطقه‌ای زبان‌ها را توصیف





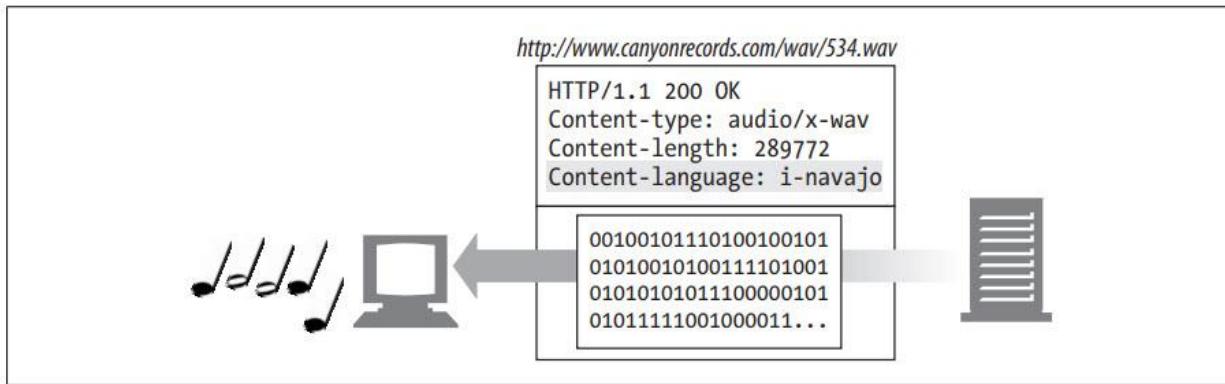
کنند، مانند پرتغالی برزیلی (pt-BR)، انگلیسی آمریکایی (en-US)، و چینی هونان (zhxiang). حتی یک تگ استاندارد برای کلینگون (i-klingon) وجود دارد!

The Content-Language Header

فیلد هدر موجودیت Content-Language زبان‌های مخاطبان مورد نظر را برای موجودیت توصیف می‌کند. اگر محتوا عمده‌اً برای مخاطب فرانسوی در نظر گرفته شده باشد، قسمت هدر ContentLanguage شامل موارد زیر است:

Content-Language: fr

هدر Content-Language به اسناد متنی محدود نمی‌شود. کلیپ‌های صوتی، فیلم‌ها و برنامه‌ها همگی ممکن است برای مخاطبان زبان خاصی در نظر گرفته شوند. هر نوع رسانه‌ای که مخاطبان زبان خاصی را هدف قرار می‌دهد، می‌تواند هدر Content-Language داشته باشد. در شکل زیر، فایل صوتی برای مخاطبان ناواهو برچسب گذاری شده است.



اگر محتوا برای چندین مخاطب در نظر گرفته شده است، می‌توانید چندین زبان را فهرست کنید. همانطور که در مشخصات HTTP پیشنهاد شده است، بازخوانی "Treaty of Waitangi" که به طور همزمان در نسخه اصلی مأموری و انگلیسی ارائه شده است، مستلزم موارد زیر است:

Content-Language: mi, en

با این حال، فقط به این دلیل که چندین زبان در یک موجودیت وجود دارد به این معنی نیست که برای مخاطبان زبانی متعدد در نظر گرفته شده است. آغازگر زبان مبتدی، مانند «درس اول به زبان لاتین»، که به وضوح برای مخاطبان انگلیسی سواد استفاده می‌شود، به درستی فقط «en» را شامل می‌شود.





The Accept-Language Header

اگر ما حداقل یک زبان بلدیم، HTTP به ما این امکان را می‌دهد تا محدودیت‌ها و ترجیحات زبانی خود را به سرورهای وب منتقل کنیم. اگر وب سرور دارای چندین نسخه از یک منبع به زبان‌های مختلف باشد، می‌تواند محتوا را به زبان دلخواه، در اختیار ما قرار دهد.

در اینجا، کلاینت محتوای اسپانیایی را درخواست می‌کند:

Accept-Language: es

می‌توانید چندین تگ Language را در هدر Accept-Language قرار دهید تا همه زبان‌های پشتیبانی شده و ترتیب اولویت (از چپ به راست) را برشمایرید. در اینجا، کلاینت انگلیسی را ترجیح می‌دهد اما آلمانی سوئیسی (de-CH) یا سایر انواع آلمانی (de) را می‌پذیرد:

Accept-Language: en, de-CH, de

کلاینت‌ها از Accept-Charset و Accept-Language برای درخواست محتوایی که می‌توانند بفهمند استفاده می‌کنند. در فصل ۱۷ خواهیم دید که چگونه این کار با جزئیات بیشتری کار می‌کند.

Types of Language Tags

تگ‌های Language دارای یک Syntax استاندارد هستند که در RFC 3066، با نام "Tags for the "Identification of Languages" مستند شده است. از تگ‌های Language می‌توان برای نشان دادن موارد زیر استفاده کرد:

- General language classes (as in "es" for Spanish)
- Country-specific languages (as in "en-GB" for English in Great Britain)
- Dialects of languages (as in "no-bok" for Norwegian "Book Language")
- Regional languages (as in "sgn-US-MA" for Martha's Vineyard sign language)
- Standardized nonvariant languages (e.g., "i-navajo")
- Nonstandard languages (e.g., "x-snowboarder-slang")



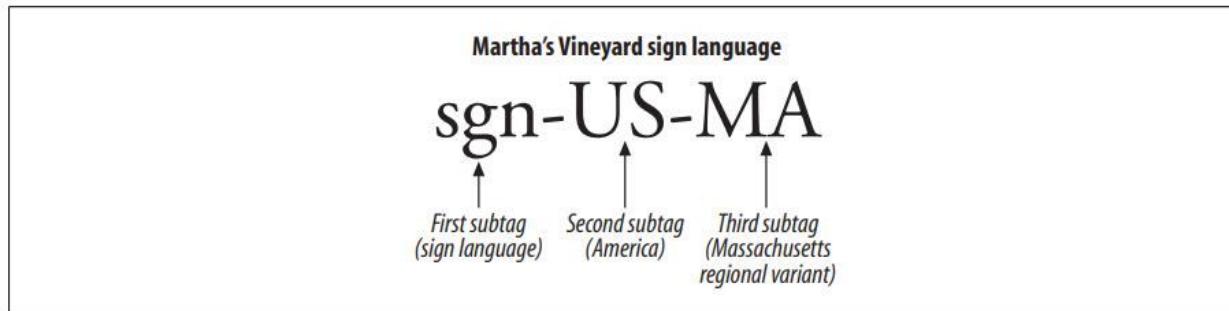


Subtags

تگ‌های Language دارای یک یا چند قسمت هستند که با خط تیره از هم جدا شده‌اند که به آن‌ها تگ فرعی^{۲۶} می‌گویند:

- اولین تگ فرعی به نام primary subtag. مقادیر استاندارد شده است.
- تگ فرعی دوم اختیاری است و از استاندارد نامگذاری خود پیروی می‌کند.
- هر گونه تگ فرعی بعدی ثبت نشده است.

فقط شامل حروف (A-Z) است. تگ‌های فرعی بعدی می‌توانند شامل حروف یا اعداد با طول حداقل هشت کاراکتر باشند. یک مثال در شکل زیر نشان داده شده است.



Capitalization

همه تگ‌ها به حروف بزرگ و کوچک حساس نیستند - تگ‌های "en" و "eN" معادل هستند. با این حال، حروف کوچک به طور معمول برای نشان دادن زبان‌های عمومی استفاده می‌شود، در حالی که حروف بزرگ برای نشان دادن کشورهای خاص استفاده می‌شود. به عنوان مثال، "fr" به معنای تمام زبان‌های طبقه بندی شده به عنوان فرانسوی است، در حالی که "FR" به معنای کشور فرانسه است.

IANA Language Tag Registrations

مقادیر تگ‌های فرعی زبان اول و دوم توسط استانداردهای مختلف و سازمان‌های نگهدارنده آن‌ها تعریف می‌شوند. IANA فهرست تگ‌های زبان استاندارد را با استفاده از قوانین مشخص شده در RFC 3066 مدیریت می‌کند.

اگر یک تگ Language از مقادیر استاندارد کشور و زبان تشکیل شده باشد، این تگ لازم نیست به طور خاص ثبت شود.

²⁶ subtags





فقط آن دسته از تگ‌های زبانی که نمی‌توانند خارج از کشور استاندارد و ارزش‌های زبانی تشکیل شوند، باید به‌ویژه در IANA ثبت شوند. بخش‌های زیر استانداردهای RFC 3066 را برای تگ‌های فرعی اول و دوم نشان می‌دهد.

First Subtag: Namespace

اولین تگ فرعی معمولاً یک توکن Language استاندارد است که از مجموعه استانداردهای زبان ISO 639 انتخاب می‌شود. اما همچنان می‌تواند حرف "i" برای شناسایی نامهای ثبت شده در IANA یا "x" برای نامهای داخلی خصوصی باشد. در اینجا قوانین وجود دارد:

اگر اول دارای subtag:

- دو کاراکتر، یک کد Language از استانداردهای ISO 639 و 639-1 است.
 - سه کاراکتر، یک کد Language است که در استاندارد ISO 639-2 و پسوندها^{۲۷} فهرست شده است.
 - حرف "i"، تگ Language به صراحت توسط IANA ثبت شده است
 - حرف "x"، تگ subtag یک Language خصوصی، غیر استاندارد و پسوندی است.
- چند نمونه در اینجا در جدول زیر نشان داده شده است.

²⁷ extensions





Language	ISO 639	ISO 639-2
Arabic	ar	ara
Chinese	zh	chi/zho
Dutch	nl	dut/nla
English	en	eng
French	fr	fra/fre
German	de	deu/ger
Greek (Modern)	el	ell/gre
Hebrew	he	heb
Italian	it	ita
Japanese	ja	jpn
Korean	ko	kor
Norwegian	no	nor
Russian	ru	rus
Spanish	es	esl/spa
Swedish	sv	sve/swe
Turkish	tr	tur

Second Subtag: Namespace

تگ فرعی دوم معمولاً یک توکن استاندارد شده Country است که از مجموعه استانداردهای کد کشور و منطقه ISO 3166 انتخاب شده است. اما ممکن است رشته دیگری نیز باشد که می‌توانید آن را در IANA ثبت کنید. در اینجا قوانین وجود دارد:

اگر subtag دوم دارای:

- دو کاراکتر، یک country/region است که توسط ISO 3166 تعریف شده است.
- سه تا هشت کاراکتر، ممکن است در IANA ثبت شود.
- یک کاراکتر، غیرقانونی^{۲۸} است.

برخی از کدهای کشور ISO 3166 در جدول زیر نشان داده شده است.

²⁸ illegal





Country	Code
Brazil	BR
Canada	CA
China	CN
France	FR
Germany	DE
Holy See (Vatican City State)	VA
Hong Kong	HK
India	IN
Italy	IT
Japan	JP
Lebanon	LB
Mexico	MX
Pakistan	PK
Russian Federation	RU
United Kingdom	GB
United States	US

Remaining Subtags: Namespace

هیچ قانونی برای تگ‌های فرعی سوم و بعدی وجود ندارد، به غیر از حداکثر هشت کاراکتر (حروف و رقم).

Configuring Language Preferences

می‌توانید تنظیمات زبان را در نمایه مرورگر خود پیکربندی کنید.

به شما امکان می‌دهد تنظیمات زبان را از طریق آن تنظیم کنید.

Edit → Preferences... → Languages...

به شما امکان می‌دهد زبان‌ها را به شکل زیر تنظیم کنید.

Tools → Internet Options... → Languages





Internationalized URIs

امروزه، URI ها از بین المللی شدن پشتیبانی زیادی نمی‌کنند. با چند استثنا (با تعریف ضعیف)، URI های امروزی از زیر مجموعه‌ای از کاراکترهای US-ASCII تشکیل شده‌اند. تلاش‌هایی در حال انجام است که ممکن است به ما اجازه دهد مجموعه‌ای غنی‌تر از کاراکترها را در نام میزبان و مسیرهای URL ها بگنجانیم، اما در حال حاضر، این استانداردها به طور گستردگی پذیرفته نشده‌اند یا به کار گرفته نشده‌اند. باید تمرين امروز را مرور کنیم.

Global Transcribability Versus Meaningful Characters

طراحان URI می‌خواستند همه در سراسر جهان بتوانند URI ها را با یکدیگر به اشتراک بگذارند - از طریق ایمیل، تلفن، از طریق بیلیورد، حتی از طریق رادیو. آن‌ها می‌خواستند که URI ها برای استفاده و به خاطر سپردن آسان باشد. این دو هدف با هم تضاد دارند.

برای سهولت در ورود، دستکاری و به اشتراک گذاری URI ها برای مردم در سراسر جهان، طراحان مجموعه بسیار محدودی از کاراکترهای رایج را برای URI ها (حروف الفبای لاتین اصلی، اعداد و چند کاراکتر خاص) انتخاب کردند. این مجموعه کوچک از کاراکترها توسط اکثر نرم افزارها و صفحه کلیدهای سراسر جهان پشتیبانی می‌شود.

متأسفانه، با محدود کردن مجموعه کاراکترها، طراحان URI کار را برای مردم سراسر جهان برای ایجاد URI هایی که استفاده و به خاطر سپردن آسان باشد، بسیار دشوارتر کردند. اکثریت شهروندان جهان حتی الفبای لاتین را نمی‌شناسند و به خاطر سپردن URI ها به عنوان الگوهای انتزاعی تقریباً غیرممکن است.

نویسنده‌گان URI احساس کردند که اطمینان از قابلیت رونویسی^{۲۹} و اشتراک گذاری^{۳۰} شناسه‌های منبع مهم‌تر از این است که آن‌ها از معنی‌دارترین کاراکترها تشکیل شده باشند. بنابراین ما URI هایی داریم که (امروزه) اساساً از یک زیر مجموعه محدود از کاراکترهای ASCII تشکیل شده است.

URI Character Repertoire

زیرمجموعه کاراکترهای US-ASCII مجاز در URI ها را می‌توان به کلاس‌های کاراکتر reserved، escape و unreserved تقسیم کرد. کلاس‌های کاراکتر unreserved را می‌توان به طور کلی در هر مؤلفه‌ای از URI که به آن‌ها اجازه می‌دهد استفاده کرد. کاراکترهای reserved در بسیاری از URI ها معانی خاصی دارند، بنابراین نباید به طور کلی استفاده شوند. جدول زیر را برای لیستی از کاراکترهای reserved و unreserved و escape ببینید.

²⁹ transcribability

³⁰ sharability





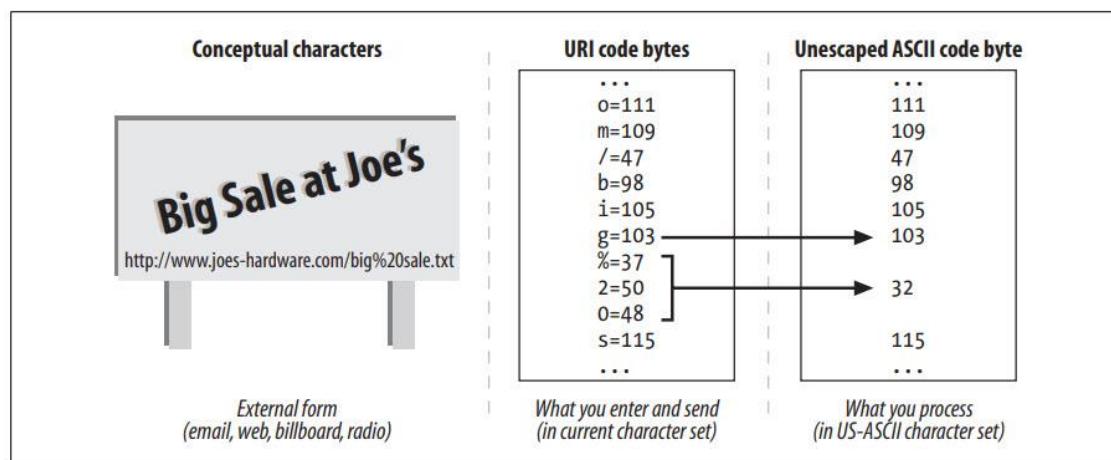
Character class	Character repertoire
Unreserved	[A-Za-z0-9] "-" "_" "." "!" "~" "*" ":" "(" ")"
Reserved	;" "/" "?" ":" "@" "&" "=" "+" "\$" ","
Escape	"%" <HEX> <HEX>

Escaping and Unescaping

راهی برای درج این کاراکترها رزرو شده و سایر کاراکترهای پشتیبانی نشده (مانند فاصله‌ها) در داخل URI ها ارائه می‌دهد. **Escape** یک دنباله سه کاراکتری است که از یک کاراکتر درصد (/) و به دنبال آن دو کاراکتر هگزا دسیمال تشکیل شده است. دو رقم هگزا نشان دهنده کد یک کاراکتر US-ASCII است.

به عنوان مثال، برای درج یک فاصله (ASCII 32) در یک URL، می‌توانید از "%20" استفاده کنید، زیرا ۲۰ نمایش هگزادسیمال ۳۲ است. به طور مشابه، اگر می‌خواهید یک علامت درصد اضافه کنید و آن را به عنوان یک **Escape** در نظر نگیرید، می‌توانید «%25» را وارد کنید، که در آن ۲۵ مقدار هگزادسیمال کد اسکی برای درصد است.

شكل زیر نشان می‌دهد که چگونه کاراکترهای مفهومی یک URI به بایت‌های کاراکترها در مجموعه کاراکترهای فعلی تبدیل می‌شوند. هنگامی که URI برای پردازش مورد نیاز است، **Escape** ها لغو می‌شوند و بایت‌های کد اسکی زیرین را به دست می‌آورند.



در داخل، برنامه‌های HTTP باید URI ها را با **Escape** ها در جای خود حمل و ارسال کنند. برنامه‌های HTTP باید تنها زمانی URI ها را **unescape** نمایند که به داده‌ها نیاز است. و مهمتر از آن، برنامه‌ها باید اطمینان حاصل کنند که هیچ URI دو بار **unescape** نشده است، زیرا علامت‌های درصد که





ممکن است در یک Escape رمزگذاری شده باشند، خود unescape هستند و منجر به از دست رفتن داده‌ها می‌شوند.

Escaping International Characters

توجه داشته باشید که مقادیر Escape باید در محدوده کدهای US-ASCII (۱۲۷-۰) باشد.

برخی از برنامه‌ها سعی می‌کنند از مقادیر Escape برای نمایش کاراکترهای توسعه‌یافته ISO-8859-1 (۱۲۸) استفاده کنند—برای مثال، سرورهای وب ممکن است به اشتباه از Escape‌ها برای کدگذاری نام فایل‌هایی که حاوی کاراکترهای بین‌المللی هستند استفاده کنند. این نادرست است و ممکن است در برخی از برنامه‌ها مشکل ایجاد کند.

برای مثال، نام فایل Sven Ölssen.html (حاوی umlaut) ممکن است توسط یک وب سرور به عنوان Sven%20%D6lssen.html کدگذاری شود.

کدگذاری Space با ۲۰٪ خوب است، اما از نظر فنی کدگذاری Ö با %D6 غیرقانونی است، زیرا کد D6 (اعشار ۲۱۴) خارج از محدوده ASCII قرار دارد.

فقط کدهایی را تا ۰x7F (اعشار ۱۲۷) تعریف می‌کند.

Modal Switches in URLs

برخی از URI‌ها همچنین از دنباله‌هایی از کاراکترهای ASCII برای نشان دادن کاراکترها در مجموعه کاراکترهای دیگر استفاده می‌کنند. به عنوان مثال، رمزگذاری ISO-2022-JP ممکن است برای درج «J» (ESC) برای جابجایی به «ESC» (B) و «JIS-Roman» برای جابجایی به ASCII استفاده شود. این در برخی شرایط محلی کار می‌کند، اما رفتار خوبی نیست. همانطور که نویسنده‌گان RFC 2396 می‌گویند، هیچ طرح استانداردی برای شناسایی کدگذاری خاص مورد استفاده برای URL وجود ندارد.

با این حال، برای دنباله‌های کاراکتر اصلی که حاوی کاراکترهای غیر ASCII هستند، وضعیت دشوارتر است. پروتکل‌های اینترنتی که توالی‌های هشت‌گانه را برای نمایش دنباله‌های کاراکتر ارسال می‌کنند، انتظار می‌رود که در صورت وجود بیش از یک [RFC2277]، راهی برای شناسایی مجموعه کاراکترهای مورد استفاده ارائه دهند.

با این حال، در حال حاضر هیچ شرطی در دستور URI عمومی برای انجام این شناسایی وجود ندارد. یک طرح URI منفرد ممکن است به یک مجموعه کاراکتر واحد نیاز داشته باشد، یک مجموعه کاراکتر





پیش فرض تعریف کند، یا راهی برای نشان دادن مجموعه کاراکتر استفاده شده ارائه دهد. انتظار می‌رود که یک درمان سیستماتیک از کدگذاری کاراکتر در URI به عنوان اصلاح آینده این مشخصات توسعه یابد.

در حال حاضر، URI ها چندان مناسب بین المللی نیستند. هدف قابل حمل بودن URI بر هدف انعطاف پذیری زبان برتری داشت. در حال حاضر تلاش‌هایی برای بین المللی کردن URI ها در حال انجام است، اما در کوتاه مدت، برنامه‌های ASCII HTTP باید از استفاده کنند. از سال ۱۹۶۸ وجود داشته است، بنابراین نمی‌تواند آنقدر بد باشد.

Other Considerations

در این بخش چند مورد دیگر که باید هنگام نوشتن برنامه‌های HTTP بین المللی در نظر داشته باشید، مورد بحث قرار می‌گیرد.

Headers and Out-of-Spec Data

هدرهای HTTP باید از کاراکترهایی از مجموعه کاراکترهای US-ASCII تشکیل شده باشد. با این حال، همه کلاینت‌ها و سرورها این را به درستی اجرا نمی‌کنند، بنابراین ممکن است در موقعي کاراکترهای غیرقانونی با مقادیر کد بزرگتر از ۱۲۷ دریافت کنید.

بسیاری از برنامه‌های HTTP از روال‌های سیستم عامل و کتابخانه برای پردازش کاراکترها استفاده می‌کنند (به عنوان مثال، کتابخانه طبقه بندی کاراکترهای ctype یونیکس). همه این کتابخانه‌ها از کدهای کاراکتر خارج از محدوده ASCII (۰-۱۲۷) پشتیبانی نمی‌کنند.

در برخی شرایط (معمولًاً با پیاده‌سازی‌های قدیمی‌تر)، این کتابخانه‌ها ممکن است نتایج نامناسبی را نشان دهند یا زمانی که کاراکترهای غیر ASCII داده می‌شوند، برنامه را خراب کنند. قبل از استفاده از آن‌ها برای پردازش پیام‌های HTTP، در صورتی که پیام‌ها حاوی داده‌های غیرقانونی باشند، اسناد مربوط به کتابخانه‌های طبقه بندی کاراکترهای خود را به دقت بخوانید.

Dates

مشخصات HTTP به وضوح فرمتهای قانونی تاریخ GMT را تعریف می‌کند، اما توجه داشته باشید که همه سرورهای وب و کلاینت‌ها از قوانین پیروی نمی‌کنند. به عنوان مثال، ما مشاهده کردیم که سرورهای وب، هدرهای تاریخ HTTP نامعتبر را با ماهها به زبان‌های محلی ارسال می‌کنند.





برنامه‌های HTTP باید سعی کنند تاریخ‌های خارج از مشخصات را تحمل کنند و در هنگام دریافت خراب نشوند، اما ممکن است همیشه نتوانند تمام تاریخ‌های ارسال شده را تفسیر کنند. اگر تاریخ قابل تجزیه نیست، سرورها باید محافظه کارانه با آن برخورد کنند.

Domain Names

DNS در حال حاضر از کاراکترهای بین المللی در نام دامنه پشتیبانی نمی‌کند. تلاش‌های استانداردی برای پشتیبانی از نام‌های دامنه چندزبانه در حال انجام است، اما آن‌ها هنوز به طور گسترده گسترش نیافته‌اند.

For More Information

موققیت وب جهانی به این معنی است که برنامه‌های HTTP به تبادل محتوای بیشتر و بیشتر در زبان‌ها و مجموعه کاراکترهای مختلف ادامه خواهند داد. برای اطلاعات بیشتر در مورد موضوع مهم اما کمی پیچیده چند رسانه‌ای چند زبانه، لطفاً به منابع زیر مراجعه کنید.

Appendices

- IANA-registered charset tags are listed in Table H-1.
- IANA-registered language tags are shown in Table G-1.
- ISO 639 language codes are shown in Table G-2.
- ISO 3166 country codes are shown in Table G-3.

Internet Internationalization

<http://www.w3.org/International/>

<http://www.ietf.org/rfc/rfc2396.txt>

CJKV Information Processing Ken Lunde, O'Reilly & Associates, Inc.

<http://www.ietf.org/rfc/rfc2277.txt>

<http://www.iana.org/numbers.htm>

<http://www.ietf.org/rfc/rfc3066.txt>

Codes for the representation of names of languages

Codes for the representation of names of languages—Part 2: Alpha-3 code

Codes for the representation of names of countries





فصل هفدهم - Content Negotiation and Transcoding

اغلب، ممکن است نیاز باشد که یک URL واحد با منابع مختلف مطابقت داشته باشد. نمونه وب سایتی را در نظر بگیرید که می خواهد محتوای خود را به چندین زبان ارائه دهد. اگر سایتی مانند Joe's Hardware کاربران فرانسوی و انگلیسی زبان باشد، ممکن است بخواهد وب سایت خود را به هر دو زبان ارائه دهد. با این حال، اگر چنین باشد، وقتی یکی از مشتریان Joe درخواست مشاهده سایت- «<http://www.joes-hardware.com>» را ارسال می کند، سرور کدام نسخه را باید ارسال کند؟ فرانسوی یا انگلیسی؟

در حالت ایدهآل، سرور نسخه انگلیسی را برای یک انگلیسی زبان و نسخه فرانسوی را برای فرانسوی زبان ارسال می کند—یک کاربر می تواند به صفحه اصلی Joe's Hardware رفته و محتوا را به زبانی که صحبت می کند دریافت کند. خوشبختانه، HTTP روش های مذاکره محتوا^{۳۱} را ارائه می کند که به کلاینتها و سرورها اجازه می دهد دقیقاً چنین تصمیماتی را انجام دهنند. با استفاده از این روش ها، یک URL واحد می تواند با منابع مختلف مطابقت داشته باشد (به عنوان مثال، نسخه فرانسوی و انگلیسی یک صفحه وب). به این نسخه های مختلف گفته می شود Variant.

سرورها همچنین می توانند انواع دیگری از تصمیمات را در مورد اینکه چه محتوایی برای یک URL خاص به کلاینت ارسال می شود، اتخاذ کنند. در برخی موارد، سرورها حتی می توانند به طور خودکار صفحات سفارشی سازی شده را تولید کنند—به عنوان مثال، یک سرور می تواند یک صفحه HTML را به یک صفحه WML برای دستگاه دستی شما تبدیل کند. به این نوع تبدیلات محتوای پویا، Transcoding گفته می شود. آن ها در پاسخ به مذاکره محتوا بین کلاینتها و سرورهای HTTP انجام می شوند.

در این فصل، در مورد مذاکره محتوا و چگونگی انجام وظایف مذاکره محتوا توسط برنامه های کاربردی وب بحث خواهیم کرد.

Content-Negotiation Techniques

سه روش متمایز برای تصمیم گیری اینکه کدام صفحه در یک سرور برای یک کلاینت مناسب است وجود دارد: ارائه انتخاب به کلاینت، تصمیم گیری خودکار در سرور، یا درخواست از یک واسطه برای انتخاب.

این سه تکنیک به ترتیب، Client-driven Negotiation، Server-driven Negotiation و Transparent Negotiation نامیده می شوند (جدول زیر را ببینید).

³¹ content-negotiation





Technique	How it works	Advantages	Drawbacks
Client-driven	Client makes a request, server sends list of choices to client, client chooses.	Easiest to implement at server side. Client can make best choice.	Adds latency: at least two requests are needed to get the correct content.
Server-driven	Server examines client's request headers and decides what version to serve.	Quicker than client-driven negotiation. HTTP provides a q-value mechanism to allow servers to make approximate matches and a Vary header for servers to tell downstream devices how to evaluate requests.	If the decision is not obvious (headers don't match up), the server must guess.
Transparent	An intermediate device (usually a proxy cache) does the request negotiation on the client's behalf.	Offloads the negotiation from the web server. Quicker than client-driven negotiation.	No formal specifications for how to do transparent negotiation.

Client-Driven Negotiation

ساده ترین کاری که یک سرور هنگام دریافت درخواست کلاینت انجام می‌دهد این است که پاسخی را با لیست صفحات موجود ارسال کند و به کلاینت اجازه دهد تصمیم بگیرد کدام یک را می‌خواهد ببیند. این، البته، ساده‌ترین راه برای پیاده‌سازی در سرور است و احتمالاً منجر به انتخاب بهترین نسخه می‌شود (به شرطی که لیست اطلاعات کافی برای انتخاب کپی مناسب را به کلاینت داشته باشد). نقطه ضعف این است که برای هر صفحه دو درخواست لازم است - یکی برای دریافت لیست و دیگری برای دریافت کپی انتخاب شده. این یک روند کند و خسته کننده است و احتمالاً برای کلاینت آزاردهنده خواهد بود.

از نظر مکانیکی، در واقع دو راه برای سرورها وجود دارد که انتخاب‌ها را برای انتخاب به کلاینت ارائه دهند: با ارسال یک سند HTML با لینک‌هایی به نسخه‌های مختلف صفحه و توضیحات هر یک از نسخه‌ها، یا با ارسال یک 1.1 HTTP/1.1. پاسخ با کد پاسخ 300 چند گزینه‌ای.

مرورگر کلاینت ممکن است این پاسخ را دریافت کند و صفحه‌ای با لینک‌ها را مانند روش اول نمایش دهد، یا ممکن است یک پنجره محاوره‌ای ظاهر شود که از کاربر بخواهد انتخابی را انجام دهد. در هر صورت، تصمیم به صورت دستی در سمت کلاینت توسط مرورگر کاربر گرفته می‌شود.

علاوه بر افزایش تأخیر و آزار چندین درخواست در هر صفحه، این روش یک اشکال دیگر نیز دارد: به چندین URL نیاز دارد - یکی برای صفحه اصلی و دیگری برای هر صفحه خاص. بنابراین، اگر درخواست اولیه برای www.joeshardware.com بود، سرور joe ممکن است با صفحه‌ای که لینک‌هایی به www.joes-hardware.com/french و www.joeshardware.com/english دارد، پاسخ دهد. آیا کلاینت‌ها اکنون باید صفحه اصلی یا انتخاب شده را Bookmark کنند؟ آیا آن‌ها باید به





دوستان خود در مورد وب سایت عالی www.joes-hardware.com بگویند یا فقط به دوستان انگلیسی زبان خود در مورد وب سایت www.joes-hardware.com/english بگویند؟

Server-Driven Negotiation

همانطور که در بخش قبل توضیح داده شد، Client-Driven Negotiation دارای چندین اشکال است. بیشتر این اشکالات حول ارتباط افزایش یافته بین کلاینت و سرور برای تصمیم گیری در مورد بهترین صفحه در پاسخ به درخواست است. یکی از راههای کاهش این ارتباطات اضافی این است که به سرور اجازه دهد تصمیم بگیرد کدام صفحه را بازگرداند - اما برای انجام این کار، کلاینت باید اطلاعات کافی در مورد اولویت‌های خود ارسال کند تا به سرور اجازه دهد تصمیمی آگاهانه بگیرد. سرور این اطلاعات را از هدرهای درخواست کلاینت دریافت می‌کند.

دو مکانیسم وجود دارد که سرورهای HTTP برای ارزیابی پاسخ مناسب برای ارسال به کلاینت از آن‌ها استفاده می‌کنند:

- بررسی مجموعه هدرهای Accept کلاینت نگاه می‌کند و سعی می‌کند آن‌ها را با هدرهای پاسخ مربوطه مطابقت دهد.
- در هدرهای دیگر (non-content-negotiation) متفاوت است. به عنوان مثال، سرور می‌تواند پاسخ‌هایی را بر اساس هدر User-Agent کلاینت ارسال کند.

Content-Negotiation Headers

کلاینت‌ها ممکن است اطلاعات ترجیحی خود را با استفاده از مجموعه هدرهای HTTP فهرست شده در جدول زیر ارسال کنند.

Header	Description
Accept	Used to tell the server what media types are okay to send
Accept-Language	Used to tell the server what languages are okay to send
Accept-Charset	Used to tell the server what charsets are okay to send
Accept-Encoding	Used to tell the server what encodings are okay to send

توجه داشته باشید که چقدر این هدرها با هدرهای موجودیت مورد بحث در فصل ۱۵ مشابه هستند. با این حال، تمایز واضحی بین اهداف این دو نوع هدر وجود دارد. همانطور که در فصل ۱۵ ذکر شد، هدرهای موجودیت مانند برچسب‌های حمل و نقل هستند - آن‌ها ویژگی‌های بدنی پیام را مشخص می‌کنند که در طول انتقال پیام‌ها از سرور به کلاینت ضروری است. از سوی دیگر، هدرهای content-negotiation توسط





کلاینت‌ها و سرورها برای تبادل اطلاعات ترجیحی و کمک به انتخاب بین نسخه‌های مختلف یک سند استفاده می‌شود، به طوری که نسخه‌ای که بیشترین تطابق را با ترجیحات کلاینت دارد ارائه می‌شود.

سرورها هدرهای Accept کلاینت‌ها را با هدرهای موجودیت مربوطه، فهرست شده در جدول زیر مطابقت می‌دهند.

Accept header	Entity header
Accept	Content-Type
Accept-Language	Content-Language
Accept-Charset	Content-Type
Accept-Encoding	Content-Encoding

توجه داشته باشید از آنجایی که HTTP یک پروتکل Stateless است (به این معنی که سرورها اولویت‌های کلاینت را در بین درخواست‌ها پیگیری نمی‌کنند)، کلاینت‌ها باید اطلاعات ترجیحی خود را با هر درخواست ارسال کنند.

اگر هر دو کلاینت اطلاعات هدر Accept-Language را ارسال کنند که زبان مورد علاقه آن‌ها را مشخص می‌کند، سرور می‌تواند تصمیم بگیرد که کدام نسخه از www.joes-hardware.com را برای هر کلاینت ارسال کند. اجازه دادن به سرور جهت انتخاب سند برای ارسال که به صورت خودکار انجام شود، تأخیر مرتبط با ارتباط رفت و برگشت مورد نیاز مدل Client-Driven را کاهش می‌دهد.

با این حال، اگر یکی از کلاینت‌ها، بگویید که اسپانیایی را ترجیح می‌دهد، سرور باید کدام نسخه از صفحه را ارسال کند؟ انگلیسی یا فرانسوی؟ سرور فقط دو انتخاب دارد: یا حدس بزند، یا به مدل Client-Driven برگردد و از کلاینت بخواهد انتخاب کند. با این حال، اگر اسپانیایی کمی انگلیسی می‌فهمد، ممکن است صفحه انگلیسی را انتخاب کند - ایده‌آل نیست، اما درست است. در این مورد، اسپانیایی به توانایی انتقال اطلاعات بیشتر در مورد ترجیحات خود نیاز دارد، به این معنی که او حداقل دانش انگلیسی را دارد و در کوتاه مدت، انگلیسی کافی است.

خوبی‌خانه، HTTP مکانیزمی را ارائه می‌کند که به کلاینت‌هایی مانند اسپانیایی ما اجازه می‌دهد تا با استفاده از اختصار «*q values*» توضیحات غنی‌تری از ترجیحات خود ارائه دهند.





Content-Negotiation Header Quality Values

پروتکل HTTP مقادیر Quality را تعریف می‌کند تا به کلاینت‌ها اجازه دهد چندین انتخاب را برای هر دسته از اولویت‌ها فهرست کنند و ترتیب اولویت را با هر انتخاب مرتبط کنند. به عنوان مثال، کلاینت‌ها واند یک هدر Accept-Language از فرم ارسال کنند:

Accept-Language: en;q=0.5, fr;q=0.0, nl;q=1.0, tr;q=0.0

که در آن مقادیر q می‌تواند از 0.0 تا 1.0 باشد (که 0.0 کمترین اولویت و 1.0 بالاترین است). هدر بالا می‌گوید که کلاینت ترجیح می‌دهد نسخه هلندی (nl) سند را دریافت کند، اما یک نسخه انگلیسی (en) این کار را می‌کند. کلاینت تحت هیچ شرایطی نسخه فرانسوی (fr) یا ترکی (tr) را نمی‌خواهد. توجه داشته باشید که ترتیب فهرست بندی اولویت‌ها مهم نیست. فقط مقادیر q مرتبط با آن‌ها مهم هستند.

گاهی اوقات، سرور ممکن است هیچ سندی مطابق با ترجیحات کلاینت نداشته باشد. در این مورد، سرور ممکن است سند را تغییر دهد یا کد را برای مطابقت با ترجیحات کلاینت تغییر دهد.

Varying on Other Headers

سرورها همچنین می‌توانند سعی کنند پاسخ‌ها را با هدرهای درخواست دیگر کلاینت مانند User-Agent مطابقت دهند. سرورها ممکن است بدانند که نسخه‌های قدیمی مرورگر از جاوا اسکریپت پشتیبانی نمی‌کنند، بنابراین ممکن است نسخه‌ای از صفحه را که حاوی جاوا اسکریپت نیست، پس بفرستند.

در این مورد، مکانیزم **q-value** جستجوی تقریبی «بهترین» را انجام نمی‌دهد. سرور یا به دنبال یک تطابق دقیق است یا به سادگی هر آنچه را که دارد (بسته به اجرای سرور) ارائه می‌دهد.

از آنجایی که Cache ها باید سعی کنند نسخه‌های صحیح "بهترین" اسناد ذخیره شده را ارائه دهند، پروتکل HTTP یک هدر **Vary** را تعریف می‌کند که سرور در پاسخ‌ها ارسال می‌کند. هدر Vary به Cache ها (و کلاینت‌ها و هر پراکسی Downstream) می‌گوید که سرور از کدام هدرها برای تعیین بهترین نسخه پاسخ برای ارسال استفاده می‌کند. هدر Vary در ادامه این فصل با جزئیات بیشتر مورد بحث قرار می‌گیرد.

Content Negotiation on Apache

در اینجا به مروری بر نحوه پشتیبانی وب سرور آپاچی از content negotiation پرداخته می‌شود. این به ارائه‌دهنده محتوای وب‌سایت - برای مثال Joe - بستگی دارد که نسخه‌های مختلف صفحه Index مربوط به Joe را ارائه کند. Joe باید تمام فایل‌های صفحه Index خود را در دایرکتوری مناسب روی سرور آپاچی مربوط به وب سایت خود قرار دهد.





دو راه برای فعال کردن content negotiation در اینجا وجود دارد:

- در دایرکتوری وب سایت، یک فایل type-map برای هر URI در وب سایت که دارای انواع مختلف است ایجاد کنید. فایل type-map همه انواع و هدرهای content-negotiation را فهرست می‌کند.
- دستورالعمل MultiViews را فعال کنید، که باعث می‌شود Apache فایلهای type-map را برای دایرکتوری به طور خودکار ایجاد نماید.

Using type-map files

سرور آپاچی باید بداند که فایلهای type-map چه شکلی هستند. برای پیکربندی این موضوع، یک کنترلر در فایل پیکربندی سرور تنظیم کنید که پسوند فایل را برای فایلهای type-map مشخص می‌کند. مثلا:

AddHandler type-map .var

این خط نشان می‌دهد که فایلهایی با پسوند var. فایلهای type-map هستند.

در اینجا یک نمونه فایل type-map آمده است:

URI: joes-hardware.html

URI: joes-hardware.en.html

Content-type: text/html

Content-language: en

URI: joes-hardware.fr.de.html

Content-type: text/html; charset=iso-8859-2

Content-language: fr, de

بوسیله این فایل type-map، سرور آپاچی می‌داند که joes-hardware.en.html را برای کلاینت‌هایی که انگلیسی درخواست می‌کنند و joes-hardware.fr.de.html را برای کلاینت‌هایی که فرانسوی درخواست می‌کنند ارسال کند. لازم به ذکر است که مقادیر Quality نیز پشتیبانی می‌شوند.

Using MultiViews

برای استفاده از MultiViews، باید آن را برای دایرکتوری حاوی وب سایت، با استفاده از دستورالعمل Options (<Directory>, <Location>, or <Files>) در بخش مناسب فایل access.conf فعال کنید.

اگر MultiViews فعال باشد و یک مرورگر منبعی به نام joes-hardware را درخواست کند، سرور به دنبال همه فایلهایی می‌گردد که «joes-hardware» در نام آنها وجود دارد و یک فایل type-





برای آن‌ها ایجاد می‌کند. بر اساس نام‌ها، سرور هدرهای **content-negotiation** مناسب را که فایل‌ها با آن‌ها مطابقت دارند، حدس می‌زند. برای مثال، نسخه فرانسوی زبان `joes-hardware.fr` باید حاوی `.fr` باشد.

Server-Side Extensions

راه دیگر برای اجرای **content negotiation** در سرور، از طریق افزونه‌های سمت سرور، مانند Microsoft's Active Server Pages (ASP) است.

Transparent Negotiation

به دنبال این است که بار مذاکره مبتنی بر سرور را از سرور دور کند، در حالی که تبادل پیام با کلاینت را با داشتن یک پروکسی واسطه از طرف کلاینت به حداقل می‌رساند. فرض بر این است که پروکسی از انتظارات کلاینت آگاهی دارد و می‌تواند مذاکرات را از طرف خود انجام دهد (پراکسی انتظارات کلاینت را در درخواست محتوا دریافت کرده است). برای پشتیبانی از مذاکره محتوای شفاف^{۳۲}، سرور باید بتواند به پراکسی‌ها بگوید که سرور چه هدرهای درخواستی را بررسی می‌کند تا بهترین تطابق را برای درخواست کلاینت تعیین کند. مشخصات 1.1/HTTP هیچ مکانیزمی را برای **Transparent Negotiation** تعریف نمی‌کند، اما هدر `Vary` را تعریف می‌کند. سرورها هدرهای `Vary` را در پاسخ‌های خود ارسال می‌کنند تا به واسطه‌ها بگویند از چه هدرهایی برای **content negotiation** استفاده می‌کنند.

پروکسی‌های Cache می‌توانند کپی‌های مختلفی از اسنادی را که از طریق یک URL به آن‌ها دسترسی دارند ذخیره کنند. اگر سرورها فرآیندهای تصمیم گیری خود را به Cache‌ها منتقل کنند، Cache‌ها می‌توانند از طرف سرورها با کلاینت‌ها مذاکره کنند. Cache‌ها همچنین مکان‌های بسیار خوبی برای رمزگذاری محتوا هستند، زیرا یک رمزگذار همه منظوره مستقر در Cache می‌تواند محتوای هر سروری، نه فقط یک سرور را رمزگذاری کند.

Caching and Alternates

محتوا فرض می‌کند که محتوا می‌تواند بعداً دوباره استفاده شود. با این حال، Cache‌ها باید بسیاری از منطق تصمیم گیری را که سرورها هنگام ارسال پاسخ انجام می‌دهند، به کار گیرند تا اطمینان حاصل کنند که پاسخ ذخیره شده صحیح را به درخواست کلاینت ارسال می‌کنند.

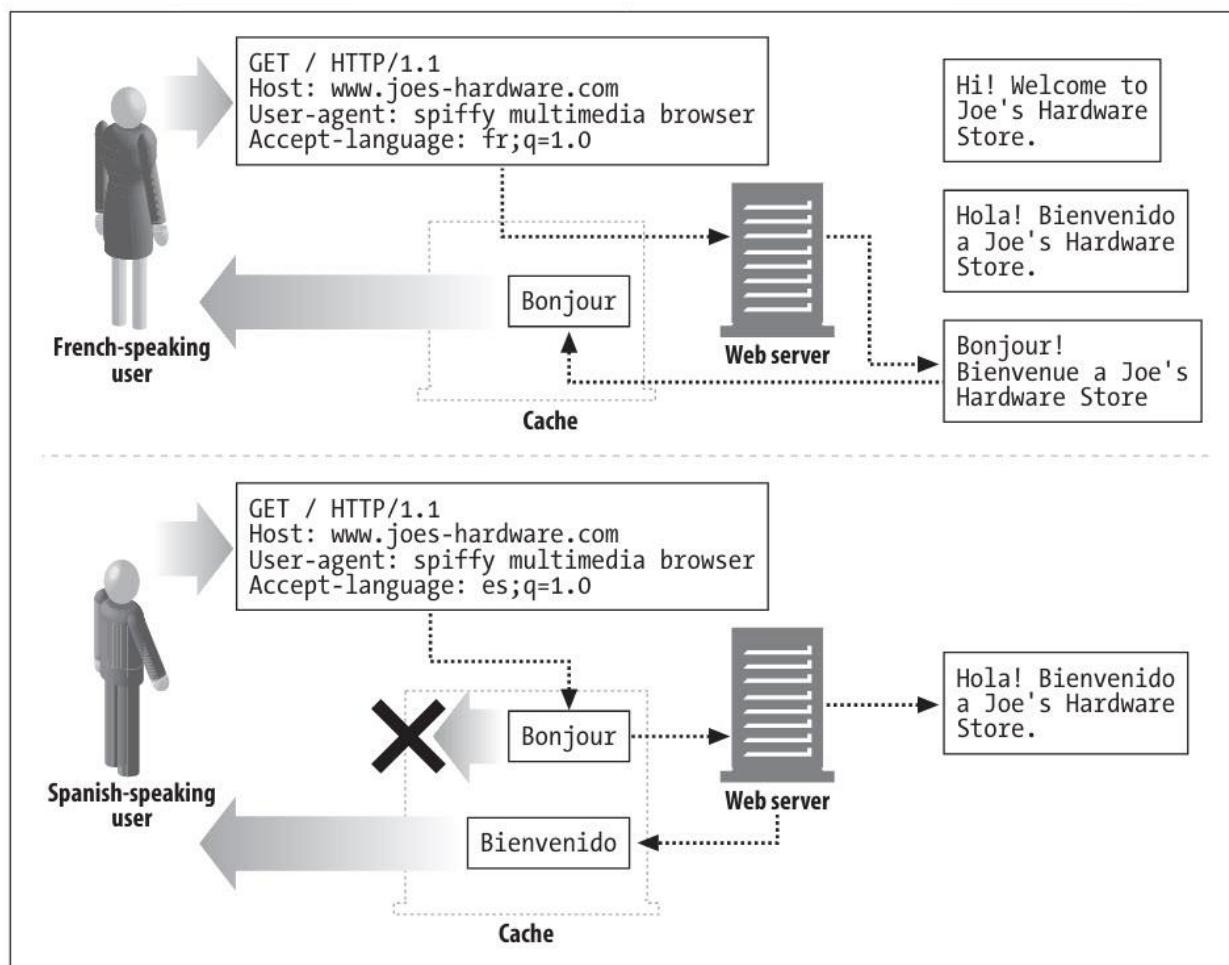
³² transparent content negotiation





بخش قبلی هدرهای Accept ارسال شده توسط کلاینتها و هدرهای موجودیت مربوطه را که سرورها آنها را با آنها مطابقت می‌دهند توضیح داد تا بهترین پاسخ را برای هر درخواست انتخاب کنند. Cache‌ها باید از همین هدرها استفاده کنند تا تصمیم بگیرند کدام پاسخ ذخیره شده در Cache را ارسال کنند.

شكل زیر توالی صحیح و نادرست عملیات مربوط به Cache را نشان می‌دهد. اولین درخواست باعث می‌شود که درخواست را به سرور ارسال کرده و پاسخ را ذخیره کند. پاسخ دوم توسط Cache جستجو می‌شود و سندی مطابق با URL پیدا می‌شود. این سند اما به زبان فرانسه است و درخواست کننده یک سند اسپانیایی می‌خواهد. اگر Cache فقط سند فرانسوی را به درخواست کننده بازگرداند، رفتار نادرست خواهد داشت.



بنابراین Cache باید درخواست دوم را نیز به سرور ارسال کند و پاسخ و پاسخ جایگزین^{۳۳} را برای آن URL ذخیره کند. اکنون دارای دو سند مختلف برای یک URL است، درست مانند سرور. به این نسخه‌های

^{۳۳} alternate





مختلف، variant یا alternate می‌گویند. مذاکره محتوا را می‌توان به عنوان فرآیند انتخاب، از بین variants بهترین تطابق برای درخواست کلاینت در نظر گرفت.

The Vary Header

در اینجا یک مجموعه معمولی از هدرهای درخواست و پاسخ از یک مرورگر و سرور آورده شده است:

```
GET http://www.joes-hardware.com/ HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.73 [en] (WinNT; U)
Host: www.joes-hardware.com
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, image/png, */
Accept-Encoding: gzip
Accept-Language: en, pdf
Accept-Charset: iso-8859-1, *, utf-8
```

```
HTTP/1.1 200 OK
Date: Sun, 10 Dec 2000 22:13:40 GMT
Server: Apache/1.3.12 OpenSSL/0.9.5a (Unix) FrontPage/4.0.4.3
Last-Modified: Fri, 05 May 2000 04:42:52 GMT
Etag: "1b7ddf-48-3912514c"
Accept-Ranges: Bytes
Content-Length: 72
Connection: close
Content-Type: text/html
```

با این حال، چه اتفاقی می‌افتد، اگر تصمیم سرور بر اساس هدرهای دیگری غیر از هدرهای Accept باشد، مانند User-Agent؟

این به آن اندازه که به نظر می‌رسد رادیکال نیست. سرورها ممکن است بدانند که نسخه‌های قدیمی یک مرورگر، برای مثال، جوا اسکریپت را پشتیبانی نمی‌کنند و بنابراین ممکن است نسخه‌ای از صفحه را که جوا اسکریپت در آن وجود ندارد، برگردانند. اگر سرورها از هدرهای دیگر برای تصمیم گیری در مورد اینکه کدام صفحات را ارسال کنند استفاده می‌کنند، Cache‌ها باید بدانند که آن هدرها چیست، تا بتوانند منطق موازی را در انتخاب صفحه ذخیره شده برای ارسال مجدد انجام دهند.

هدر پاسخ HTTP Vary همه هدرهای درخواست کلاینت را که سرور برای انتخاب سند یا تولید محتوای سفارشی در نظر می‌گیرد (علاوه بر هدرهای content-negotiation معمولی) فهرست می‌کند. به

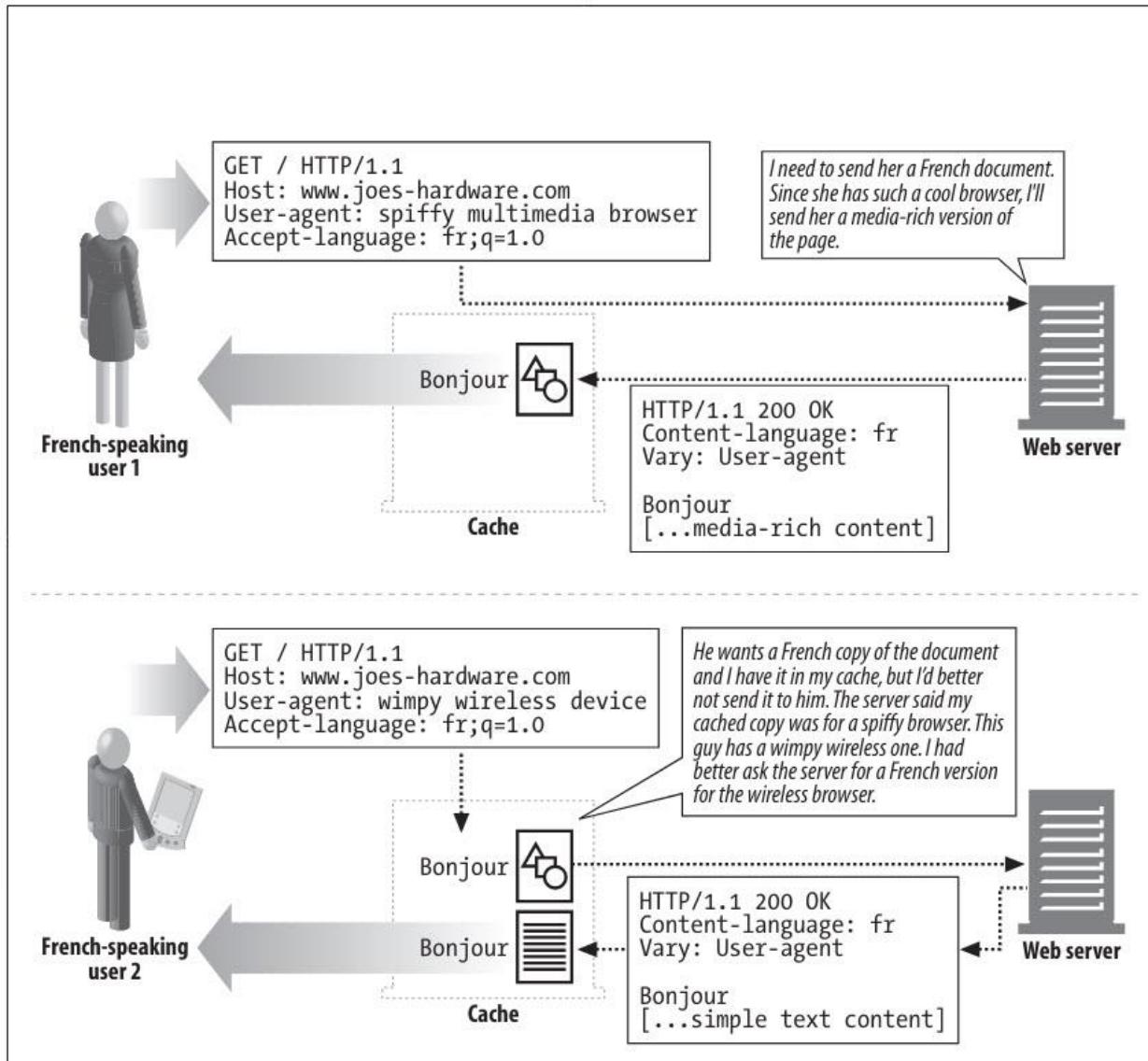




عنوان مثال، اگر سند ارائه شده به هدر User-Agent بستگی دارد، هدر Vary باید شامل "User-Agent" باشد.

هنگامی که یک درخواست جدید وارد می‌شود، Cache با استفاده از هدرهای content-negotiation تطابق را پیدا می‌کند. با این حال، قبل از اینکه بتواند این سند را به کلاینت ارائه دهد، باید ببیند که آیا سرور یک هدر Vary در پاسخ ذخیره شده در Cache ارسال کرده است یا خیر. اگر هدر Vary وجود داشته باشد، مقادیر هدرها در درخواست جدید باید با مقادیر هدر در درخواست ذخیره شده قدیمی مطابقت داشته باشند. از آنجایی که سرورها ممکن است پاسخهای خود را بر اساس هدرهای درخواست کلاینت تغییر دهند، Cache‌ها باید هم هدرهای درخواست کلاینت و هم هدرهای پاسخ سرور مربوطه را با هر متغیر ذخیره شده ذخیره کنند تا بتوانند را اجرا کنند. این در شکل زیر نشان داده شده است.





اگر هدر `Vary` سرور به این شکل باشد، تعداد زیادی از مقادیر مختلف `User-Agent` و `Cookie` می‌تواند انواع مختلفی را ایجاد کند:

`Vary: User-Agent, Cookie`

یک Cache باید هر نسخه سند مربوط به هر نوع را ذخیره کند. هنگامی که Cache جستجو را انجام می‌دهد، ابتدا محتوا را با هدرهای `content-negotiation` مطابقت می‌دهد، سپس نوع درخواست را با انواع مطابقت می‌دهد. اگر مطابقت وجود نداشته باشد، Cache سند را از سرور مبدا واکشی می‌کند.





Transcoding

ما در مورد مکانیزمی که توسط آن کلاینتها و سرورها می‌توانند بین مجموعه‌ای از اسناد برای URL انتخاب کنند و اسنادی را که به بهترین وجه با نیازهای کلاینت منطبق است را ارسال کنند، با جزئیات صحبت کرده‌ایم. این مکانیسم‌ها بر وجود اسنادی تکیه می‌کنند که با نیازهای کلاینت مطابقت دارند - خواه کاملاً با نیازها مطابقت داشته باشند یا نه.

با این حال، چه اتفاقی می‌افتد، وقتی یک سرور اصلاً سندی مطابق با نیازهای کلاینت نداشته باشد؟ سرور ممکن است مجبور باشد با خطأ پاسخ دهد، اما از نظر تئوری، سرور ممکن است بتواند یکی از اسناد موجود خود را به چیزی تبدیل کند که کلاینت بتواند از آن استفاده کند. به این گزینه Transcoding می‌ویند.

جدول زیر برخی از Transcoding های فرضی را فهرست می‌کند.

Before	After
HTML document	WML document
High-resolution image	Low-resolution image
Image in 64K colors	Black-and-white image
Complex page with frames	Simple text page without frames or images
HTML page with Java applets	HTML page without Java applets
Page with ads	Page with ads removed

سه دسته از Content و Information Synthesis وجود دارد: Transcoding و Format Conversion و Injection.

Format Conversion

Format Conversion عبارت است از تبدیل داده‌ها از یک فرمت به فرمت دیگر برای قابل مشاهده شدن توسط کلاینت. دستگاه بی‌سیمی که به دنبال دسترسی به سندی است که معمولاً توسط یک کلاینت دسکتاب مشاهده می‌شود، ممکن است با تبدیل HTML به WML این کار را انجام دهد. کلاینت که از طریق یک لینک آهسته^{۳۴} به یک صفحه وب دسترسی پیدا می‌کند که علاقه زیادی به تصاویر با وضوح بالا ندارد، ممکن است بتواند یک صفحه غنی از تصویر را راحت‌تر مشاهده کند، اگر تصاویر با تبدیل آن‌ها از رنگی به سیاه و سفید از نظر اندازه و وضوح کاهش پیدا کنند.

³⁴ slow link





تبديل قالب توسط هدرهای Content-Negotiation Headers در جدول content-negotiation پیشتر به آن اشاره شد هدایت می‌شود، اگرچه ممکن است توسط هدر User-Agent نیز هدایت شود. توجه داشته باشید که Transfer با Content Encoding یا Transcoding یا Content Transformation متفاوت است، زیرا دو مورد دوم معمولاً برای انتقال کارآمدتر یا این محتوا استفاده می‌شوند، در حالی که اولی برای قابل مشاهده کردن محتوا در دستگاه دسترسی استفاده می‌شود.

Information Synthesis

استخراج قطعات کلیدی اطلاعات از یک سند – معروف به Information Synthesis – می‌تواند یک فرآیند section heading Transcoding مفید باشد. یک مثال ساده از این، تولید طرح کلی از یک سند بر اساس یا حذف تبلیغات و لوگوها از یک صفحه است.

فناوری‌های پیچیده‌تر که صفحات را بر اساس کلمات کلیدی در محتوا دسته‌بندی می‌کنند نیز در خلاصه کردن^{۳۵} ماهیت یک سند مفید هستند. این فناوری اغلب توسط سیستم‌های طبقه‌بندی خودکار صفحات وب^{۳۶}، مانند فهرست راهنمای صفحات وب در سایتها پورتال استفاده می‌شود.

Content Injection

دو دسته از Transcoding‌هایی که تاکنون توضیح داده شد، معمولاً مقدار محتوا را در اسناد وب کاهش می‌دهند، اما دسته دیگری از Transformation‌ها وجود دارند که میزان محتوا را افزایش می‌دهند: Content-Injection Transcoding، نمونه‌هایی از Injection Transcoding خودکار و سیستم‌های ردیابی کاربر^{۳۷} هستند.

جدایت (و تخلف) یک Transcoder درج آگهی را تصور کنید که به طور خودکار تبلیغات را به هر صفحه HTML اضافه می‌کند. Transcoding از این نوع باید پویا باشد – برای اینکه در افزودن تبلیغاتی که در حال حاضر مرتبط هستند یا به نحوی برای یک کاربر خاص هدف قرار گرفته‌اند، مؤثر باشد، باید در لحظه انجام شود. همچنین می‌توان سیستم‌های ردیابی کاربر را برای افزودن محتوا به صفحات به صورت پویا و به منظور جمع‌آوری آمار در مورد نحوه مشاهده صفحه و نحوه گشت و گذار کلاینت‌ها در وب ایجاد کرد.

³⁵ summarizing

³⁶ web page-classification

³⁷ user-tracking systems



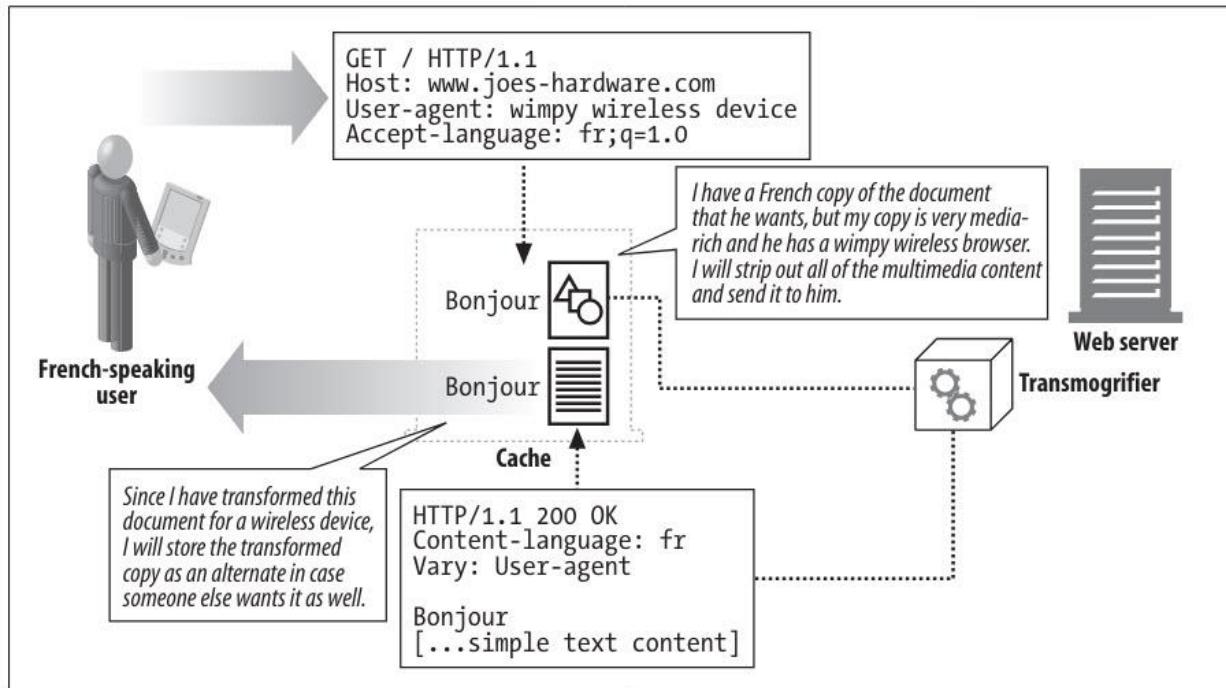


Transcoding Versus Static Pregeneration

یک جایگزین برای Transcoding، ساختن کپی‌های مختلف از صفحات وب در سرور وب است - برای مثال، یکی با HTML، یکی با WML، یکی با تصاویر با وضوح بالا، یکی با تصاویر با وضوح پایین، یکی با محتوای چند رسانه‌ای، و دیگری بدون آن. با این حال، به دلایل بسیاری، این یک تکنیک چندان کاربردی نیست: هر تغییر کوچکی در یک صفحه نیاز به اصلاح چندین صفحه دارد، فضای بیشتری برای ذخیره تمام نسخه‌های مختلف هر صفحه لازم است، و فهرست نویسی صفحات و برنامه نویسی وب سرورها برای سرویس دهی مناسب سخت‌تر است. برخی از Transcoding‌ها، مانند درج آگهی (مخصوصاً درج آگهی هدفمند)، نمی‌توانند به صورت ایستا انجام شوند—تبليغ درج شده به درخواست کاربر از صفحه بستگی دارد.

تبديل سريع یک صفحه root می‌تواند راه حل ساده‌تری نسبت به پیش تولید استاتیک باشد. با این حال، ممکن است به قیمت افزایش تاخیر در ارائه محتوا باشد. با این حال، برخی از این محاسبات را می‌توان توسط شخص ثالث انجام داد، در نتیجه محاسبات را از وب سرور تخلیه می‌کند - تغییر شکل می‌تواند توسط یک عامل خارجی در یک پروکسی یا Cache انجام شود. شکل زیر Transcoding را در یک Cache پراکسی نشان می‌دهد.





Next Steps

داستان Content Negotiation به چند دلیل به هدرهای Content و Accept ختم نمی‌شود:

- برای محتوای مناسب، یا تلاش برای "حدس زدن" بهترین تطابق، می‌تواند پرهزینه باشد. آیا راههایی برای ساده سازی و تمرکز پروتکل Content Negotiation وجود دارد؟ RFC‌های ۲۲۹۵ و ۲۲۹۶ تلاش می‌کنند تا به این سوال برای Transparent HTTP Content Negotiation پاسخ دهند.
- HTTP نه تنها پروتکلی نیست که نیاز به Content Negotiation دارد. Fax و Streaming Media هم Content Negotiation را در آن کلاینت و سرور باید بهترین پاسخ را به درخواست مشتری مورد بحث قرار دهند. آیا می‌توان یک پروتکل مذاکره محتوای عمومی را در بالای پروتکل‌های کاربردی TCP/IP توسعه داد؟ کارگروه Content Negotiation برای رسیدگی به این سوال تشکیل شد. این گروه اکنون بسته شده است، اما چندین RFC را ارائه کرده است.

For More Information

پیش‌نویس‌های اینترنتی و اسناد آنلاین زیر می‌توانند جزئیات بیشتری در مورد Content Negotiation به شما ارائه دهند:

<http://www.ietf.org/rfc/rfc2616.txt>





<http://search.ietf.org/rfc/rfc2295.txt>

<http://search.ietf.org/rfc/rfc2296.txt>

<http://search.ietf.org/rfc/rfc2936.txt>

<http://www.imc.org/ietf-medfree/index.htm>





فصل هجدهم – Web Hosting

هنگامی که منابع را روی یک وب سرور عمومی قرار می‌دهید، در واقع آن‌ها را در دسترس جامعه اینترنتی قراردادهاید. این منابع می‌توانند به سادگی فایل‌های متنی یا تصاویر، یا به پیچیدگی نقشه‌های راندگی در زمان واقعی یا دروازه‌های خرید تجارت الکترونیکی باشند. بسیار مهم است که این تنوع غنی از منابع، متعلق به سازمان‌های مختلف، به راحتی در وب‌سایتها منتشر شود و بر روی سرورهای وب قرار داده شود که عملکرد خوبی را با قیمت منصفانه ارائه می‌دهند.

وظایف جمعی ذخیره سازی، واسطه گری و مدیریت منابع محتوا را web Hosting یکی از وظایف اصلی یک وب سرور است. برای نگهداری، سرویس دهی، ورود به سیستم و مدیریت محتوای خود به یک سرور نیاز دارد. اگر نمی‌خواهید سخت‌افزار و نرم‌افزار مورد نیاز را خودتان مدیریت کنید، به یک سرویس Hoster یا Hosting Service نیاز دارید. Hoster ها سرویس و خدمات مدیریت وب سایت را به شما اجاره داده و درجات مختلفی از امنیت، گزارش دهی و سهولت استفاده را ارائه می‌دهند. Hoster ها معمولاً وب‌سایتها را روی سرورهای وب سنگین برای کارآمدی، قابلیت اطمینان و عملکرد جمع می‌کنند.

این فصل برخی از مهم ترین ویژگی‌های خدمات میزبانی وب و نحوه تعامل آن‌ها با برنامه‌های کاربردی HTTP را توضیح می‌دهد. به طور خاص، این فصل شامل موارد زیر است:

- چگونه وب‌سایتها مختلف را می‌توان به صورت مجازی روی یک سرور میزبانی کرد و چگونه این امر بر HTTP تأثیر می‌گذارد.
- چگونه می‌توان وب سایتها را تحت ترافیک سنگین قابل اعتماد تر کرد.
- چگونه وب سایتها را سریعتر بارگذاری کنیم.

Hosting Services

در روزهای اولیه شبکه جهانی وب، سازمان‌های فردی سخت‌افزار کامپیوتر خود را خریداری کردند، اتاق‌های کامپیوتري خود را ساختند، اتصالات شبکه خود را به دست آوردند و نرم‌افزار وب سرور خود را مدیریت کردند.

از آنجایی که وب به سرعت به جریان اصلی دنیا تبدیل شد، همه می‌خواستند یک وب‌سایت داشته باشند، اما تعداد کمی از مردم مهارت یا زمان لازم برای ساخت اتاق‌های سرور مجهز به تهווیه مطبوع، ثبت نام دامنه، میزبانی وب یا خرید پهنانی باند شبکه را داشتند.

برای صرفه جویی در زمان، بسیاری از مشاغل جدید ظهرور کردند که خدمات میزبانی وب با مدیریت حرفه‌ای را ارائه می‌دادند. بسیاری از سطوح خدمات در دسترس هستند، از مدیریت امکانات فیزیکی





(تامین فضاء، تهويه مطبوع و کابل کشی) تا میزبانی وب با خدمات کامل، که در آن تمام کاری که مشتری انجام می‌دهد ارائه محتوا است.

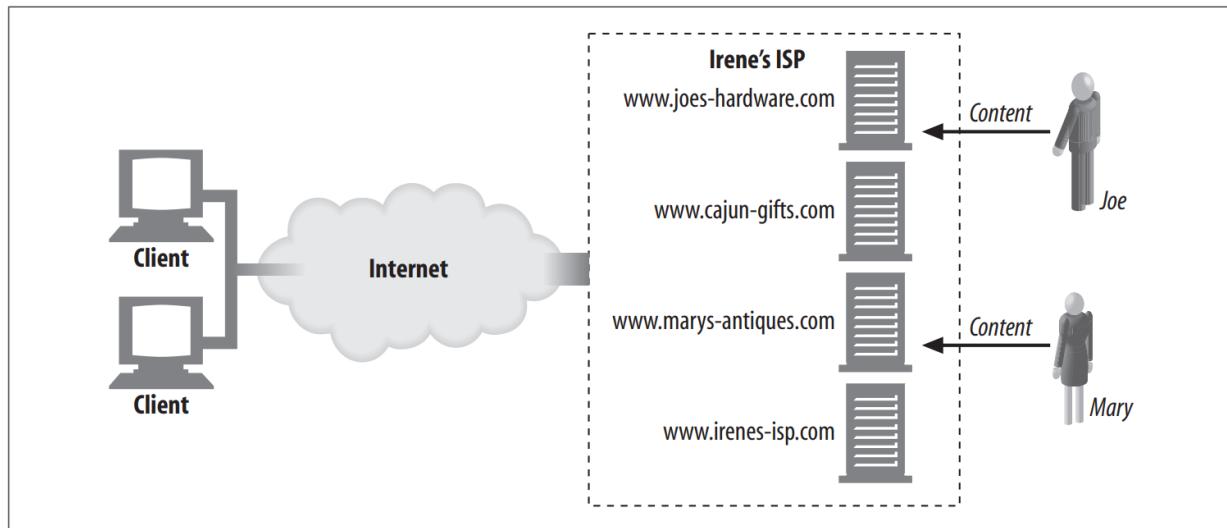
این فصل بر آنچه که سرور وب میزبان ارائه می‌دهد تمرکز دارد. بسیاری از مواردی که باعث می‌شود یک وب سایت کار کند. به عنوان مثال، توانایی در پشتیبانی از زبان‌های مختلف و توانایی آن برای انجام معاملات تجارت الکترونیک ایمن به قابلیت‌هایی بستگی دارد که سرور میزبان وب از آن پشتیبانی می‌کند.

A Simple Example: Dedicated Hosting

فرض کنید که Mary's Antique Auction و Joe's Hardware Online هر دو وب سایت‌هایی با حجم نسبتاً بالا می‌خواهند. ISP Irene دارای Rack هایی پر از وب سرورهای یکسان و با کارایی بالا است که می‌تواند به جو و مری اجاره داده شود. این دو به جای اینکه سرورهای خود را خریداری کنند و نرم افزار سرور را در آن قرار دهند، می‌توانند از خدمات این شرکت استفاده نمایند.

در شکل زیر، جو و مری هر دو برای سرویس میزبانی وب اختصاصی ارائه شده توسط ISP Irene ثبت نام می‌کنند. جو یک وب سرور اختصاصی را اجاره می‌کند که توسط ISP Irene خریداری و نگهداری می‌شود. مری یک سرور اختصاصی متفاوت از ISP Irene دریافت می‌کند. ISP Irene می‌تواند سختافزار سرور را با حجم خریداری کند و می‌تواند سختافزاری را انتخاب کند که قابل اعتماد، تست شده و کم‌هزینه باشد. اگر محبوبیت Joe's اضافی به Mary یا Joe می‌تواند بلافاصله سرورهای ISP Irene افزایش یابد، Mary's Antique Auction یا Hardware Online





در این مثال، مرورگرها درخواست‌های HTTP را برای www.joes-hardware.com به آدرس IP سرور Joe و درخواست‌های www.marys-antiques.com را به آدرس IP (متفاوت) سرور Mary ارسال می‌کنند.

Virtual Hosting

بسیاری از مردم می‌خواهند در وب حضور داشته باشند اما وب سایت‌های پربازدید ندارند. برای این افراد، ارائه یک وب سرور اختصاصی ممکن است بیهوده باشد، زیرا آن‌ها صدها دلار در ماه برای اجاره سروری که عمدتاً بیکار است پرداخت می‌کنند!

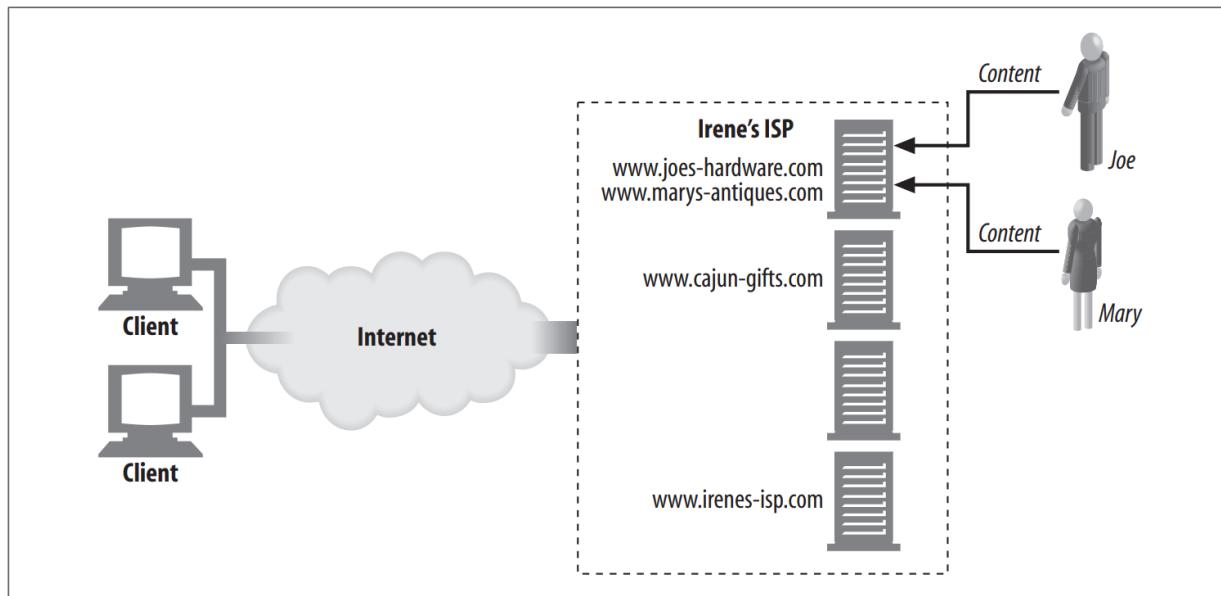
بسیاری از میزبان‌های وب با به اشتراک گذاشتن یک کامپیوتر بین چندین مشتری، خدمات میزبانی وب ارزان‌تری را ارائه می‌دهند. به این مدل Virtual Hosting یا Shared Hosting می‌گویند. به نظر می‌رسد هر وب سایت توسط سرور متفاوتی میزبانی می‌شود، اما آن‌ها واقعاً بر روی یک سرور فیزیکی میزبانی می‌شوند. از دیدگاه کاربر نهایی، وب‌سایت‌هایی که به صورت مجازی میزبانی می‌شوند باید از سایت‌هایی که روی سرورهای اختصاصی جداگانه میزبانی می‌شوند، قابل تشخیص نباشند.

به دلیل صرفه جویی در هزینه، فضا و دلایل مدیریتی، یک شرکت میزبان مجازی می‌خواهد دهها، صدها یا هزاران وب سایت را بر روی یک سرور میزبانی کند - اما این لزوماً به این معنی نیست که 1000 وب سایت فقط از یک رایانه شخصی ارائه می‌شوند. میزبان‌ها می‌توانند بانک‌هایی از سرورهای تکراری (به نام Server Farm) ایجاد کنند و بار را در Server Farm‌ها پخش کنند. از آنجایی که هر سرور در Farm شبیه سازی از سرورهای دیگر است و میزبان بسیاری از وب سایت‌های مجازی است، مدیریت بسیار آسان‌تر است. (در فصل 20 در مورد Server Farm‌ها بیشتر صحبت خواهیم کرد).





هنگامی که جو و مری کسب و کار خود را شروع کردند، ممکن بود تا زمانی که سطح ترافیک آنها به شکل مطلوب رسیده و خرید سرور اختصاصی به صرفه باشد، میزبانی مجازی را برای صرفه جویی در هزینه انتخاب کنند.

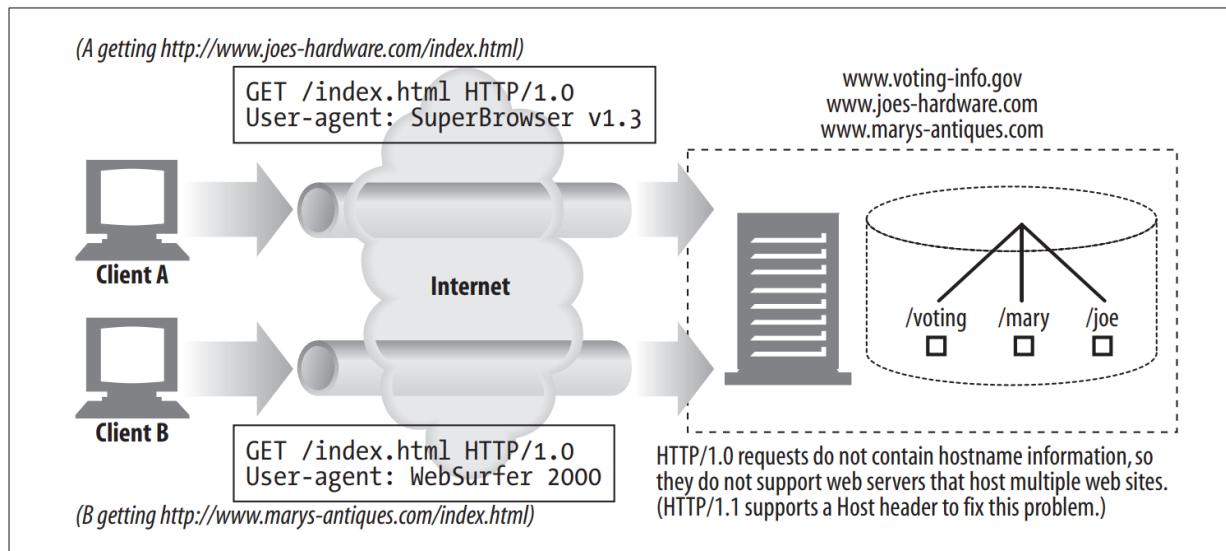


Virtual Server Request Lacks Host Information

متأسفانه، یک نقص طراحی در HTTP/1.0 وجود داشت که باعث می‌شود هاستهای مجازی در این مدل قابل استفاده نباشند. مشخصات HTTP/1.0 هیچ وسیله‌ای را برای سرورهای وب اشتراکی فراهم نمی‌کند تا تشخیص دهنده به کدام یک از وب سایتها مجازی میزبانی شده هدایت شوند.

به یاد داشته باشید که درخواست‌های HTTP/1.0 فقط اجزای مربوط به مسیر URL را در پیام درخواست ارسال می‌کنند. اگر سعی کنید `http://www.joes-hardware.com/index.html` را دریافت کنید، مرورگر به سرور `www.joes-hardware.com` متصل می‌شود، اما درخواست `HTTP/1.0 GET "/index.html"` می‌گوید. بدون ذکر نام میزبان. اگر سرور به صورت مجازی چندین سایت را میزبانی می‌کند، این اطلاعات کافی نیست تا بفهمید به چه وب سایت مجازی دسترسی دارد.





به عنوان مثال، در شکل بالا:

اگر مشتری A سعی کند به <http://www.joes-hardware.com/index.html> دسترسی پیدا کند، درخواست "GET /index.html" به وب سرور مشترک ارسال می‌شود.

اگر مشتری B سعی کند به <http://www.marys-antiques.com/index.html> دسترسی پیدا کند، درخواست یکسان "GET /index.html" به وب سرور مشترک ارسال می‌شود.

تا آنجا که به وب سرور مربوط می‌شود، اطلاعات کافی برای تعیین اینکه به کدام وب سایت در حال دسترسی است وجود ندارد! این دو درخواست یکسان به نظر می‌رسند، حتی اگر برای اسناد کاملاً متفاوت (از وب سایتها مختلف) باشند. مشکل این است که اطلاعات میزبان وب سایت از درخواست حذف شده است.

همانطور که در فصل 6 دیدیم، جایگزین‌های HTTP (Intercepting Proxy و Reverse Proxy) نیز به اطلاعات تعیین کننده سایت نیاز دارند.

Making Virtual Hosting Work

اطلاعات میزبان گمشده نادیده گرفته شده در مشخصات HTTP اصلی بود که به اشتباه فرض می‌کرد که هر وب سرور دقیقاً یک وب سایت را میزبانی می‌کند. طراحان HTTP از سرورهای میزبانی مجازی و مشترک پشتیبانی نمی‌کنند. به همین دلیل، اطلاعات نام میزبان در URL به عنوان زائد مشاهده شد و حذف شد. فقط باید اجزای مسیر ارسال شود.





از آنجایی که مشخصات اولیه برای میزبانی مجازی پیش بینی نشده بود، میزبانهای وب نیاز به ایجاد راه حل ها و قراردادهایی برای پشتیبانی از میزبانی مجازی مشترک داشتند. مشکل را می‌توان به سادگی با اجباری نمودن تمام پیامهای درخواست HTTP با ارسال کل URL به جای ارسال اجزای مسیر، حل کرد.

HTTP/1.1 به سرورها نیاز دارد تا URL های کامل را در خطوط درخواست پیامهای HTTP مدیریت کنند، اما زمان زیادی طول می‌کشد تا همه برنامه‌های قدیمی به این مشخصات ارتقا داده شوند. در این میان، چهار تکنیک به وجود آمده است:

Virtual hosting by URL path

افزودن یک جزء مسیر خاص به URL تا سرور بتواند سایت را تعیین کند.

Virtual hosting by port number

اختصاص یک شماره پورت متفاوت به هر سایت، بنابراین درخواست‌ها توسط نمونه‌های جداگانه‌ای از وب سرور بررسی می‌شوند.

Virtual hosting by IP address

اختصاص آدرس‌های IP مختلف برای سایتها مجازی مختلف و اتصال تمام آدرس‌های IP به یک ماشین واحد. این موضوع به سرور وب اجازه می‌دهد تا نام سایت را با آدرس IP شناسایی کند.

Virtual hosting by Host header

بسیاری از میزبانان وب، طراحان HTTP را برای حل این مشکل تحت فشار قرار دادند. نسخه‌های پیشرفته HTTP/1.0 و نسخه رسمی HTTP/1.1 یک هدر درخواست میزبان را تعریف می‌کنند که نام سایت را دارد. وب سرور می‌تواند سایت مجازی را از هدر Host شناسایی کند.

بیایید نگاهی دقیق‌تر به هر یک از این تکنیک‌ها بیندازیم.

Virtual hosting by URL path

شما می‌توانید با اختصاص دادن مسیرهای URL مختلف به سایتها مجازی روی یک سرور مشترک، از brute force استفاده کنید. به عنوان مثال، می‌توانید به هر وب سایت یک پیشوند مسیر خاص بدهید:

فروشگاه سخت افزار Joe می‌تواند <http://www.joes-hardware.com/joe/index.html> باشد.





فروشگاه عتیقه جات Mary می‌تواند <http://www.marys-antiques.com/mary/index.html> باشد.

هنگامی که درخواست‌ها به سرور می‌رسند، اطلاعات نام میزبان در درخواست وجود ندارد، اما سرور می‌تواند آن‌ها را بر اساس مسیر تشخیص دهد:

درخواست سخت افزار Joe به صورت GET /joe/index.html است.

درخواست عتیقه جات Mary به صورت GET /mary/index.html است.

این راه حل خوبی نیست. پیشوندهای "/joe" و "/mary" اضافی و گیج کننده هستند (ما قبلاً "joe" را در نام میزبان ذکر کردیم). بدتر از آن، قرارداد رایج تعیین <http://www.joes-hardware.com> یا برای صفحه اصلی کار نخواهد کرد.

به طور کلی، میزبانی مجازی مبتنی بر URL راه حل ضعیفی است و به ندرت استفاده می‌شود.

Virtual hosting by port number

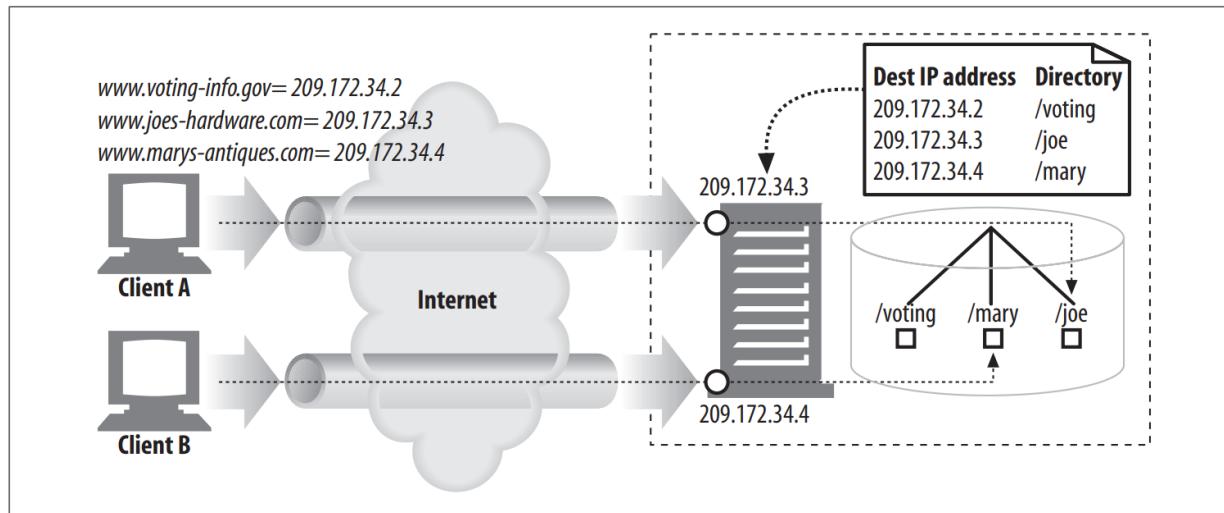
به جای تغییر نام مسیر، Joe و Mary می‌توانند به هر کدام یک شماره پورت متفاوت در وب سرور اختصاص دهند. به عنوان مثال، Joe می‌تواند به جای پورت 80، 82 و Mary می‌تواند 83 را داشته باشند. اما این راه حل مشکل مشابهی دارد: کاربر نهایی انتظار دارد منابع را بدون نیاز به تعیین یک پورت غیر استاندارد در URL پیدا کند.

Virtual hosting by IP address

یک رویکرد بسیار بهتر (در استفاده رایج) آدرس دهی IP مجازی است. در اینجا، هر وب سایت مجازی یک یا چند آدرس IP منحصر به فرد دریافت می‌کند. آدرس‌های IP تمامی وب‌سایتها مجازی به یک سرور مشترک متصل می‌شوند. سرور می‌تواند آدرس IP مقصد اتصال HTTP را جستجو کند و از آن برای تعیین اینکه مشتری فکر می‌کند به چه وب سایتی متصل است، استفاده کند.

فرض کنید یک میزبان آدرس IP 209.172.34.3 را به www.joes-hardware.com اختصاص داده است، آدرس IP 209.172.34.4 را به www.marys-antiques.com اختصاص داده است و هر دو آدرس IP را به یک دستگاه سرور فیزیکی متصل کرده است. سپس سرور وب می‌تواند از آدرس IP مقصد برای شناسایی سایت مجازی مورد درخواست استفاده کند، همانطور که در شکل زیر نشان داده شده است:





کلاینت A، Fetch را `http://www.joes-hardware.com/index.html` می‌کند.

کلاینت A آدرس IP را برای `www.joes-hardware.com` پیدا می‌کند و 209.172.34.3 را دریافت می‌کند.

کلاینت A یک اتصال TCP را به وب سرور مشترک در 209.172.34.3 باز می‌کند.

کلاینت A درخواست "GET /index.html HTTP/1.0" را ارسال می‌کند.

قبل از اینکه وب سرور پاسخی را ارائه دهد، آدرس IP مقصد واقعی (209.172.34.3) را یادداشت می‌کند، تعیین می‌کند که این یک آدرس IP مجازی برای وب سایت Joe است و درخواست از زیر شاخه `/joe` را برآورده می‌کند. در این حالت صفحه `/joe/index.html` برگردانده می‌شود.

به طور مشابه، اگر کلاینت B، `http://www.marys-antiques.com/index.html` را درخواست کند:

کلاینت B آدرس IP را برای `www.marys-antiques.com` پیدا می‌کند و 209.172.34.4 را دریافت می‌کند.

کلاینت B یک اتصال TCP به وب سرور در 209.172.34.4 باز می‌کند.

کلاینت B درخواست "GET /index.html HTTP/1.0" را ارسال می‌کند.

وب سرور تعیین می‌کند که 209.172.34.4 وب سایت Mary است و درخواست زیر شاخه `/mary` را انجام می‌دهد و سند `/mary/index.html` را برگردانده می‌شود.

میزبانی IP مجازی کار می‌کند، اما مشکلاتی را به خصوص برای هاستهای بزرگ ایجاد می‌کند:





- سیستم‌های رایانه‌ای معمولاً محدودیتی در تعداد آدرس‌های IP مجازی که می‌توانند به یک ماشین متصل شوند، دارند. میزبان‌هایی که می‌خواهند صدها یا هزاران سایت مجازی روی یک سرور مشترک میزبانی شوند، ممکن است خوش شانس نباشد.
- آدرس‌های IP یک کالای کمیاب هستند. میزبان‌هایی با سایت‌های مجازی زیاد ممکن است نتوانند آدرس IP مجازی کافی برای وب سایت‌های میزبانی شده به دست آورند.
- کمبود آدرس IP زمانی بدتر می‌شود که میزبان‌ها سرورهای خود را برای ظرفیت اضافی، Replicate می‌کنند. آدرس‌های IP مجازی متفاوتی ممکن است بر روی هر Replicated Server، بسته به معماری load-balancing مورد نیاز باشد، بنابراین تعداد آدرس‌های IP مورد نیاز می‌تواند در تعداد Replicated Server ها ضرب شود.

با وجود مشکلات مصرف آدرس در میزبانی IP مجازی، این روش به طور گستردگی مورد استفاده قرار می‌گیرد.

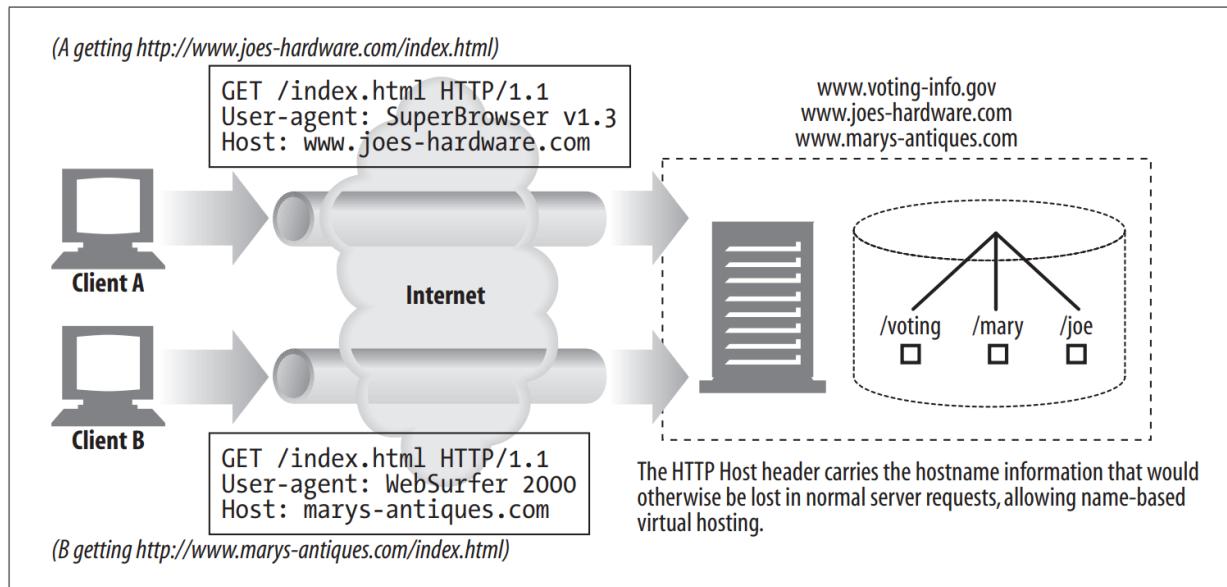
Virtual hosting by Host header

برای جلوگیری از مصرف بیش از حد آدرس و محدودیت‌های IP مجازی، می‌خواهیم آدرس IP یکسانی را بین سایت‌های مجازی به اشتراک بگذاریم، اما همچنان بتوانیم سایتها را از هم جدا کنیم. اما همانطور که دیدیم، چون اکثر مرورگرها فقط بخش مسیر URL را به سرورها ارسال می‌کنند، اطلاعات حیاتی نام میزبان مجازی از بین می‌رود.

برای حل این مشکل، پیاده‌سازهای مرورگر و سرور HTTP را گسترش دادند تا نام میزبان اصلی را به سرورها ارائه دهند. اما مرورگرها نمی‌توانند فقط یک URL کامل ارسال کنند، زیرا این کار باعث می‌شود بسیاری از سرورهایی که انتظار داشتند فقط یک بخش مسیر را دریافت کنند، خراب شود. در عوض، نام میزبان (و پورت) در یک هدر پسوند میزبان در همه درخواست‌ها ارسال می‌شود.

در شکل زیر، کلاینت A و سرویس گیرنده B هر دو هدرهای Host را ارسال می‌کنند که نام میزبان اصلی در حال دسترسی را دارند. هنگامی که سرور درخواست /index.html را دریافت می‌کند، می‌تواند از هدر Host برای تصمیم گیری از منابع استفاده کند.





هدرهای Host برای اولین بار با HTTP/1.0+ توسعه یافته توسط HTTP/1.0 معرفی شدند. هدرهای Host برای انطباق با HTTP/1.1 مورد نیاز است. هدر Host توسط اکثر مرورگرها و سرورهای مدرن پشتیبانی می‌شود، اما هنوز تعداد کمی از کلاینت‌ها و سرورها (و روبات‌ها) وجود دارند که از آن‌ها پشتیبانی نمی‌کنند.

HTTP/1.1 Host Headers

هدر Host یک هدر درخواست HTTP/1.1 است که در RFC 2068 تعریف شده است. سرورهای مجازی آنقدر رایج هستند که اکثر سرویس گیرندگان HTTP، حتی اگر مطابق با HTTP/1.1 نباشند، هدر Host را پیاده سازی می‌کنند.

Syntax and usage

هدر Host، میزبان اینترنت و شماره پورت منبع درخواست را مشخص می‌کند، همانطور که از URL اصلی بدست آمده است:

Host = "Host" ":" host [":" port]

به خصوص:

- اگر هدر Host حاوی پورت نباشد، پورت پیشفرض برای scheme در نظر گرفته می‌شود.
- اگر URL حاوی یک آدرس IP باشد، هدر Host باید حاوی همان آدرس باشد.
- اگر URL حاوی نام میزبان باشد، هدر Host باید دارای همان نام باشد.





- اگر URL حاوی یک نام میزبان باشد، هدر Host نباید حاوی آدرس IP معادل نام میزبان URL باشد، زیرا این امر منجر به Break شدن سرورهای میزبان مجازی که چندین سایت مجازی را روی یک آدرس IP واحد لایه لایه می‌کنند، می‌شود.
- اگر URL حاوی یک نام میزبان باشد، هدر Host نباید حاوی نام مستعار دیگری برای این نام میزبان باشد، زیرا این کار سرورهای میزبان مجازی را نیز خراب می‌کند.
- اگر کلاینت از یک سرور Explicit Proxy استفاده می‌کند، کلاینت باید نام و پورت سرور مبدا را در هدر Host قرار دهد، نه سرور پراکسی. در گذشته، چندین کلاینت وب دارای اشکالاتی بودند که در آن هدر Host خروجی روی نام میزبان پروکسی تنظیم می‌شد. این موضوع به زمانی مربوط می‌شد که تنظیمات پروکسی کلاینت فعال بود. این رفتار نادرست باعث می‌شود که پراکسی‌ها و سرورهای مبدا رفتار مناسبی از خود نشان ندهند.
- کلاینت‌های وب باید یک فیلد هدر Host را در همه پیام‌های درخواست، قرار دهند.
- پروکسی‌های وب باید هدرهای Host را برای درخواست پیام‌ها قبل از ارسال آن‌ها اضافه کنند.
- سرورهای وب HTTP/1.1 باید با یک کد وضعیت 400 به هر پیام درخواستی HTTP/1.1 که فاقد فیلد هدر Host است پاسخ دهند.

در اینجا یک نمونه پیام درخواست HTTP استفاده شده برای واکشی صفحه اصلی به همراه فیلد هدر Host مورد نیاز است:

```
GET http://www.joes-hardware.com/index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */
Accept-Encoding: gzip
Accept-Language: en
Host: www.joes-hardware.com
```

Missing Host headers

درصد کمی از مرورگرهای قدیمی در حال استفاده، هدر Host را ارسال نمی‌کنند. اگر یک سرور میزبان مجازی از هدرهای Host برای تعیین اینکه کدام وب سایت را ارائه می‌دهد استفاده می‌کند و هیچ هدر Host ای وجود ندارد، احتمالاً کاربر به یک صفحه وب پیش فرض (مانند صفحه وب ISP) هدایت می‌شود یا یک خطای گردانده و یا صفحه‌ای که به کاربر پیشنهاد می‌کند مرورگر خود را ارتقا دهد، نمایش داده می‌شود.





Interpreting Host headers

یک سرور اصلی که به صورت مجازی میزبانی نمی‌شود و اجازه نمی‌دهد منابع بر اساس میزبان درخواستی متفاوت باشد، ممکن است مقدار فیلد هدر Host را نادیده بگیرد. اما هر سرور مبدا که منابع را بر اساس میزبان متمایز می‌کند باید از قوانین زیر برای تعیین منبع درخواستی در یک درخواست HTTP/1.1 استفاده کند:

- اگر URL در پیام درخواست HTTP مطلق باشد (به عنوان مثال، شامل یک طرح و جزء میزبان باشد)، مقدار موجود در هدر Host به نفع URL نادیده گرفته می‌شود.
- اگر URL موجود در پیام درخواست HTTP میزبانی نداشته باشد و درخواست حاوی هدر Host باشد، مقدار میزبان/پورت از هدر Host به دست می‌آید.
- اگر هیچ میزبان معتبری را نتوان از طریق مراحل 1 یا 2 تعیین کرد، یک پاسخ Bad Response 400 به کلاینت برگردانده می‌شود.

Host headers and proxies

برخی از نسخه‌های مرورگر، هدرهای میزبان Host را ارسال می‌کنند، به خصوص زمانی که برای استفاده از پراکسی پیکربندی شده باشند. به عنوان مثال، هنگامی که مرورگر برای استفاده از یک پروکسی پیکربندی شده است، برخی از نسخه‌های قدیمی‌تر کلاینت‌های PointCast و Apple به اشتباه نام پروکسی را به جای سرور مبدا در هدر Host ارسال می‌کنند.

Making Web Sites Reliable

چندین مورد وجود دارد که در طی آن وب سایتها معمولاً Break می‌شوند:

- سرور Downtime
- افزایش ترافیک: ناگهان همه می‌خواهند یک پخش خبری خاص را ببینند یا برای فروش عجله کنند. جهش‌های ناگهانی می‌تواند منجر به افزایش بار سرور وب شود و سرعت آن را کند یا به طور کامل متوقف کند.
- قطع یا از بین رفتن شبکه (Network outages or losses)

در این بخش راههایی برای پیش بینی و مقابله با این مشکلات رایج ارائه می‌شود.



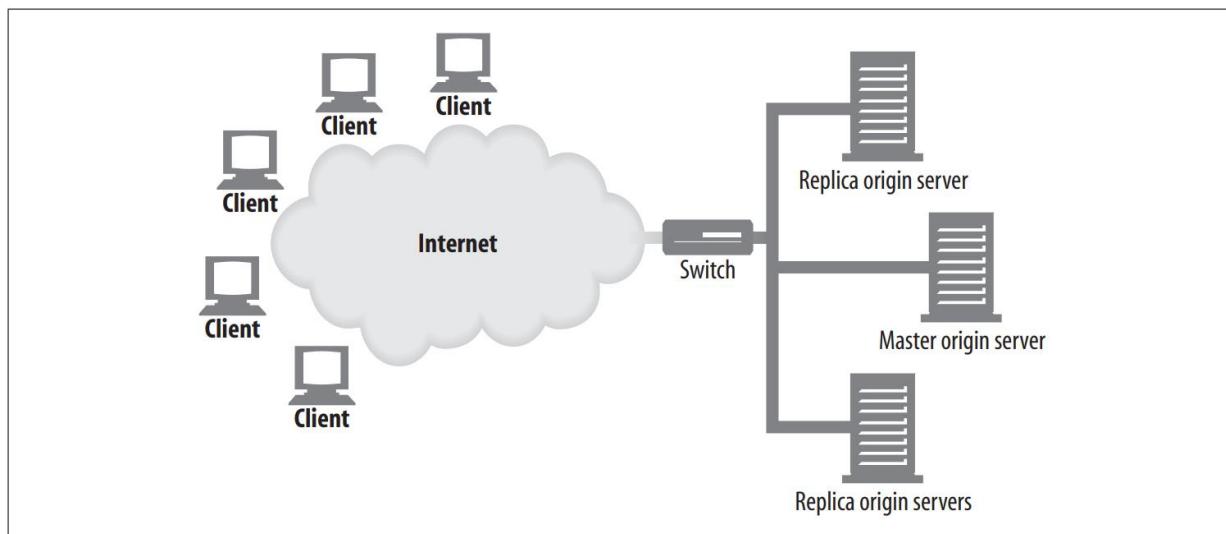


Mirrored Server Farms

Server Farm، بانکی از سرورهای وب با پیکربندی یکسان است که می‌توانند یکدیگر را پوشش دهند. محتوای موجود در هر سرور در Farm را می‌توان انعکاس داد، به طوری که اگر یکی مشکل داشت، دیگری می‌تواند جای آن را پر کند.

اغلب، میرورهای Mirrored Server ها از یک رابطه سلسله مراتبی پیروی می‌کنند. یک سرور ممکن است به عنوان مرجع محتوا^{۳۸} عمل کند - سروری که حاوی محتوای اصلی است (شاید سروری که نویسنده‌گان محتوا در آن پست ارسال می‌کنند). این سرور، Master Origin Server نامیده می‌شود. Mirrored Server ها که محتوا را از Master Origin Server دریافت می‌کنند، Replica Origin Server نامیده می‌شوند. یک راه ساده برای استقرار یک Server Farm استفاده از سوئیچ شبکه برای توزیع درخواست‌ها به سرورها است. آدرس IP برای هر یک از وب سایتها میزبانی شده روی سرورها آدرس IP سوئیچ است.

در شکل زیر نشان داده شده است، Master Origin Server که در شکل زیر نشان داده شده است، مسئول ارسال محتوا به Replica Origin Server است. برای دنیای خارج، آدرس IP برای این محتوا، آدرس IP سوئیچ است. سوئیچ وظیفه ارسال درخواست به سرورها را بر عهده دارد.



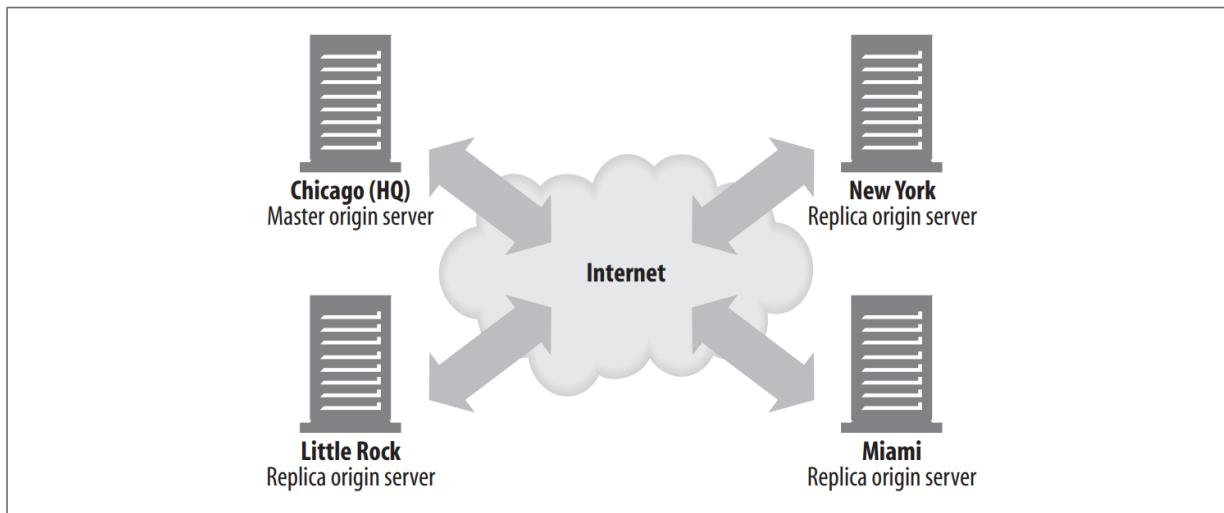
میرورهای Mirrored Web Server ها می‌توانند حاوی کپی‌هایی از محتوای یکسان در مکان‌های مختلف باشند. شکل زیر چهار Mirrored Server را نشان می‌دهد که یک سرور اصلی در شیکاگو و نسخه‌های مشابه آن در نیویورک،

³⁸ content authority





میامی و لیتل راک وجود دارد. سرور اصلی به کلاینت‌ها در منطقه شیکاگو خدمات می‌دهد و همچنین وظیفه انتشار محتوای خود را به سرورهای مشابه دارد.



در سناریوی شکل بالا، چند راه وجود دارد که درخواست‌های کلاینت به یک سرور خاص هدایت می‌شود:

HTTP Redirection

URL مربوط به محتوا می‌تواند به آدرس IP سرور اصلی Resolve شود، که سپس می‌تواند URL‌ها را به سرورهای مشابه ارسال کند.

DNS Redirection

برای محتوا می‌تواند به چهار آدرس IP، DNS Resolve شود و سرور DNS می‌تواند آدرس IP را که برای کلاینت‌ها ارسال می‌کند انتخاب کند.

Content Distribution Networks

شبکه توزیع محتوا (CDN) به سادگی شبکه‌ای است که هدف آن توزیع محتوای خاص است. Node‌های شبکه می‌توانند وب سرورها، جانشین‌ها^{۳۹} یا Cache‌های پراکسی باشند.

Surrogate Caches in CDNs

Surrogate Cache را می‌توان به جای سرورهای مبداء مشابه در شکل‌های پیشین استفاده کرد. ها که به عنوان Reverse Proxy نیز شناخته می‌شوند، درخواست‌های سرور برای محتوا را درست مانند Mirrored Web Server ها دریافت می‌کنند. آن‌ها درخواست‌های سرور را از طرف مجموعه

³⁹ surrogates





خاصی از سرورهای مبدأ دریافت می‌کنند (این به دلیل نحوه Advertise آدرس‌های IP برای محتوا امکان پذیر است؛ معمولاً یک رابطه کاری بین Origin Server و Surrogate وجود دارد و Surrogate ها انتظار دارند درخواست‌هایی را با هدف Origin Server های خاص دریافت کنند).

تفاوت بین سرور Mirrored و Surrogate در این است که Surrogate ها معمولاً مبتنی بر تقاضا هستند. آن‌ها کپی کاملی از محتوای Origin Server را ذخیره نمی‌کنند. آن‌ها هر محتوایی را که کلاینت‌هایشان درخواست کنند ذخیره می‌کنند. نحوه توزیع محتوا در Cache آن‌ها بستگی به درخواست‌هایی دارد که دریافت می‌کنند. Origin Server مسئولیتی برای بروزرسانی محتوا خود ندارد. برای دسترسی آسان به محتوای «گرم» (محتوای که تقاضای بالایی دارد)، برخی Surrogate ها دارای ویژگی‌های Prefetching هستند که به آن‌ها امکان می‌دهد مطالب را قبل از درخواست کاربر جذب کنند.

یک پیچیدگی اضافه در CDN با Surrogate ها، امکان سلسله مراتب Cache است.

Proxy Caches in CDNs

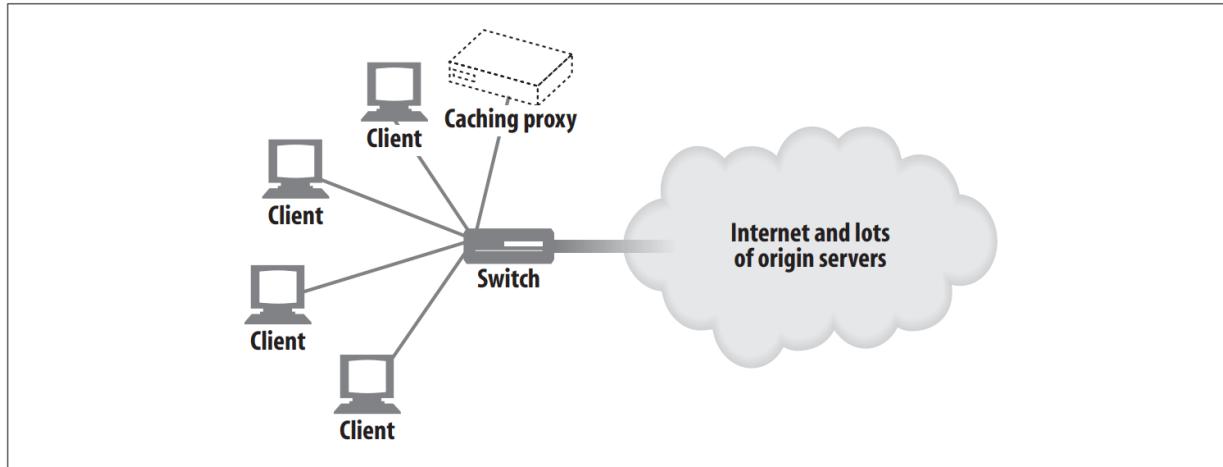
Proxy Cache ها نیز می‌توانند در پیکربندی‌های مشابه با شکل های پیشین مستقر شوند. برخلاف Surrogate ها، Proxy Cache های سنتی می‌توانند درخواست‌هایی را با هدف هر سرور وب دریافت کنند (نیازی نیست که هیچ رابطه کاری یا توافق نامه آدرس IP بین یک Proxy Cache و یک Origin Server وجود داشته باشد). با این حال، مانند Surrogate ها، محتوای Proxy Cache معمولاً مبتنی بر تقاضا است و انتظار نمی‌رود که دقیقاً تکراری از محتوای Origin Server باشد. برخی از Proxy Cache ها نیز می‌توانند با محتوای داغ^{۴۰} از قبل بارگذاری شوند.

Proxy Cache های مبتنی بر تقاضا می‌توانند در انواع دیگری از پیکربندی‌ها مستقر شوند - به ویژه پیکربندی‌های رهگیری^{۴۱}، که در آن یک دستگاه لایه-2 یا -3 (سوئیچ یا روتر) ترافیک وب را رهگیری می‌کند و آن را به یک Proxy Cache می‌فرستد (شکل زیر را ببینید).

⁴⁰ hot content

⁴¹ interception configurations





یک پیکربندی رهگیری به توانایی راهاندازی شبکه بین کلاینت‌ها و سرورها بستگی دارد تا تمام درخواست‌های HTTP مناسب به صورت فیزیکی به Cache هدایت شوند. (به فصل 20 مراجعه کنید). محتوا با توجه به درخواست‌هایی که دریافت می‌کند در Cache توزیع می‌شود.

Making Web Sites Fast

بسیاری از فناوری‌های ذکر شده در بخش قبل نیز به بارگذاری سریع‌تر وبسایت‌ها کمک می‌کنند. Server و Proxy Cache Farm های توزیع شده یا سرورهای Surrogate، ترافیک شبکه را توزیع نموده و از ازدحام جلوگیری می‌کنند. توزیع محتوا آن را به کاربران نهایی نزدیکتر می‌کند، به طوری که زمان سفر از سرور به کلاینت کمتر است. کلید سرعت دسترسی به منابع این است که چگونه درخواست‌ها و پاسخ‌ها از کلاینت به سرور و در سراسر اینترنت هدایت می‌شوند. برای جزئیات بیشتر در مورد روش‌های تغییر مسیر به فصل 20 مراجعه کنید.

روش دیگر برای افزایش سرعت وب سایتها، Encoding محتوا برای حمل و نقل سریع است. این می‌تواند برای مثال به معنای فشرده سازی محتوا باشد، با این فرض که کلاینت گیرنده می‌تواند آن را از حالت فشرده خارج کند. برای جزئیات بیشتر به فصل 15 مراجعه کنید.

For More Information

<http://www.ietf.org/rfc/rfc3040.txt>

<http://search.ietf.org/internet-drafts/draft-ietf-cdi-request-routing-reqs-00.txt>

Apache: The Definitive Guide Ben Laurie and Peter Laurie ,





فصل نوزدهم – Publishing Systems

چگونه می‌توان صفحات وب را ایجاد کرد و آن‌ها را به سرور وب منتقل کرد؟

در عصر تاریک وب (مثلاً در سال ۱۹۹۵)، ممکن است HTML خود را به صورت دستی در یک ویرایشگر متنی ساخته باشید و به صورت دستی محتوا را با استفاده از FTP در سرور وب آپلود کرده باشید. این روش در دنیاک بود، همانگی با همکاران در این روش دشوار بود و به ویژه از امنیت لازم نیز برخوردار نبود.

ابزارهای انتشار مدرن، ایجاد، انتشار و مدیریت محتوای وب را بسیار راحت‌تر می‌کنند. امروزه، می‌توانید به صورت تعاملی محتوای وب را همانطور که روی صفحه می‌بینید ویرایش کنید و با یک کلیک آن محتوا را در سرورها منتشر نمایید، همچنین از هر فایلی که تغییر کرده است نیز مطلع می‌شوید.

بسیاری از ابزارهایی که از انتشار محتوا از راه دور پشتیبانی می‌کنند از پسوندهای^{۴۲} پروتکل HTTP استفاده می‌کنند. در این فصل، دو فناوری مهم برای انتشار محتوای وب مبتنی بر HTTP را توضیح می‌دهیم: DAV و

FrontPage Server Extensions for Publishing Support

(که معمولاً به عنوان FP نامیده می‌شود) یک ابزار همه کاره برای نوشتن و انتشار وب است که توسط شرکت مایکروسافت ارائه شده است. ایده اصلی برای FrontPage (FrontPage 1.0) در سال ۱۹۹۴ در Vermeer Technologies Inc. مطرح شده و به عنوان اولین محصول ترکیبی مدیریت وب سایت و ایجاد یک ابزار واحد و یکپارچه شناخته شد. مایکروسافت Vermeer را خریداری کرد و FrontPage 1.1 را در سال ۱۹۹۶ منتشر نمود.

FrontPage Server Extensions

مایکروسافت به عنوان بخشی از استراتژی publish anywhere مجموعه‌ای از نرمافزار سمت سرور به نام FPSE^{۴۳} یا FrontPage Server Extensions را منتشر کرد. این Extension‌های سمت سرور با وب سرور یکپارچه می‌شوند و ترجمه لازم را بین وب سایت و کلاینت در حال اجرا FrontPage (و سایر کلاینت‌هایی که از این برنامه‌های افزودنی پشتیبانی می‌کنند) ارائه می‌دهند.

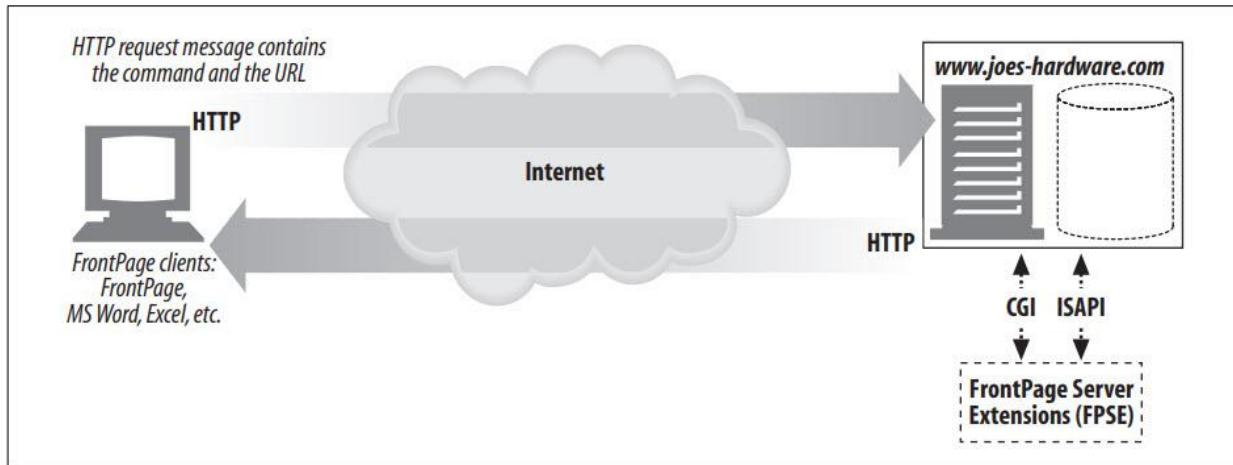
علاقه اصلی ما در پروتکل انتشار بین کلاینت‌های FP و FPSE نهفته است. این پروتکل نمونه‌ای از طراحی برنامه‌های افزودنی برای سرویس‌های اصلی موجود در HTTP بدون تغییر معنای HTTP ارائه می‌دهد.

⁴² extensions





پروتکل انتشار FrontPage یک لایه RPC را در بالای درخواست HTTP POST پیاده سازی می کند. این به کلاینت FrontPage اجازه می دهد تا دستوراتی را برای بروزرسانی اسناد در وبسایت، انجام جستجوها، همکاری بین نویسندهای وب و غیره به سرور ارسال کند. شکل زیر یک نمای کلی از ارتباطات را نشان می دهد.



وب سرور درخواست های POST خطاب به FPSE را می بیند (که به عنوان مجموعه ای از برنامه های CGI، در مورد سرور غیر مایکروسافت IIS اجرا می شود) و بر این اساس آن درخواست ها را هدایت می کند. تا زمانی که فایروال های مداخله ای و سرور های پراکسی برای اجازه دادن به روش POST پیکربندی شده باشند، FrontPage می تواند به ارتباط با سرور ادامه دهد.

FrontPage Vocabulary

قبل از اینکه عمیق تر به لایه RPC تعریف شده توسط FPSE برویم، ممکن است ایجاد واژگان مشترک در این بخش برای ما بسیار کمک کننده باشد:

Virtual server

یکی از چندین وب سایت در حال اجرا بر روی یک سرور، که هر کدام یک نام دامنه و آدرس IP منحصر به فرد دارند. در اصل، یک سرور مجازی به یک وب سرور اجازه می دهد تا چندین وب سایت را میزبانی کند، که هر یک از آنها به نظر یک مرورگر توسط وب سرور خود میزبانی می شود. وب سروری که از سرورهای مجازی پشتیبانی می کند، Multi-Hosting Web Server نامیده می شود. ماشینی که با چندین آدرس IP پیکربندی شده است، Multi-Homed Server نامیده می شود (برای جزئیات بیشتر، لطفاً به Virtual Hosting در فصل ۱۸ مراجعه کنید).





Root web

Top-Level Content Directory یک وب سرور، یا در یک محیط چند میزبانی، یک Web Root Content Directory یک وب سرور مجازی. برای دسترسی به URL کافی است سرور را بدون تعیین نام صفحه مشخص کنید. در هر وب سرور فقط یک Web Root می‌تواند وجود داشته باشد.

Subweb

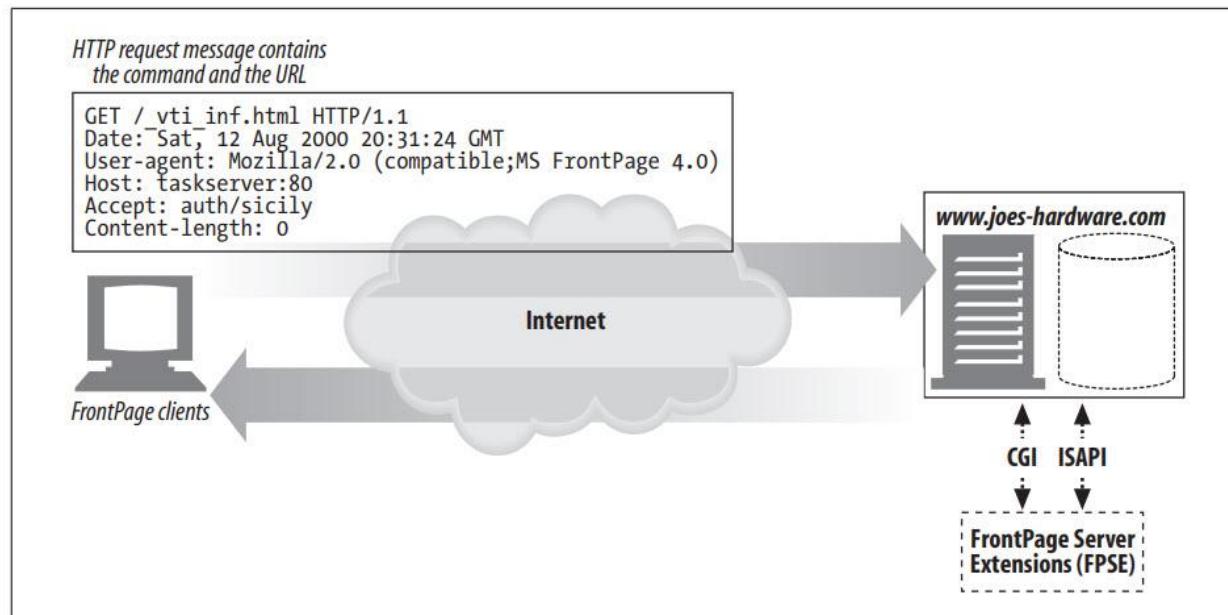
یک Subdirectory نامگذاری شده از Web Root یا یک Subweb یا یک گسترده FPSE کامل است. یک Subweb می‌تواند یک موجودیت مستقل کامل با قابلیت تعیین مجوزهای مدیریت و تألیف خود باشد. علاوه بر این، Subweb ممکن است محدودهای را برای روش‌هایی مانند جستجو فراهم کنند.

The FrontPage RPC Protocol

کلاینت FrontPage و FPSE با استفاده از پروتکل RPC اختصاصی با هم ارتباط برقرار می‌کنند. این پروتکل با تعییه متدهای RPC و متغیرهای مرتبط با آن‌ها در بدن درخواست POST در بالای HTTP POST لایه بندی شده است.

برای شروع فرآیند، کلاینت باید مکان و نام برنامه‌های هدف را روی سرور تعیین کند (بخشی از بسته FPSE که می‌تواند درخواست POST را اجرا کند). سپس یک درخواست GET ویژه صادر می‌کند (شکل زیر را ببینید).





هنگامی که فایل برگردانده می‌شود، کلاینت FrontPage، پاسخ را می‌خواند و مقادیر مرتبط با FPAadminScriptUrl و FPAuthorScriptUrl را پیدا می‌کند. به طور معمول، این موارد ممکن است به شکل زیر باشند:

```
FPShtmlScriptUrl=_vti_bin/_vti_rpc/shtml.dll
FPAuthorScriptUrl=_vti_bin/_vti_aut/author.dll
FPAadminScriptUrl=_vti_bin/_vti_adm/admin.dll
```

FPShtmlScriptUrl به کلاینت می‌گوید که درخواست‌های دستورات Browse Time (مثلاً دریافت نسخه FPAuthorScriptUrl) را کجا پست کند تا اجرا شود.

Authoring FPAuthorScriptUrl به کلاینت می‌گوید که کجا باید درخواست‌ها را برای اجرای دستورات Time پست کند. به طور مشابه، FPAadminScriptUrl به کلاینت FPAadminScriptUrl می‌گوید که در کجا درخواست‌ها را برای اقدامات مدیریتی ارسال کند.

اکنون که می‌دانیم برنامه‌های مختلف در کجا قرار دارند، آماده ارسال درخواست هستیم.

Request

بدن درخواست POST حاوی دستور RPC، به شکل "`method=<command>`" و هر پارامتر مورد نیاز است. به عنوان مثال، پیام RPC را که لیستی از اسناد را درخواست می‌کند، به صورت زیر در نظر بگیرید:





```
POST /_vti_bin/_vti_auuthor.dll HTTP/1.1  
Date: Sat, 12 Aug 2000 20:32:54 GMT  
User-Agent: MSFrontPage/4.0
```

```
.....  
<BODY>  
method=list+documents%3a4%2e0%2e2%2e3717&service%5fname=&listHiddenDocs=false&listExplorerDocs=false&listRecurse=false&listFiles=true&listFolders=true&listLinkInfo=true&listIncludeParent=true&listDerived=false  
&listBorders=false&listChildWebs=true&initialUrl=&folderList=%5b%3bTW%7c12+Aug+2000+2  
0%3a33%3a04+%2d0000%5d
```

بدنی دستور POST حاوی دستور RPC است که به FPSE ارسال می‌شود. همانند برنامه‌های CGI، فضاهای متدهای مخصوص کاراکترهای علامت مثبت (+) کدگذاری می‌شوند. تمام کاراکترهای غیرalfabiyi دیگر در متدهای استفاده از فرمات %XX کدگذاری می‌شوند، جایی که XX مخفف نمایش ASCII کاراکتر است. با استفاده از این نماد، یک نسخه قابل خواندن‌تر از بدنی مانند زیر خواهد بود:

```
method=list+documents:4.0.1.3717  
&service_name=  
&listHiddenDocs=false  
&listExplorerDocs=false  
.....
```

برخی از عناصر ذکر شده عبارتند از:

service_name

آدرس وب سایتی که متدهای روی آن عمل کند. باید یک پوشه موجود یا یک سطح زیر یک پوشه موجود باشد.

listHiddenDocs

اگر مقدار آن true باشد، استناد پنهان را در وب نشان می‌دهد. استناد URL هایی با اجزای مسیر که با "_" شروع می‌شوند، مشخص می‌شوند.

listExploreDocs

اگر مقدار "true" باشد، لیست وظایف را فهرست می‌کند.





Response

اکثر متدهای پروتکل RPC دارای مقادیر بازگشته هستند. رایج‌ترین مقادیر بازگشته مربوط به متدها و خطاهای موفق هستند. برخی از متدها دارای بخش سوم نیز هستند، «Sample Return Code» درستی کدها را تفسیر می‌کند تا بازخورد دقیقی به کاربر ارائه دهد.

در ادامه مثال ما، FPSE درخواست "list+documents" را پردازش می‌کند و اطلاعات لازم را برمی‌گرداند. نمونه پاسخ به شرح زیر است:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 12 Aug 2000 22:49:50 GMT
Content-type: application/x-vermeer-rpc
X-FrontPage-User-Name: IUSER_MINSTAR
```

```
<html><head><title>RPC packet</title></head>
<body>
<p>method=list documents: 4.0.2.3717
<p>document_list=
<ul>
  <li>document_name=help.gif
<\ul>
```

همانطور که از پاسخ می‌بینید، یک لیست قالب بندی شده از اسناد موجود در وب سرور به کلاینت FP بازگردانده می‌شود. شما می‌توانید لیست کامل دستورات و پاسخ‌ها را در وب سایت مایکروسافت پیدا کنید.

FrontPage Security Model

هر سیستم Publishing که مستقیماً به محتوای سرور وب دسترسی دارد باید از پیامدهای امنیتی اقدامات خود بسیار آگاه باشد. در بیشتر موارد، FPSE برای تامین امنیت به سرور وب بستگی دارد.

مدل امنیتی FPSE سه نوع کاربر را تعریف می‌کند: مدیران، نویسندها و مرورگرها، که مدیران کنترل کامل دارند. همه مجوزها تجمعی هستند. یعنی همه مدیران می‌توانند وب FrontPage را بنویسند و مرور کنند. به طور مشابه، همه نویسندها مجاز دارای مجوزهای مرور هستند.

لیست مدیران، نویسندها و مرورگرها برای یک وب توسعه یافته FPSE مشخص شده است. همه زیر وب‌ها ممکن است مجوزها را از Web Root به ارث ببرند یا مجوزهای خود را تنظیم کنند. برای وب





سرورهای غیر IIS، همه برنامه‌های FPSE باید در دایرکتوری‌هایی با علامت Executable ذخیره شوند (همان محدودیت برای هر برنامه دیگر CGI). ابزار مدیریت سرور FrontPage، Fpsrvadm، ممکن است برای این منظور استفاده شود. در سرورهای IIS، مدل امنیتی یکپارچه ویندوز غالب است.

در سرورهای غیر IIS، مکانیسم‌های کنترل دسترسی به سرور وب، کاربرانی را که مجاز به دسترسی به یک برنامه خاص هستند، مشخص می‌کنند. در سرورهای وب NCSA و Apache، فایل htaccess وجود دارد. در سرورهای Netscape nsconfig، نامگذاری شده است. فایل دسترسی، کاربران، گروه‌ها و آدرس‌های IP را با سطوح مختلفی از مجوزها مرتبط می‌کند: GET (خواندن)، POST (اجرا)، و غیره. برای مثال، برای اینکه یک کاربر نویسنده در یک وب سرور آپاچی باشد، در فایل htaccess باید به آن کاربر اجازه ارسال به author.exe را بدهید. این فایل‌های مشخصات دسترسی اغلب بر اساس هر دایرکتوری تعریف می‌شوند و انعطاف‌پذیری بیشتری در تعریف مجوزها فراهم می‌کنند.

در سرورهای IIS، مجوزها در برابر ACL‌ها برای یک SubRoot یا Root مشخص بررسی می‌شوند. هنگامی که IIS درخواستی دریافت می‌کند، ابتدا وارد سیستم می‌شود و هویت کاربر را جعل^{۴۳} می‌کند، سپس درخواست را به یکی از سه کتابخانه DLL (Dynamic Link Libraries) ارسال می‌کند. DLL اعتبار جعل هویت را در برابر ACL تعریف شده برای پوشش مقصود بررسی می‌کند. در صورت موفقیت آمیز بودن بررسی، عملیات درخواستی توسط پسوند DLL اجرا می‌شود. در غیر این صورت، پیام Permission Denied به کلاینت ارسال می‌شود. با توجه به ادغام شدید امنیت ویندوز با IIS، ممکن است از User Manager برای تعریف کنترل دقیق استفاده شود.

علیرغم این مدل امنیتی پیچیده، فعال کردن FPSE به عنوان یک Security Risk بی‌همیت شهرت یافته است. در بیشتر موارد، این به دلیل شیوه‌های نامناسب اتخاذ شده توسط مدیران وب سایت است. با این حال، نسخه‌های قبلی FPSE دارای حفره‌های امنیتی شدیدی بودند. این مشکل همچنان با اعمال محترمانگی مورد نیاز برای اجرای کامل یک مدل امنیتی سخت تشدید شد.

WebDAV and Collaborative Authoring

WebDAV یا Web Distributed Authoring and Versioning یک بعد اضافی به انتشار وب می‌افزاید. در حال حاضر، رایج‌ترین روش همکاری^{۴۴} کاملاً با فناوری پایین است: عمدتاً ایمیل، گاهی اوقات با اشتراک‌گذاری فایل‌های توزیع شده ترکیب می‌شود. ثابت شده است که این عمل بسیار ناخوشایند و مستعد خطا است،

⁴³ impersonate

⁴⁴ collaboration





که این کار با کنترل کمی یا بدون کنترل بر فرآیند انجام می‌شود. نمونه‌ای از راه اندازی یک وب سایت چند ملیتی و چند زبانه برای یک خودروساز را در نظر بگیرید. به راحتی می‌توان نیاز به یک سیستم قوی با موارد اولیه انتشار ایمن و قابل اعتماد، همراه با اصول اولیه همکاری مانند قفل کردن^{۴۵} و نسخه سازی^{۴۶} را مشاهده کرد.

WebDAV (منتشر شده به عنوان RFC 2518) بر گسترش HTTP برای ارائه یک پلتفرم مناسب برای نویسنده‌گی مشترک متمرکز است. در حال حاضر این یک تلاش IETF با پشتیبانی از فروشنده‌گان مختلف از جمله Adobe، Xerox، Oracle، Novell، Netscape، Microsoft، IBM، Apple و Apple است.

WebDAV Methods

مجموعه‌ای از متدهای HTTP جدید را تعریف می‌کند و محدوده عملیاتی چند متد HTTP دیگر را تغییر می‌دهد. متدهای جدید اضافه شده توسط WebDAV عبارتند از:

PROPFIND

ویژگی‌های یک منبع را بازیابی می‌کند.

PROPPATCH

یک یا چند ویژگی را روی یک یا چند منبع تنظیم می‌کند.

MKCOL

مجموعه‌ها را ایجاد می‌کند.

COPY

یک منبع یا مجموعه‌ای از منابع را از یک منبع معین به یک مقصد معین کپی می‌کند. مقصد لازم نیست روی یک دستگاه باشد.

MOVE

یک منبع یا مجموعه‌ای از منابع را از یک منبع معین به یک مقصد معین منتقل می‌کند. مقصد لازم نیست روی یک دستگاه باشد.

⁴⁵ locking
⁴⁶ versioning





LOCK

یک منبع یا چندین منبع را قفل می‌کند.

UNLOCK

یک منبع قفل شده قبلی را باز می‌کند.

متدهای HTTP اصلاح شده توسط WebDAV عبارتند از OPTIONS، PUT، DELETE. هر دو متدهای جدید و اصلاح شده در ادامه این فصل مورد بحث قرار خواهند گرفت.

WebDAV and XML

متدهای WebDAV عموماً نیازمند اطلاعات زیادی هستند که هم با درخواستها و هم با پاسخها مرتبط باشند. HTTP معمولاً این اطلاعات را در هدرهای پیام ارسال می‌کند. با این حال، انتقال اطلاعات لازم در هدرها به تنها یک محدودیت‌هایی را تحمیل می‌کند، از جمله مشکلات اعمال انتخابی اطلاعات هدر به منابع متعدد در یک درخواست، برای نشان دادن سلسله مراتب وغیره.

برای حل این مشکل، XML از WebDAV استفاده می‌کند، یک زبان فرانشانه‌گذاری که قالبی را برای توصیف داده‌های ساخت‌یافته فراهم می‌کند. WebDAV، XML را با موارد زیر فراهم می‌کند:

- متدهای برای قالب‌بندی دستورالعمل‌هایی که نحوه پردازش داده‌ها را توصیف می‌کند.
- متدهای برای قالب‌بندی پاسخ‌های پیچیده از سرور
- متدهای برای برقراری ارتباط اطلاعات سفارشی‌شده درباره مجموعه‌ها و منابع مورد استفاده
- وسیله‌ای انعطاف‌پذیر برای خود داده‌ها
- یک راه حل قوی برای اکثر افراد از مسائل بین‌المللی سازی

به طور سنتی، تعریف Schema برای اسناد XML در یک Document Type Definition (DTD) نگهداری شده که در خود سند XML ارجاع داده می‌شود. بنابراین، هنگام تلاش برای تفسیر یک سند XML، موجودیت تعريف DOCTYPE نام فایل DTD مرتبط با سند XML مورد نظر را می‌دهد.

WebDAV یک فضای نام واضح "DAV:" XML را تعریف می‌کند. بدون پرداختن به جزئیات زیاد، فضای نام XML مجموعه‌ای از نام عناصر یا ویژگی‌ها است. فضای نام، نام‌های تعییه‌شده را به‌طور منحصر‌به‌فرد در سراسر دامنه واجد شرایط می‌کند، بنابراین از هرگونه تصادم نام جلوگیری خواهد شد.

⁴⁷ namespace





کامل XML در مشخصات RFC 2518 WebDAV تعریف شده است. وجود یک schema از پیش تعریف شده به نرم افزار تجزیه^{۴۸} اجازه می‌دهد تا بدون نیاز به خواندن، در فایل‌های DTD فرضیاتی را در مورد ایجاد کند و آن‌ها را به درستی تفسیر نماید.

WebDAV Headers

WebDAV چندین هدر HTTP را برای تقویت عملکرد متدهای جدید معرفی می‌کند. این بخش یک نمای کلی ارائه می‌دهد. برای اطلاعات بیشتر پیشنهاد می‌شود تا به RFC 2518 مراجعه کنید. سرفصل‌های جدید عبارتند از:

DAV

برای برقراری ارتباط با قابلیت‌های WebDAV سرور استفاده می‌شود. تمام منابع پشتیبانی شده توسط WebDAV برای برگرداندن این هدر در پاسخ به درخواست OPTIONS مورد نیاز هستند. برای جزئیات بیشتر به "متدهای OPTIONS" مراجعه کنید.

DAV = "DAV" ":" "1" ["," "2"] ["," 1#extend]

Depth

عنصر حیاتی برای گسترش WebDAV به منابع گروه‌بندی شده با سطوح سلسله مراتبی چندگانه (برای «Collections and Namespace Management»، لطفاً به «Collections and Namespace Management» مراجعه کنید).

Depth = "Depth" ":" ("0" | "1" | "infinity")

بیایید به یک مثال ساده نگاه کنیم. یک دایرکتوری DIR_A با فایل‌های file_1.html و file_2.html استفاده کند، متدهای Depth: 0 و Depth: 1 اعمال می‌شود و برای پوشش DIR_A و فایل‌های آن، file_1.html و file_2.html اعمال می‌شود.

هدر Depth بسیاری از متدهای تعریف شده توسط WebDAV را تغییر می‌دهد. برخی از متدهایی که از هدر Depth استفاده می‌کنند عبارتند از LOCK، COPY و MOVE.

Destination

برای کمک به متدهای COPY یا MOVE در شناسایی URI مقصد تعریف شده است.

⁴⁸ parsing software





Destination = "Destination" ":" absoluteURI

If

تنها state token تعريف شده يك lock token است (به "The LOCK Method" مراجعه کنيد). هدر If مجموعه‌اي از شرط‌ها را تعريف مي‌کند. اگر همه آن‌ها نادرست ارزیابی شوند، درخواست با شکست مواجه خواهد شد. متدهایی مانند COPY و PUT با تعیین پیش شرط‌ها در هدر If، کاربرد پذیری را مشروط می‌کنند. در عمل، متداول‌ترین پیش شرط برای رضایت، دستیابی قبلی به lock است.

```
If = "If" ":" (1*No-tag-list | 1*Tagged-list)
  No-tag-list = List
  Tagged-list = Resource 1>List
  Resource = Coded-URL
  List = "(" 1*([ "Not"])(State-token | [ " entity-tag "])) ")"
  State-token = Coded-URL
  Coded-URL = "<" absoluteURI ">"
```

Lock-Token

متد UNLOCK برای تعیین قفلی که باید برداشته شود استفاده می‌شود. پاسخ به متد LOCK همچنین دارای هدر Lock-Token است که حاوی اطلاعات لازم در مورد قفل گرفته شده است.

Lock-Token = "Lock-Token" ":" Coded-URL

Overwrite

توسط متدهای MOVE و COPY برای تعیین اينکه آيا مقصود باید رونويسی شود يا خير استفاده می‌شود. برای جزئيات بيشتر، بحث متدهای COPY و MOVE را در ادامه اين فصل ببینيد.

Overwrite = "Overwrite" ":" ("T" | "F")

Timeout

هدر درخواستی که توسط يك کلاینت برای تعیین مقدار زمان قفل مورد نظر استفاده می‌شود. برای اطلاعات بيشتر، به بخش «Lock refreshes and the Timeout header» مراجعه کنيد.

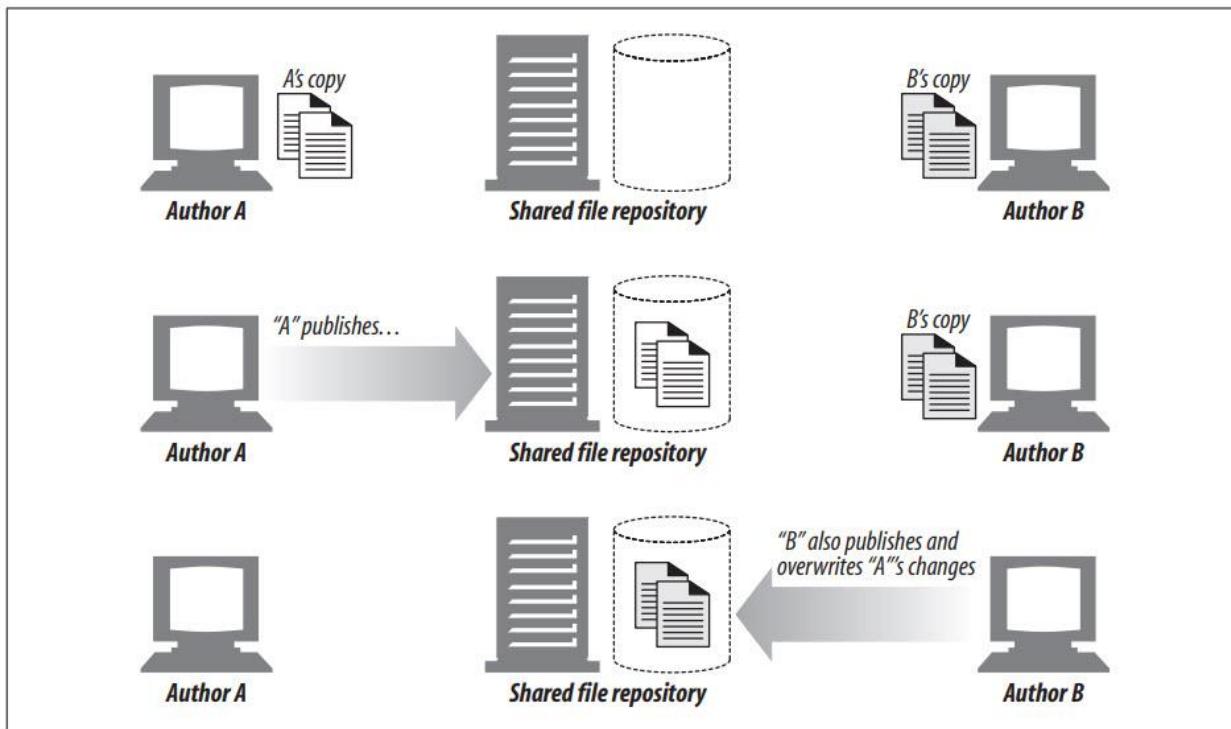


```
TimeOut = "Timeout" ":" 1#TimeType  
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other)  
DAVTimeOutVal = 1*digit  
Other = "Extend" field-value
```

اکنون که هدف و اجرای WebDAV را ترسیم کردیم، بیایید با دقت بیشتری به توابع ارائه شده نگاه کنیم.

WebDAV Locking and Overwrite Prevention

طبق تعریف، همکاری به بیش از یک نفر نیاز دارد که روی یک سند معین کار کنند. مشکل ذاتی مرتبط با همکاری در شکل زیر نشان داده شده است.



در این مثال، نویسندهای A و B به طور مشترک در حال نوشتن مشخصات هستند. A و B به طور مستقل مجموعه‌ای از تغییرات را در سند ایجاد می‌کنند. A سند به روز شده را به مخزن Push می‌دهد و در مرحله بعد، B نسخه خود را از سند در مخزن Post می‌کند. متأسفانه، چون B هرگز از تغییرات A اطلاعی نداشت، هرگز نسخه خود را با نسخه A ادغام نکرد و در نتیجه کار A از بین رفت.





برای بهبود این مشکل، WebDAV از مفهوم قفل کردن پشتیبانی می‌کند. قفل کردن به تنها بی مشکل را به طور کامل حل نمی‌کند. پشتیبانی از نسخه و پیام رسانی^{۴۹} برای تکمیل راه حل مورد نیاز است.

از دو نوع قفل پشتیبانی می‌کند:

- قفل نوشتن انحصاری^{۵۰} یک منبع یا مجموعه
- قفل نوشتن مشترک^{۵۱} یک منبع یا مجموعه

یک قفل نوشتن انحصاری امتیازات نوشتن را فقط برای صاحب قفل تضمین می‌کند. این نوع قفل درگیری‌های احتمالی را به طور کامل از بین می‌برد. قفل نوشتن مشترک، به گروهی از افراد اجازه می‌دهد تا روی یک سند معین کار کنند. این نوع قفل در محیطی که همه نویسنده‌گان از فعالیت‌های یکدیگر آگاه هستند به خوبی کار می‌کند. WebDAV مکانیزم کشف ویژگی را از طریق PROPFIND برای تعیین پشتیبانی از قفل و انواع قفل‌های پشتیبانی شده فراهم می‌کند.

دو روش جدید برای پشتیبانی از قفل دارد: LOCK و UNLOCK.

برای انجام قفل کردن، باید مکانیزمی برای شناسایی نویسنده وجود داشته باشد. WebDAV به Digest Authentication نیاز دارد (که در فصل ۱۳ به آن پرداخته شد).

هنگامی که یک قفل اعطا می‌شود، سرور رمزی را که در سراسر دامنه منحصر به فرد است به کلاینت برمی‌گرداند. مشخصات به این opaque lock token URI scheme اشاره دارد. زمانی که کلاینت متعاقباً می‌خواهد نوشتنی را انجام دهد، به سرور متصل می‌شود و توالی Digest Authentication را تکمیل می‌کند. هنگامی که احراز هویت کامل شد، سرویس گیرنده WebDAV رمز قفل را همراه با درخواست PUT ارائه می‌دهد. بنابراین، ترکیب کاربر صحیح و lock token برای تکمیل نوشتن مورد نیاز است.

The LOCK Method

یکی از ویژگی‌های قدرتمند WebDAV توانایی آن در قفل کردن چندین منبع با یک درخواست LOCK است. قفل WebDAV نیازی به اتصال کلاینت به سرور ندارد.

برای مثال، در اینجا یک درخواست LOCK ساده وجود دارد:

⁴⁹ Versioning and messaging support

⁵⁰ Exclusive

⁵¹ Shared





```
LOCK /ch-publish.fm HTTP/1.1
Host: minstar
Content-Type: text/xml
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Content-Length: 201
```

```
<?xml version="1.0"?>
<a:lockinfo xmlns:a="DAV:">
    <a:lockscope><a:exclusive/></a:lockscope>
    <a:locktype><a:write/></a:locktype>
    <a:owner><a:href>AuthorA</a:href></a:owner>
</a:lockinfo>
```

XML ارسال شده دارای عنصر `<lockinfo>` به عنوان عنصر پایه است. در ساختار `<lockinfo>`, سه عنصر فرعی وجود دارد:

`<locktype>`

نوع قفل را نشان می‌دهد. در حال حاضر تنها یکی وجود دارد و آن هم "write" است.

`<lockscope>`

نشان می‌دهد که آیا این یک قفل انحصاری است یا یک قفل مشترک.

`<owner>`

فیلد با شخصی که قفل فعلی را دارد تنظیم می‌شود.

در اینجا یک پاسخ موفقیت آمیز به درخواست LOCK ما آمده است:





HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 20:56:18 GMT
Content-Type: text/xml
Content-Length: 419

```
<?xml version="1.0"?>
<a:prop xmlns:a="DAV:">
<a:lockdiscovery><a:activelock>
<a:locktype><a:write/></a:locktype>
<a:lockscope><a:exclusive/></a:lockscope>
<a:owner xmlns:a="DAV:"><a:href>AutherA</a:href></a:owner>
<a:locktoken><a:href>opaqueloocktoken:*****</a:href></a:locktoken>
<a:depth>0</a:depth>
<a:timeout>Second-180</a:timeout>
</a:activelock></a:lockdiscovery>
</a:prop>
```

عنصر `<lockdiscovery>` به عنوان یک Container برای اطلاعات در مورد قفل عمل می‌کند. در عنصر `<lockdiscovery>` یک عنصر فرعی `<activelock>` تعبیه شده است که اطلاعات ارسال شده همراه با درخواست (`<owner>`, `<locktype>`) را در خود نگهداری می‌کند. علاوه بر این، `<activelock>` دارای عناصر فرعی زیر است:

`<locktoken>`

به طور منحصر به فرد قفل را در یک طرح URI به نام `opaqueloocktoken` شناسایی می‌کند. با توجه به ماهیت بدون حالت HTTP، این توکن برای شناسایی مالکیت قفل در درخواست‌های آینده استفاده می‌شود.

`<depth>`

مقدار هدر Depth را منعکس می‌کند.

`<timeout>`

مدت زمان مربوط به قفل را نشان می‌دهد. در پاسخ فوق (شکل پیشین)، مقدار timeout ۱۸۰ ثانیه است.





The opaquelocktoken scheme

برای ارائه یک توکن منحصر به فرد در همه منابع برای همه زمان‌ها طراحی شده است. برای تضمین منحصر به فرد بودن، مشخصات WebDAV استفاده از مکانیسم universal unique identifier یا UUID را که در ISO-11578 توضیح داده شده است، الزامی می‌کند.

وقتی نوبت به اجرای واقعی می‌رسد، مقداری آزادی عمل وجود دارد. سرور این انتخاب را دارد که برای هر درخواست LOCK یک UUID ایجاد کند، یا یک UUID واحد تولید کند و با اضافه کردن کاراکترهای اضافی در پایان، منحصر به فرد بودن را حفظ کند. برای ملاحظات عملکرد، انتخاب دوم بهتر است. با این حال، اگر سرور اجرای گزینه دوم را انتخاب کند، باید تضمین کند که هیچ یک از پسوندهای اضافه شده هرگز مجدد استفاده نخواهد شد.

The <lockdiscovery> XML element

عنصر `<lockdiscovery>` XML مکانیزمی برای کشف قفل فعال فراهم می‌کند. اگر دیگران سعی کنند فایل را قفل کنند در حالی که یک قفل در جای خود است، یک عنصر `<lockdiscovery>` دریافت خواهد کرد که مالک فعلی را نشان می‌دهد. عنصر `<lockdiscovery>` همه قفل‌های برجسته را به همراه ویژگی‌های آن‌ها فهرست می‌کند.

Lock refreshes and the Timeout header

برای بازخوانی یک قفل، یک کلاینت باید یک درخواست قفل را با نشانه قفل در هدر `If` ارسال کند. مقدار بازگردانده شده ممکن است با مقادیر مهلت زمانی قبلی متفاوت باشد.

به جای پذیرش مقدار زمان پایان داده شده توسط سرور، یک سرویس گیرنده ممکن است مقدار وقفه مورد نیاز در درخواست `LOCK` را نشان دهد. این کار از طریق هدر `Timeout` Syntax انجام می‌شود. هدر `Timeout` به کلاینت اجازه می‌دهد تا چند گزینه را در یک لیست جدا شده با کاما مشخص کند. مثلا:

Timeout : Infinite, Second-86400

سرور موظف به رعایت هیچ یک از گزینه‌ها نیست. با این حال، لازم است زمان انقضای قفل در عنصر `XML` ارائه شود. در همه موارد، زمان پایان قفل تنها یک دستورالعمل است و لزوماً برای سرور الزام آور نیست. ممکن است مدیر یک بازن Shanی دستی انجام دهد، یا یک رویداد غیرعادی دیگر ممکن است باعث شود سرور قفل را بازن Shanی کند. کلاینت‌ها باید از قفل‌های طولانی خودداری کنند.





با وجود این موارد اولیه، ممکن است "lost update problem" نشان داده شده در شکل پیشین را به طور کامل حل نکنیم. برای حل کامل آن، یک Cooperative Event System با کنترل نسخه‌سازی مورد نیاز است.

The UNLOCK Method

متدهای UNLOCK قفل یک منبع را به صورت زیر حذف می‌کند:

```
UNLOCK /ch-publish.fm HTTP/1.1
Host: minstar.inktomi.com
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Lock-Token:
opaquelocktoken:*****
```

```
HTTP/1.1 204 OK
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 20:56:18 GMT
```

همانند بسیاری از درخواست‌های مدیریت منابع، WebDAV برای موفقیت UNLOCK دو الزام دارد: تکمیل قبلی یک توالی Digest Authentication موفقیت‌آمیز، و تطبیق رمز قفل که در هدر Lock-Token ارسال می‌شود.

در صورت موفقیت آمیز بودن باز کردن قفل، کد وضعیت ۲۰۴ بدون محتوا به کلاینت بازگردانده می‌شود. جدول زیر کدهای وضعیت ممکن را با متدهای LOCK و UNLOCK خلاصه می‌کند.



Status code	Defined by	Method	Effect
200 OK	HTTP	LOCK	Indicates successful locking.
201 Created	HTTP	LOCK	Indicates that a lock on a nonexistent resource succeeded by creating the resource.
204 No Content	HTTP	UNLOCK	Indicates successful unlocking.
207 Multi-Status	WebDAV	LOCK	The request was for locking multiple resources. Not all status codes returned were the same. Hence, they are all encapsulated in a 207 response.
403 Forbidden	HTTP	LOCK	Indicates that the client does not have permission to lock the resource.
412 Precondition Failed	HTTP	LOCK	Either the XML sent with the LOCK command indicated a condition to be satisfied and the server failed to complete the required condition, or the lock token could not be enforced.
422 Unprocessable Property	WebDAV	LOCK	Inapplicable semantics—an example may be specifying a non-zero Depth for a resource that is not a collection.
423 Locked	WebDAV	LOCK	Already locked.
424 Failed Dependency	WebDAV	UNLOCK	UNLOCK specifies other actions and their success as a condition for the unlocking. This error is returned if the dependency fails to complete.

Properties and META Data

ویژگی‌ها^{۵۲} اطلاعات مربوط به منبع را توصیف می‌کنند، از جمله نام نویسنده، تاریخ اصلاح، رتبه‌بندی محتوا، و غیره. تگ‌های متا در HTML مکانیزمی برای قرار دادن این اطلاعات به عنوان بخشی از محتوا فراهم می‌کنند. با این حال، بسیاری از منابع (مانند هر داده باینری) هیچ قابلیتی برای قرار دادن داده‌های META ندارند.

یک سیستم مشارکتی توزیع شده مانند WebDAV پیچیدگی بیشتری را به نیاز دارایی^{۵۳} اضافه می‌کند. به عنوان مثال، یک ویژگی نویسنده را در نظر بگیرید: هنگامی که یک سند ویرایش می‌شود، این ویژگی باید به روز شود تا نویسنده‌گان جدید را منعکس کند. WebDAV چنین ویژگی‌هایی را که به صورت پویا قابل تغییر هستند، ویژگی‌های «live» می‌نامد. خواص ثابت‌تر، مانند Content-Type، خواص "dead" نامیده می‌شوند.

برای پشتیبانی از کشف و اصلاح ویژگی‌ها، WebDAV HTTP PROPFIND و PROPPATCH گسترش می‌دهد. مثال‌ها و عناصر XML مربوطه در بخش‌های زیر توضیح داده شده‌اند.

⁵² Properties

⁵³ property requirement



The PROPFIND Method

متد PROPFIND (property find) برای بازیابی ویژگی‌های یک فایل داده شده یا گروهی از فایل‌ها (همچنین به عنوان "collection" شناخته می‌شود) استفاده می‌شود. PROPFIND از سه نوع عملیات پشتیبانی می‌کند:

- درخواست تمام ویژگی‌ها و مقادیر آن‌ها.
- مجموعه‌ای از خواص و مقادیر انتخاب شده را درخواست کنید.
- تمام نام property‌ها را درخواست کنید.

در اینجا سناریویی وجود دارد که در آن تمام ویژگی‌ها و مقادیر آن‌ها درخواست می‌شود:

```
PROPFIND /ch-publish.fm HTTP/1.1
Host: minstar.inktomi.com
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Depth: 0
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 0
```

عنصر درخواست <propfind> ویژگی‌هایی را که باید از یک متد PROPFIND برگردانده شوند را مشخص می‌کند. فهرست زیر چند عنصر XML را که با درخواست‌های PROPFIND استفاده می‌شوند، خلاصه می‌کند:

<allprop>

مستلزم بازگشت همه نام‌ها و مقادیر دارایی است. برای درخواست همه ویژگی‌ها و مقادیر آن‌ها، یک سرویس گیرنده WebDAV ممکن است یک عنصر فرعی XML را به عنوان بخشی از عنصر ارسال کند، یا درخواستی را بدون بدن ارسال کند.

<propname>

مجموعه‌ای از نام‌های دارایی را که باید برگردانده شوند را مشخص می‌کند.

<prop>

یک عنصر فرعی از عنصر <propfind> بوده و خاصیت خاصی را مشخص می‌کند که مقدار باید برگردانده شود .<a:prop> <a:owner />..... </a:prop> به عنوان مثال:

در اینجا پاسخی به یک نمونه درخواست PROPFIND آمده است:





HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0

.....

```
<?xml version="1.0"?>
<a:multistatus xmlns:b="urn:uuid:*****/" xmlns:c="xml:" xmlns:a="DAV:">
<a:response>
  <a:href>http://minstar/ch-publish.fm </a:href>
  <a:propstat>
    <a:status>HTTP/1.1 200OK</a:status>
    <a:prop>
      <a:getcontentlength b:dt="int">1155</a:getcontentlength>
      .....
      .....
      <a:ishidden b:dt="boolean">0</a:ishidden>
      <a:iscollection b:dt="boolean">0</a:iscollection>
    </a:prop>
  </a:propstat>
</a:response></a:multistatus>
```

در این مثال، سرور با کد 207 Multi-Status WebDAV از پاسخ می‌دهد. PROPFIND و WebDAV دیگر استفاده می‌کند که به طور همزمان روی چندین منبع عمل می‌کنند و به طور بالقوه پاسخ‌های متفاوتی برای هر منبع دارند.

چند عنصر XML در پاسخ باید تعریف شود:

<multistatus>

یک Container برای پاسخ‌های متعدد.

<href>

URI منبع را شناسایی می‌کند.

<status>

حاوی کد وضعیت HTTP برای درخواست خاص است.

<propstat>





یک عنصر `<status>` و یک عنصر `<prop>` را گروه بندی می‌کند. عنصر `<prop>` ممکن است شامل یک یا چند جفت `name/value` ویژگی برای منبع داده شده باشد.

در پاسخ نمونه ذکر شده در بالا، پاسخ برای یک URI، `http://minstar/chpublish.fm` است. عنصر `<propstat>` یک عنصر `<status>` و یک عنصر `<prop>` را تعبیه می‌کند. برای این URI، سرور یک پاسخ `OK 200`، همانطور که توسط عنصر `<status>` تعریف شده است، برگرداند. توجه داشته باشید که عنصر `<prop>` چندین عنصر فرعی دارد. فقط برخی از آن‌ها در مثال ذکر شده است.

یکی از کاربردهای فوری PROPFIND پشتیبانی از Directory Listing است. با توجه به قابلیت بیان یک درخواست PROPFIND، یک فراخوان می‌تواند کل سلسله مراتب مجموعه را با تمام ویژگی‌های موجودیت‌های منفرد بازیابی کند.

The PROPPATCH Method

متدهای PROPPATCH یک مکانیسم Atomic برای تنظیم یا حذف چندین ویژگی در یک منبع داده شده فراهم می‌کند. Atomic بودن تضمین می‌کند که یا همه درخواست‌ها موفقیت‌آمیز هستند یا هیچ‌کدام از آن‌ها آن را انجام نداده‌اند.

عنصر پایه XML برای متدهای PROPPATCH `<propertyupdate>` است. این به عنوان یک Container برای تمام ویژگی‌هایی که نیاز به بروزرسانی دارند عمل می‌کند. عناصر `<set>` و `<remove>` برای تعیین عملیات استفاده می‌شوند:

`<set>`

مقادیر ویژگی‌هایی که باید تنظیم شوند را مشخص می‌کند. `<set>` حاوی یک یا چند عنصر فرعی `<prop>` است که به نوبه خود شامل جفت `name/value` خصوصیاتی است که باید برای منبع تنظیم شوند. اگر ویژگی از قبل وجود داشته باشد، مقدار جایگزین می‌شود.



```
<d:propertyupdate xmlns:d="DAV:" xmlns:o="http://name-space/scheme/">
  <d:set>
    <d:prop>
      <o:owner>Author A</o:owner>
    </d:prop>
  </d:set>

  <d:remove>
    <d:prop>
      <o:owner/>
    </d:prop>
  </d:remove>
</d:propertyupdate>
```

پاسخ به درخواست‌های PROPPATCH بسیار شبیه به درخواست‌های PROPFIND است. برای اطلاعات بیشتر به RFC 2518 مراجعه کنید.

جدول زیر کدهای وضعیت متدهای PROPPATCH و PROPFIND را خلاصه می‌کند.

Status code	Defined by	Methods	Effect
200 OK	HTTP	PROPFIND, PROPPATCH	Command success.
207 Multi-Status	WEBDAV	PROPFIND, PROPPATCH	When acting on one or more resources (or a collection), the status for each object is encapsulated into one 207 response. This is a typical success response.
401 Unauthorized	HTTP	PROPPATCH	Requires authorization to complete the property modification operation.
403 Forbidden	HTTP	PROPFIND, PROPPATCH	For PROPFIND, the client is not allowed to access the property. For PROPPATCH, the client may not change the property.
404 Not Found	HTTP	PROPFIND	No such property.
409 Conflict	HTTP	PROPPATCH	Conflict of update semantics—for example, trying to update a read-only property.
423 Locked	WebDAV	PROPPATCH	Destination resource is locked and there is no lock token or the lock token does not match.
507 Insufficient Storage	WebDAV	PROPPATCH	Not enough space for registering the modified property.

Collections and Namespace Management

مجموعه به گروه بندی منطقی یا فیزیکی منابع در یک سلسله مراتب از پیش تعریف شده اشاره دارد. یک مثال کلاسیک از یک مجموعه یک دایرکتوری است. مانند دایرکتوری‌ها در یک سیستم فایل،





مجموعه‌ها به عنوان Container های منابع دیگر، از جمله مجموعه‌های دیگر (معادل دایرکتوری‌های روی سیستم فایل) عمل می‌کنند.

XML از مکانیسم فضای نام XML استفاده می‌کند. برخلاف فضای نام سنتی، پارتبیشن‌های فضای نام امکان کنترل ساختاری دقیق را فراهم می‌کنند و در عین حال از هرگونه برخورد فضای نام جلوگیری می‌کنند.

WebDAV پنج متد برای دستکاری فضای نام ارائه می‌دهد: MOVE، COPY، MKCOL، DELETE و PROPFIND. قبلاً در این فصل مورد بحث قرار گرفت، اما اجازه دهید در مورد متد‌های دیگر صحبت کنیم.

The MKCOL Method

متد MKCOL به کلابینت‌ها اجازه می‌دهد تا مجموعه‌ای را در URL مشخص شده در سرور ایجاد کنند. در نگاه اول، ممکن است تعریف یک متد کاملاً جدید فقط برای ایجاد یک مجموعه غیر ضروری به نظر برسد. همپوشانی^{۵۴} روی متد POST یا PUT جایگزین مناسبی به نظر می‌رسد. طراحان پروتکل WebDAV این جایگزین‌ها را در نظر گرفته‌اند و همچنان متد جدیدی را تعریف کردند. برخی از دلایل این تصمیم عبارتند از:

- برای ایجاد یک مجموعه POST یا PUT، کلاینت باید مقداری «semantic glue» اضافی را همراه با درخواست ارسال کند. در حالی که این مطمئناً امکان پذیر است، تعریف یک پروتکل ad hoc ممکن است خسته کننده و مستعد خطأ باشد.
- اکثر مکانیسم‌های کنترل دسترسی بر اساس نوع متد است - فقط تعداد کمی مجاز به ایجاد و حذف منابع در مخزن هستند. اگر متد‌های دیگر را بیش از حد بارگذاری کنیم، این مکانیسم‌های کنترل دسترسی کار نمی‌کنند.

به عنوان مثال، یک درخواست ممکن است:

```
MKCOL /publishing HTTP/1.1
Host: minstar
Content-Length: 0
Connection: Keep-Alive
```

و پاسخ ممکن است این باشد:

^{۵۴} Overlaying





```
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 23:20:36 GMT
Location: http://minstar/publishing/
Content-Length: 0
```

جازه دهید چند مورد پاتولوژیک را بررسی کنیم:

- فرض کنید مجموعه از قبل وجود داشته باشد. اگر یک درخواست MKCOL/colA انجام شود و 405 Method Not Allowed بود.
 - اگر مجوز نوشتن وجود نداشته باشد، درخواست MKCOL با کد وضعیت 403 Forbidden ناموفق خواهد بود.
 - اگر درخواستی مانند MKCOL /colA/colB انجام شود و colA وجود نداشته باشد، درخواست با کد وضعیت تضاد 409 Conflict ناموفق خواهد بود.
- پس از ایجاد فایل یا مجموعه، می‌توانیم با متدهای DELETE آن را حذف کنیم.

The DELETE Method

قبلًا متدهای DELETE را در فصل ۳ دیدیم. اگر نیاز به حذف دایرکتوری داشته باشیم، هدر Depth مورد نیاز است. اگر هدر Depth مشخص نشده باشد، متدهای DELETE فرض می‌کند که هدر Depth روی بینهایت تنظیم می‌شود، یعنی تمام فایل‌های دایرکتوری و زیر شاخه‌های آن حذف می‌شوند. این پاسخ همچنین دارای یک هدر Content-Location است که مجموعه‌ای را که به تازگی حذف شده را شناسایی می‌کند. درخواست ممکن است به شرح زیر باشد:

```
DELETE /publishing HTTP/1.0
Host: minstar
```

و پاسخ ممکن است به شرح زیر باشد:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 14 May 2002 16:41:44 GMT
Content-Location: http://minstar/publishing/
Content-Type: text/xml
Content-Length: 0
```





هنگام حذف مجموعه‌ها، همیشه این احتمال وجود دارد که یک فایل در مجموعه توسط شخص دیگری قفل شده باشد و نتوان آن را حذف کرد. در چنین حالتی، خود مجموعه را نمی‌توان حذف کرد و سرور با یک کد وضعیت 207 Multi-Status پاسخ می‌دهد. درخواست ممکن است به شرح زیر باشد:

```
DELETE /publishing HTTP/1.0  
Host: minstar
```

و پاسخ ممکن است به شرح زیر باشد:

```
HTTP/1.1 207 Multi-Status  
Server: Microsoft-IIS/5.0  
Content-Location: http://minstar/publishing/  
.....  
<?xml version="1.0"?>  
<a:multistatus xmlns:a="DAV:">  
<a:response>  
<a:href>http://minstar/index3/ch-publish.fm</a:href>  
<a:status> HTTP/1.1 423 Locked </a:status>  
</a:response>  
</a:multistatus>
```

در این تراکنش، عنصر XML حاوی کد وضعیت 403 Locked است که نشان می‌دهد منبع توسط کاربر دیگری قفل شده است.

The COPY and MOVE Methods

مانند MKCOL، جایگزین‌هایی برای تعریف متدهای جدید برای عملیات COPY و MOVE وجود دارد. یکی از این جایگزین‌ها برای متد COPY انجام یک درخواست GET بر روی منبع، در نتیجه دانلود منبع، و سپس بارگذاری مجدد آن به سرور با درخواست PUT است. سناریوی مشابهی را می‌توان برای MOVE (با عملیات اضافی) متصور شد. با این حال، این فرآیند به خوبی مقیاس نمی‌شود - همه مسائل مربوط به مدیریت یک عملیات MOVE یا COPY در یک مجموعه چند سطحی را در نظر بگیرید.

هر دو متد COPY و MOVE از URL درخواست به عنوان منبع و از محتویات هدر HTTP مقصد به عنوان هدف استفاده می‌کنند. متد MOVE برخی کارهای اضافی را فراتر از متد COPY انجام می‌دهد: URL منبع را در مقصد کپی می‌کند، یکپارچگی URI جدید ایجاد شده را بررسی می‌کند و سپس منبع را حذف می‌کند. درخواست ممکن است به شرح زیر باشد:





```
{COPY,MOVE} /publishing HTTP/1.1  
Destination: http://minstar/pub-new  
Depth: infinity  
Overwrite: T  
Host: minstar
```

و پاسخ ممکن است به شرح زیر باشد:

```
HTTP/1.1 201 Created  
Server: Microsoft-IIS/5.0  
Date: Wed, 15 May 2002 18:29:53 GMT  
Location: http://minstar.inktomi.com/pub-new/  
Content-Type: text/xml  
Content-Length: 0
```

هنگام اقدام روی یک مجموعه، رفتار COPY یا MOVE تحت تأثیر هدر Depth قرار می‌گیرد. در غیاب هدر Depth، بی‌نهایت فرض می‌شود (یعنی به طور پیش فرض، کل ساختار فهرست منبع کپی یا منتقل می‌شود). اگر Depth روی صفر تنظیم شود، متدهای MOVE و COPY فقط برای منبع اعمال می‌شود. اگر ما یک کپی یا جابجایی یک مجموعه را انجام می‌دهیم، فقط مجموعه‌ای که دارای ویژگی‌های یکسان با ویژگی‌های منبع می‌باشد، در مقصد ایجاد می‌شود - هیچ عضو داخلی مجموعه کپی یا منتقل نمی‌شود.

به دلایل واضح، تنها مقدار Depth از بی‌نهایت با متدهای MOVE و COPY مجاز است.

Overwrite header effect

متدهای MOVE و COPY همچنین ممکن است از هدر Overwrite استفاده کنند. هدر Overwrite را می‌توان روی T یا F تنظیم کرد. اگر روی T تنظیم شده باشد و مقصد وجود داشته باشد، قبل از عملیات COPY یا MOVE یک DELETE با مقدار Depth بی‌نهایت در منبع مقصد انجام می‌شود. اگر پرچم Overwrite روی F تنظیم شود و منبع مقصد وجود داشته باشد، عملیات با شکست مواجه خواهد شد.

COPY/MOVE of properties

هنگامی که یک مجموعه یا یک عنصر کپی می‌شود، تمام ویژگی‌های آن به طور پیش فرض کپی می‌شود. با این حال، یک درخواست ممکن است حاوی یک بدن XML اختیاری باشد که اطلاعات اضافی را برای عملیات ارائه می‌کند. می‌توانید مشخص کنید که برای موفقیت عملیات باید تمام ویژگی‌ها با موفقیت کپی شوند یا اینکه مشخص کنید کدام ویژگی‌ها باید برای موفقیت عملیات کپی شوند.





چند مورد پاتولوژیک که باید در نظر گرفته شود عبارتند از:

- فرض کنید COPY یا MOVE روی خروجی یک برنامه CGI یا اسکریپت دیگری که محتوا تولید می‌کند اعمال می‌شود. برای حفظ مفهوم^{۵۵}، اگر قرار است فایلی که توسط یک اسکریپت CGI تولید شده است کپی یا منتقل شود، WebDAV عناصر "link" و "src" را ارائه می‌دهد که به محل برنامه‌ای که صفحه را ایجاد کرده اشاره می‌کند.
- متدهای COPY و MOVE ممکن است نتوانند به طور کامل همه ویژگی‌های Live را کپی کنند. به عنوان مثال، یک برنامه CGI را در نظر بگیرید. اگر دور از فهرست cgi-bin کپی شود، ممکن است دیگر اجرا نشود. مشخصات کنونی WebDAV باعث می‌شود COPY و MOVE به یک راه حل «best» تبدیل شود که تمام ویژگی‌های استاتیک و ویژگی‌های Live مناسب را کپی می‌کند.

Locked resources and COPY/MOVE

اگر منبعی در حال حاضر قفل است، هر دو COPY و MOVE از جابجایی یا تکثیر قفل در مقصد منع می‌شوند. در هر دو حالت، اگر قرار باشد مقصد تحت یک مجموعه موجود با قفل خاص خود ایجاد شود، منبع تکراری یا جابجا شده به قفل اضافه می‌شود. به مثال زیر توجه کنید:

```
COPY /publishing HTTP/1.1
Destination: http://minstar/archived/publishing-old
```

بیایید فرض کنیم که /archived و /publishing در حال حاضر تحت دو قفل مختلف، lock1 و lock2 قرار دارند. وقتی عملیات COPY کامل شد، همچنان در محدوده /publishing قرار دارد، در حالی که، به دلیل انتقال به مجموعه‌ای که قبلًاً توسط lock2 قفل شده است، lock2 publishing-old به publishing-old اضافه می‌شود. اگر عملیات یک MOVE بود، فقط publishing-old به lock2 اضافه می‌شود.

جدول زیر اکثر کدهای وضعیت ممکن را برای متدهای MOVE, COPY, DELETE, MKCOL و فهرست می‌کند.

⁵⁵ semantics





Status code	Defined by	Methods	Effect
102 Processing	WebDAV	MOVE, COPY	If the request takes longer than 20 seconds, the server sends this status code to keep clients from timing out. This usually is seen with a COPY or MOVE of a large collection.
201 Created	HTTP	MKCOL, COPY, MOVE	For MKCOL, a collection has been created. For COPY and MOVE, a resource/collection was copied or moved successfully.
204 No Content	HTTP	DELETE, COPY, MOVE	For DELETE, a standard success response. For COPY and MOVE, the resource was copied over successfully or moved to replace an existing entity.
207 Multi-Status	WebDAV	MKCOL, COPY, MOVE	For MKCOL, a typical success response. For COPY and MOVE, if an error is associated with a resource other than the request URI, the server returns a 207 response with the XML body detailing the error.
403 Forbidden	HTTP	MKCOL, COPY, MOVE	For MKCOL, the server does not allow creation of a collection at the specified location. For COPY and MOVE, the source and destination are the same.
409 Conflict	HTTP	MKCOL, COPY, MOVE	In all cases, the methods are trying to create a collection or a resource when an intermediate collection does not exist—for example, trying to create colA/colB when colA does not exist.
412 Precondition Failed	HTTP	COPY, MOVE	Either the Overwrite header is set to F and the destination exists, or the XML body specifies a certain requirement (such as keeping the “liveness” property) and the COPY or MOVE methods are not able to retain the property.
415 Unsupported Media Type	HTTP	MKCOL	The server does not support or understand the creation of the request entity type.
422 Unprocessable Entity	WebDAV	MKCOL	The server does not understand the XML body sent with the request.
423 Locked	WebDAV	DELETE, COPY, MOVE	The source or the destination resource is locked, or the lock token supplied with the method does not match.
502 Bad Gateway	HTTP	COPY, MOVE	The destination is on a different server and permissions are missing.
507 Insufficient Storage	WebDAV	MKCOL COPY	There is not enough free space to create the resource.

Enhanced HTTP/1.1 Methods

GET مفاهیم متدهای WebDAV را تغییر می‌دهد. مفهوم برای متدهای OPTIONS, PUT, HTTP DELETE و HEAD بدون تغییر باقی می‌ماند. عملیات انجام شده توسط POST همیشه توسط اجرای سرور خاص تعریف می‌شود و WebDAV هیچ یک از معنای POST را تغییر نمی‌دهد. با توجه به اینکه قبلاً متدهای OPTIONS, PUT و DELETE را در پوشش دادیم، در اینجا در مورد متدهای OPTIONS بحث خواهیم کرد.





The PUT method

اگرچه PUT توسط WebDAV تعریف نشده است، اما تنها راه برای نویسنده برای انتقال محتوا به یک سایت مشترک است. ما در مورد عملکرد کلی PUT در فصل ۳ بحث کردیم. WebDAV رفتار خود را برای پشتیبانی از قفل تغییر می‌دهد.

به مثال زیر توجه کنید:

```
PUT /ch-publish.fm HTTP/1.1
Accept: */
If:<http://minstar/index.htm>(<opaque locktoken:*****>)
User-Agent: DAV Client (C)
Host: minstar.inktomi.com
Connection: Keep-Alive
Cache-Control: no-cache
Content-Length: 1155
```

برای پشتیبانی از قفل، WebDAV یک هدر If را به درخواست PUT اضافه می‌کند. در تراکنش فوق، معنای هدر If بیان می‌کند که اگر نشانه قفل مشخص شده با هدر If با قفل منبع (در این مورد ch-publish.fm) مطابقت داشته باشد، باید عملیات PUT انجام شود. هدر If نیز با چند متادیگر مانند DELETE، PROPPATCH و غیره استفاده می‌شود.

The OPTIONS method

ما در فصل ۳ مورد بحث قرار دادیم. این معمولاً اولین درخواستی است که یک کلاینت WebDAV-enabled ارائه می‌کند. با استفاده از متاداده OPTIONS، کلاینت سعی می‌کند قابلیت سرور WebDAV را Establish کند. تراکنشی را در نظر بگیرید که در آن درخواست به شرح زیر است:

```
OPTIONS /ch-publish.fm HTTP/1.1
Accept: /*
Host: minstar.inktomi.com
```

و در پاسخ آمده است:





```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
MS-Author-Via: DAV
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND,
PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, COPY, MOVE, PROPFIND, PROPPATCH,
SEARCH, LOCK, UNLOCK
```

چندین عنوان جالب در پاسخ به متدهای **OPTIONS** وجود دارد. یک بررسی کمی خارج از دستور^{۵۶} به شرح زیر است:

هدر DAV اطلاعات مربوط به کلاس‌های سازگاری DAV را حمل می‌کند. دو دسته از انطباق وجود دارد:

Class 1 compliance

서ور را ملزم به رعایت تمام الزامات MUST در تمام بخش‌های RFC 2518 می‌کند. اگر منبع فقط در سطح Class 1 مطابقت داشته باشد، آنرا با هدر DAV ارسال می‌کند.

Class 2 compliance

تمام الزامات Class 2 را برآورده نموده و برای متدهای LOCK پشتیبانی می‌کند. همراه با **LOCK**، انطباق Class 2 به پشتیبانی از هدرهای **Timeout** و **Lock-Token** و عناصر **<supportedlock>** و **XML** و **<lockdiscovery>** نیاز دارد. مقدار ۲ در هدر DAV نشان دهنده انطباق با Class 2 است.

در مثال بالا، هدر DAV هم انطباق با Class 1 و هم Class 2 را نشان می‌دهد.

هدر Public تمام متدهای پشتیبانی شده توسط این سرور خاص را فهرست می‌کند.

هدر Allow معمولاً شامل زیر مجموعه‌ای از متدهای هدر Public است و فقط آن دسته از متدهایی را فهرست می‌کند که در این منبع خاص مجاز هستند (ch-publish.fm).

هدر DASL نوع گرامر پرس و جو مورد استفاده در متدهای **SEARCH** را ارائه می‌دهد. در این مورد **sql** است. جزئیات بیشتر در مورد هدر DASL در <http://www.webdav.org> ارائه شده است.

^{۵۶} out-of-order





Version Management in WebDAV

با توجه به "V" در "DAV" ممکن است طعنه آمیز باشد، اما نسخه سازی، ویژگی است که اولین برش را انجام نداد. در یک محیط multi-author و مشارکتی، مدیریت نسخه بسیار مهم است. در واقع، برای رفع کامل مشکل بروزرسانی از دست رفته (نشان داده شده در شکل پیشین)، قفل کردن و نسخه سازی ضروری است. برخی از ویژگی های رایج مرتبط با نسخه سازی، امکان ذخیره و دسترسی به نسخه های قبلی سند و توانایی مدیریت تاریخچه تغییرات و هرگونه حاشیه نویسی مرتبط با جزئیات تغییرات است.

نسخه سازی در RFC 3253 به WebDAV اضافه شد.

Future of WebDAV

امروزه به خوبی پشتیبانی می شود. پیاده سازی های کاری کلاینت ها شامل IE 5.x و بالاتر، Microsoft Office و Windows Explorer است. در سمت سرور، پیاده سازی ها شامل IIS 5.x و بالاتر، آپاچی با mod_dav و بسیاری دیگر می شود. هر دو ویندوز XP و Mac OS 10.x از WebDAV پشتیبانی می کنند. بنابراین، هر برنامه ای که برای اجرا روی این سیستم عامل ها نوشته شده اند، به صورت بومی با WebDAV فعال می شوند.

For More Information

<http://officeupdate.microsoft.com/frontpage/wpp/serk>

<http://www.ietf.org/rfc/rfc2518.txt?number=2518>

<http://www.ietf.org/rfc/rfc3253.txt?number=3253>

http://www.ics.uci.edu/pub/ietf/webdav/intro/webdav_intro.pdf

http://www.ics.uci.edu/~ejw/http-future/whitehead/http_pos_paper.html

<http://www.microsoft.com/msj/0699/dav/davtop.htm>

<http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html>





فصل بیستم – Redirection and Load Balancing

در ساختار وب تمام موارد به تنها بی مربوط به HTTP نمی‌شود. داده‌های یک پیام HTTP توسط پروتکل‌های زیادی در مسیر دسترسی به وب کنترل می‌شوند. HTTP فقط به نقاط پایانی سفر یعنی فرستنده و گیرنده اهمیت می‌دهد، اما در دنیایی با Mirrored Server ها، پروکسی‌های وب و Cache، مقصد یک پیام HTTP لزوماً ساده نیست.

این فصل در مورد فناوری‌های تغییر مسیر^{۵۷} است - ابزارهای شبکه، تکنیک‌ها و پروتکل‌هایی که مقصد نهایی یک پیام HTTP را تعیین می‌کنند. فناوری‌های تغییر مسیر عموماً تعیین می‌کنند که آیا پیام به یک پروکسی، یک کش یا یک وب سرور خاص در یک Server Farm ختم می‌شود یا خیر. فناوری‌های تغییر مسیر ممکن است پیام‌های شما را به مکان‌هایی بفرستند که مشتری صریحاً درخواست نکرده است.

در این فصل، نگاهی به تکنیک‌های تغییر مسیر زیر، نحوه کارکرد آن‌ها و قابلیت‌های Load-Balancing (در صورت وجود) خواهیم داشت:

- HTTP redirection
- DNS redirection
- Anycast routing
- Policy routing
- IP MAC forwarding
- IP address forwarding
- The Web Cache Coordination Protocol (WCCP)
- The Intercache Communication Protocol (ICP)
- The Hyper Text Caching Protocol (HTCP)
- The Network Element Control Protocol (NECP)
- The Cache Array Routing Protocol (CARP)
- The Web Proxy Autodiscovery Protocol (WPAD)

Why Redirect

تغییر مسیر یک واقعیت زندگی در وب مدرن است زیرا برنامه‌های HTTP همیشه می‌خواهند سه کار را انجام دهند:

⁵⁷ redirection





- تراکنش‌های HTTP را با اطمینان انجام دهند.
- تاخیر را به حداقل برسانند.
- پهنه‌ای باند شبکه را حفظ کنند.

به این دلایل، محتوای وب اغلب در چندین مکان توزیع می‌شود. این برای قابلیت اطمینان انجام می‌شود، به طوری که اگر یک مکان از دسترسی خارج شد، مکان دیگری در دسترس باشد. این کار برای کاهش زمان پاسخ انجام می‌شود، زیرا اگر کلاینت‌ها بتوانند به یک منبع نزدیکتر دسترسی داشته باشند، محتوای درخواستی خود را سریع‌تر دریافت می‌کنند. و این کار برای کاهش تراکم شبکه با گسترش سرورهای هدف نیز انجام می‌شود. شما می‌توانید تغییر مسیر را به عنوان مجموعه‌ای از تکنیک‌ها در نظر بگیرید که به یافتن "بهترین" محتوای توزیع شده کمک می‌کند.

موضوع Load-Balancing گنجانده شده است زیرا تغییر مسیر و Load-Balancing در کنار هم وجود دارند. بیشتر استقرارهای تغییر مسیر شامل نوعی Load-Balancing هستند. یعنی می‌توانند بار پیام دریافتی را بین مجموعه‌ای از سرورها پخش کنند. بر عکس، هر شکلی از Load-Balancing شامل تغییر مسیر می‌شود، زیرا پیام‌های دریافتی باید به نحوی در میان سرورهایی باشد که بار را به اشتراک می‌گذارند.

Where to Redirect

سرورها، پراکسی‌ها و Cache‌ها همگی به عنوان سرور برای کلاینت‌ها ظاهر می‌شوند، به این معنا که یک کلاینت یک درخواست HTTP برای آن‌ها ارسال می‌کند و آن‌ها آن را پردازش می‌کنند. بسیاری از تکنیک‌های تغییر مسیر برای سرورها، پراکسی‌ها، Cache‌ها و Gateway‌ها به دلیل ویژگی‌های رایج و سرور مانندشان کار می‌کنند. سایر تکنیک‌های تغییر مسیر به طور ویژه برای یک کلاس خاص از نقطه پایانی طراحی شده‌اند و به طور کلی قابل اجرا نیستند. در بخش‌های بعدی این فصل، تکنیک‌های کلی و تکنیک‌های تخصصی را خواهیم دید.

وب سرورها درخواست‌ها را بر اساس IP انجام می‌دهند. توزیع درخواست‌ها در Duplicate Server‌ها به این معنی است که هر درخواست برای یک URL خاص باید به یک وب سرور بهینه (نزدیک‌ترین به کلاینت، یا کمترین بارگذاری یا برخی بهینه‌سازی‌های دیگر) ارسال شود. تغییر مسیر به یک سرور مانند فرستادن همه رانندگان در جستجوی بنزین به نزدیک‌ترین پمپ بنزین است.

پروکسی‌ها تمایل دارند درخواست‌ها را بر اساس پروتکل رسیدگی کنند. در حالت ایده‌آل، تمام ترافیک HTTP در همسایگی یک پروکسی باید از طریق پراکسی انجام شود. برای مثال، اگر یک Proxy Cache در نزدیکی کلاینت‌های مختلف باشد، تمام درخواست‌ها در حالت ایده‌آل از طریق





جريان می‌یابند، زیرا Cache اسناد محبوب را ذخیره می‌کند و مستقیماً به آن‌ها سرویس می‌دهد و از سفرهای طولانی‌تر و گران‌تر به سرورهای مبدأ جلوگیری می‌کند. تغییر مسیر به یک پروکسی مانند از بین بردن ترافیک در یک جاده دسترسی اصلی (بدون توجه به اینکه به کجا می‌رود) به یک میانبر محلی است.

Overview of Redirection Protocols

هدف از تغییر مسیر، ارسال پیام‌های HTTP به سرورهای وب موجود در سریع‌ترین زمان ممکن است. جهتی که یک پیام HTTP در مسیر خود از طریق اینترنت می‌گیرد، تحت تأثیر برنامه‌های کاربردی HTTP و دستگاه‌های مسیریابی است که از آن، از طریق آن و به سمت آن عبور می‌کند. مثلا:

- برنامه مرورگری که پیام کلاینت را ایجاد می‌کند می‌تواند پیکربندی شود تا آن را به یک سرور پراکسی ارسال کند.
- DNS Resolvers آدرس IP مورد استفاده برای آدرس‌دهی پیام را انتخاب می‌کنند. این آدرس IP می‌تواند برای کلاینت‌های مختلف در مکان‌های جغرافیایی مختلف متفاوت باشد.
- همانطور که پیام از طریق شبکه‌ها عبور می‌کند، به بسته‌های آدرس‌دهی شده تقسیم می‌شود. سوئیچ‌ها و روترهای آدرس دهی TCP/IP روی بسته‌ها را بررسی می‌کنند و بر اساس آن در مورد مسیریابی بسته‌ها تصمیم می‌گیرند.
- وب سرورها می‌توانند درخواست‌ها را با تغییر مسیرهای HTTP به سرورهای وب مختلف برگردانند.

پیکربندی مرورگر، DNS، مسیریابی TCP/IP و HTTP همگی مکانیسم‌هایی را برای هدایت مجدد پیام‌ها فراهم می‌کنند. توجه داشته باشید که برخی از روش‌ها، مانند پیکربندی مرورگر، فقط برای هدایت ترافیک به پراکسی‌ها منطقی هستند، در حالی که روش‌های دیگر، مانند تغییر مسیر DNS، برای ارسال ترافیک به هر سروری قابل استفاده هستند.

جدول زیر روش‌های تغییر مسیر مورد استفاده برای هدایت پیام‌ها به سرورها را خلاصه می‌کند.





Mechanism	How it works	Basis for rerouting	Limitations
HTTP redirection	Initial HTTP request goes to a first web server that chooses a "best" web server to serve the content. The first web server sends the client an HTTP redirect to the chosen server. The client resends the request to the chosen server.	Many options, from round-robin load balancing, to minimizing latency, to choosing the shortest path.	Can be slow—every transaction involves the extra redirect step. Also, the first server must be able to handle the request load.
DNS redirection	DNS server decides which IP address, among several, to return for the hostname in the URL.	Many options, from round-robin load balancing, to minimizing latency, to choosing the shortest path.	Need to configure DNS server.
Anycast addressing	Several servers use the same IP address. Each server masquerades as a backbone router. The other routers send packets addressed to the shared IP to the nearest server (believing they are sending packets to the nearest router).	Routers use built-in shortest-path routing capabilities.	Need to own/configure routers. Risks address conflicts. Established TCP connections can break if routing changes and packets associated with a connection get sent to different servers.
IP MAC forwarding	A network element such as a switch or router reads a packet's destination address; if the packet should be redirected, the switch gives the packet the destination MAC address of a server or proxy.	Save bandwidth and improve QOS. Load balance.	Server or proxy must be one hop away.
IP address forwarding	Layer-4 switch evaluates a packet's destination port and changes the IP address of a redirect packet to that of a proxy or mirrored server.	Save bandwidth and improve QOS. Load balance.	IP address of the client can be lost to the server/proxy.

جدول زیر روش‌های تغییر مسیر مورد استفاده برای هدایت پیام‌ها به سرورهای پراکسی را خلاصه می‌کند.





Mechanism	How it works	Basis for rerouting	Limitations
Explicit browser configuration	Web browser is configured to send HTTP messages to a nearby proxy, usually a cache. The configuration can be done by the end user or by a service that manages the browser.	Save bandwidth and improve QOS. Load balance.	Depends on ability to configure the browser.
Proxy auto-configuration (PAC)	Web browser retrieves a PAC file from a configuration server. The PAC file tells the browser what proxy to use for each URL.	Save bandwidth and improve QOS. Load balance.	Browser must be configured to query the configuration server.
Web Proxy Autodiscovery Protocol (WPAD)	Web browser asks a configuration server for the URL of a PAC file. Unlike PAC alone, the browser does not have to be configured with a specific configuration server.	The configuration server bases the URL on information in client HTTP request headers. Load balance.	Only a few browsers support WPAD.
Web Cache Coordination Protocol (WCCP)	Router evaluates a packet's destination address and encapsulates redirect packets with the IP address of a proxy or mirrored server. Works with many existing routers. Packet can be encapsulated, so the client's IP address is not lost.	Save bandwidth and improve QOS. Load balance.	Must use routers that support WCCP. Some topographical limitations.
Internet Cache Protocol (ICP)	A proxy cache can query a group of sibling caches for requested content. Also supports cache hierarchies.	Obtaining content from a sibling or parent cache is faster than applying to the origin server.	False cache hits can arise because only the URL is used to request content.
Cache Array Routing Protocol (CARP)	A proxy cache hashing protocol. Allows a cache to forward a request to a parent cache. Unlike with ICP, the content on the caches is disjoint, and the group of caches acts as a single large cache.	Obtaining content from a nearby peer cache is faster than applying to the origin server.	CARP cannot support sibling relationships. All CARP clients must agree on the configuration; otherwise, different clients will send the same URI to different parents, reducing hit ratios.
Hyper Text Caching Protocol (HTCP)	Participating proxy caches can query a group of sibling caches for requested content. Supports HTTP 1.0 and 1.1 headers to fine-tune cache queries.	Obtaining content from a sibling or parent cache is faster than applying to the origin server.	

General Redirection Methods

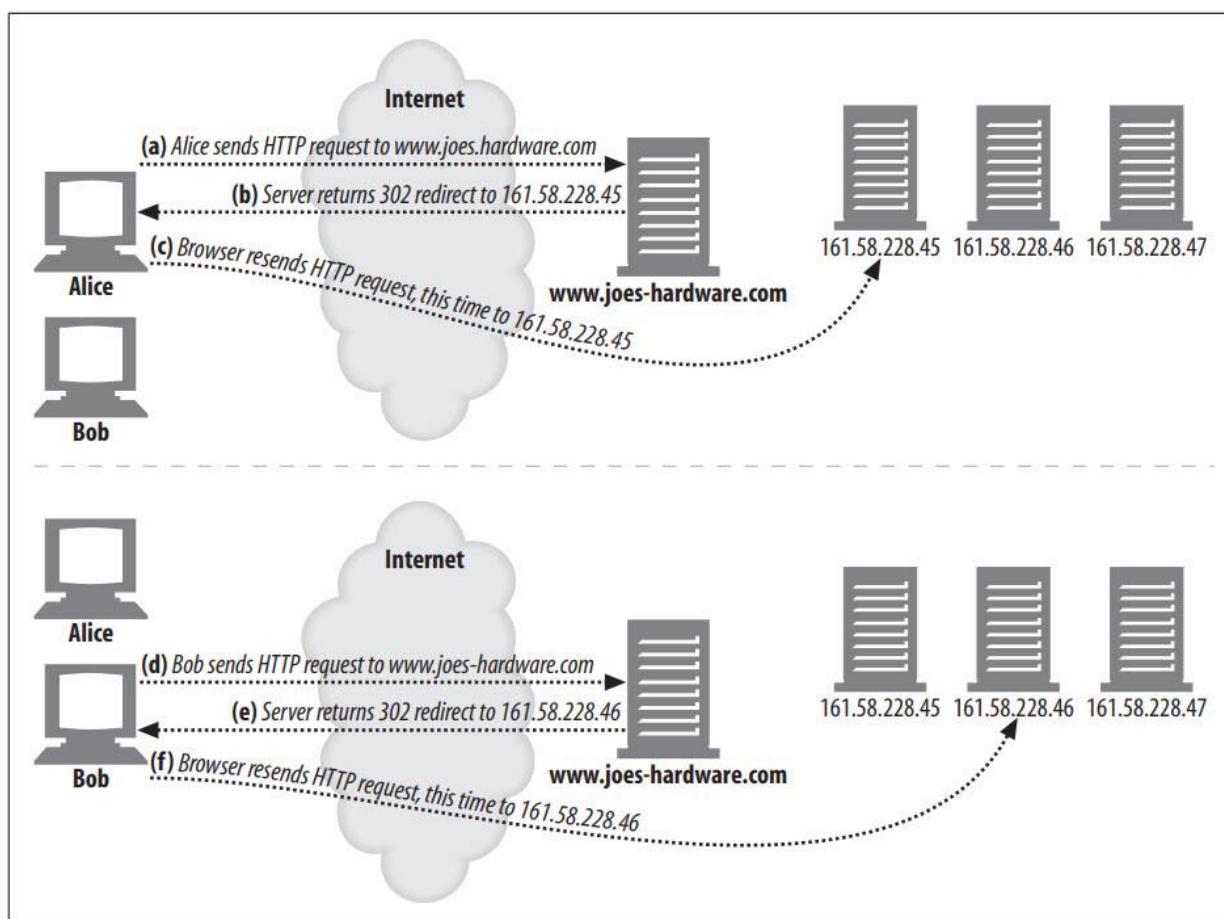
در این بخش، روش‌های مختلف تغییر مسیر که معمولاً برای سرورها و پروکسی‌ها استفاده می‌شوند را عمیق‌تر بررسی خواهیم کرد. این تکنیک‌ها را می‌توان برای هدایت ترافیک به یک سرور متفاوت (احتمالاً بهینه‌تر) یا انتقال ترافیک از طریق یک پروکسی استفاده کرد. به طور خاص، ما تغییر مسیر HTTP، DNS، تغییر مسیر IP MAC، انتقال IP anycast و حمل و نقل آدرس IP را پوشش خواهیم داد.



HTTP Redirection

سرورهای وب می‌توانند پیام‌های تغییر مسیر کوتاهی را به کلاینت‌ها ارسال کنند و به آن‌ها بگویند که در جای دیگری امتحان کنند. برخی از وب‌سایت‌ها از تغییر مسیر HTTP به عنوان شکلی ساده برای Load Balancing استفاده می‌کنند. سروری که تغییر مسیر را مدیریت می‌کند (سرور هدایت کننده) کمترین بارگذاری سرور را پیدا نموده و مرورگر را به آن سرور هدایت می‌کند. برای وب‌سایت‌های گسترده، تعیین «بهترین» سرور موجود پیچیده‌تر می‌شود، نه تنها بار سرورها، بلکه فاصله اینترنت بین مرورگر و سرور نیز در نظر گرفته می‌شود. یکی از مزیت‌های تغییر مسیر HTTP نسبت به سایر اشکال تغییر مسیر این است که سرور هدایت کننده آدرس IP کلاینت را می‌داند و در تئوری، ممکن است بتواند انتخاب آگاهانه‌تری داشته باشد.

در اینجا نحوه عملکرد تغییر مسیر HTTP آمده است.



در بخش a شکل، آلیس درخواستی را به www.joes-hardware.com ارسال می‌کند:





```
GET /hammers.html HTTP/1.0
Host: www.joes-hardware.com
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

در بخش b شکل، به جای بازگرداندن بدنه صفحه وب با کد وضعیت 200 HTTP، سرور یک پیام تغییر مسیر با کد وضعیت 302 را ارسال می‌کند:

```
HTTP/1.0 302 Redirect
Server: Stronghold/2.4.2 Apache/1.3.6
Location: http://161.58.228.45/hammers.html
```

اکنون، در بخش c شکل، مرورگر درخواست را با استفاده از URL تغییر مسیر داده شده، این بار به میزبان 161.58.228.45 ارسال می‌کند:

```
GET /hammers.html HTTP/1.0
Host: 161.58.228.45
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

کلاینت دیگر می‌تواند به سرور دیگری هدایت شود. در بخش d تا f شکل، درخواست باب به 161.58.228.46 هدایت می‌شود.

تغییر مسیر HTTP با معایبی هم همراه است:

- مقدار قابل توجهی از قدرت پردازشی از سرور اصلی مورد نیاز است تا تعیین شود به کدام سرور تغییر مسیر دهید. گاهی اوقات تقریباً به همان اندازه اسب بخار سرور برای صدور تغییر مسیر مورد نیاز است که برای ارائه خود صفحه لازم است.
- تاخیرهای کاربر افزایش می‌یابد، زیرا برای دسترسی به صفحات، دو رفت و برگشت لازم است.
- اگر سرور تغییر مسیر از دست رفته یا خراب باشد، سایت نیز دچار مشکل می‌شود.

به دلیل این نقاط ضعف، تغییر مسیر HTTP معمولاً در ترکیب با برخی از تکنیک‌های تغییر مسیر دیگر استفاده می‌شود.

DNS Redirection

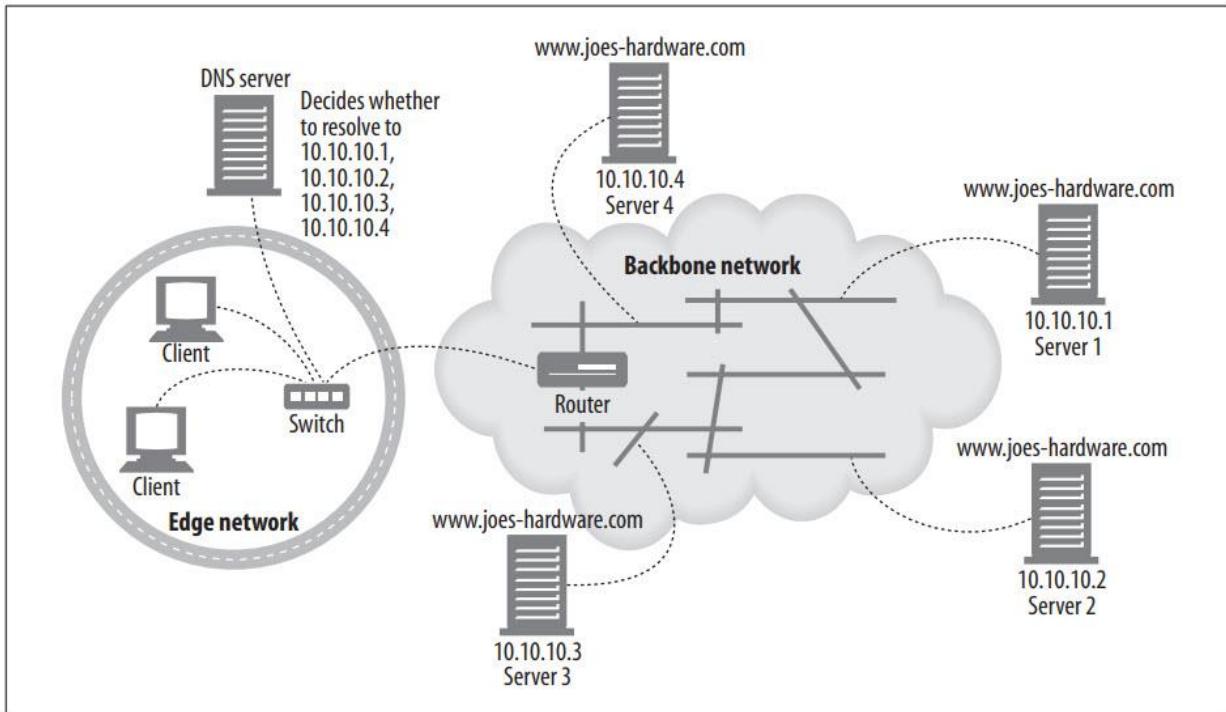
هر بار که کلاینت سعی می‌کند به وب سایت Joe's Hardware دسترسی پیدا کند، نام دامنه www.joes-hardware.com باید به یک آدرس IP تبدیل شود. DNS Resolveer ممکن است سیستم عامل خود کلاینت، یک سرور DNS در شبکه کلاینت یا یک سرور DNS راه دورتر باشد.





به چندین آدرس IP اجازه می‌دهد تا به یک دامنه منفرد مرتبط شوند و DNS Resolver ها می‌توانند پیکربندی یا برنامه ریزی شوند تا آدرس‌های IP متفاوتی را برگردانند. مبنایی که Resolver IP نشانیResolver را بر می‌گرداند می‌تواند از ساده (round robin) به پیچیده اجرا شود (مانند بررسی بار روی چندین سرور و بازگرداندن آدرس IP سرور با کمترین بارگذاری).

در شکل زیر، Joe چهار سرور را برای www.joes-hardware.com اجرا می‌کند. سرور DNS باید تصمیم بگیرد که کدام یک از چهار آدرس IP را برای www.joes-hardware.com بازگرداند. ساده‌ترین الگوریتم تصمیم‌گیری round robin یک DNS ساده است.



DNS round robin

یکی از رایج‌ترین تکنیک‌های تغییر مسیر نیز یکی از ساده‌ترین آن‌هاست. DNS round robin از ویژگی Load-Balancing برای hostname resolution در مزرعه‌ای از سرورهای وب استفاده می‌کند. این یک استراتژی Load-Balancing خالص است و هیچ عاملی را در مورد موقعیت کلاینت نسبت به سرور یا استرس فعلی روی سرور در نظر نمی‌گیرد.

بیایید ببینیم CNN.com واقعاً چه می‌کند. در اوایل ماه مه سال ۲۰۰۰، ما از ابزار nslookup برای یافتن آدرس‌های IP مرتبط با CNN.com استفاده کردیم. مثال ۲۰-۱ نتایج را نشان می‌دهد.

Example 20-1. IP addresses for www.cnn.com





```
% nslookup www.cnn.com
```

Name: cnn.com

Addresses: 207.25.71.5, 207.25.71.6, 207.25.71.7, 207.25.71.8
207.25.71.9, 207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23
207.25.71.24, 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
207.25.71.29, 207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245
207.25.71.246

Aliases: www.cnn.com

وب سایت www.cnn.com در واقع مزروعه‌ای از ۲۰ آدرس IP مجزا است! هر آدرس IP معمولاً ممکن است به یک سرور فیزیکی متفاوت ترجمه شود.

Multiple addresses and round-robin address rotation

اکثر کلاینت‌ها DNS فقط از اولین آدرس مجموعه چند آدرس استفاده می‌کنند. برای Load-Balancing، اکثر سرورهای DNS هر بار که جستجو انجام می‌شود، آدرس‌ها را می‌چرخانند.^{۵۸} این چرخش آدرس اغلب DNS round robin نامیده می‌شود.

به عنوان مثال، سه جستجوی متوالی DNS www.cnn.com ممکن است لیست‌های چرخشی از آدرس‌های IP مانند موارد نشان داده شده در مثال ۲-۲۰ را برگرداند.

^{۵۸} rotate





```
% nslookup www.cnn.com
```

```
Name:    cnn.com
```

```
Addresses: 207.25.71.5, 207.25.71.6, 207.25.71.7, 207.25.71.8  
          207.25.71.9, 207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23  
          207.25.71.24, 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28  
          207.25.71.29, 207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245  
          207.25.71.246
```

```
% nslookup www.cnn.com
```

```
Name:    cnn.com
```

```
Addresses: 207.25.71.6, 207.25.71.7, 207.25.71.8, 207.25.71.9  
          207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23, 207.25.71.24  
          207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28, 207.25.71.29  
          207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245, 207.25.71.246  
          207.25.71.5
```

```
% nslookup www.cnn.com
```

```
Name:    cnn.com
```

```
Addresses: 207.25.71.7, 207.25.71.8, 207.25.71.9, 207.25.71.12  
          207.25.71.20, 207.25.71.22, 207.25.71.23, 207.25.71.24, 207.25.71.25  
          207.25.71.26, 207.25.71.27, 207.25.71.28, 207.25.71.29, 207.25.71.30  
          207.25.71.82, 207.25.71.199, 207.25.71.245, 207.25.71.246, 207.25.71.5  
          207.25.71.6
```

در Example 20-2

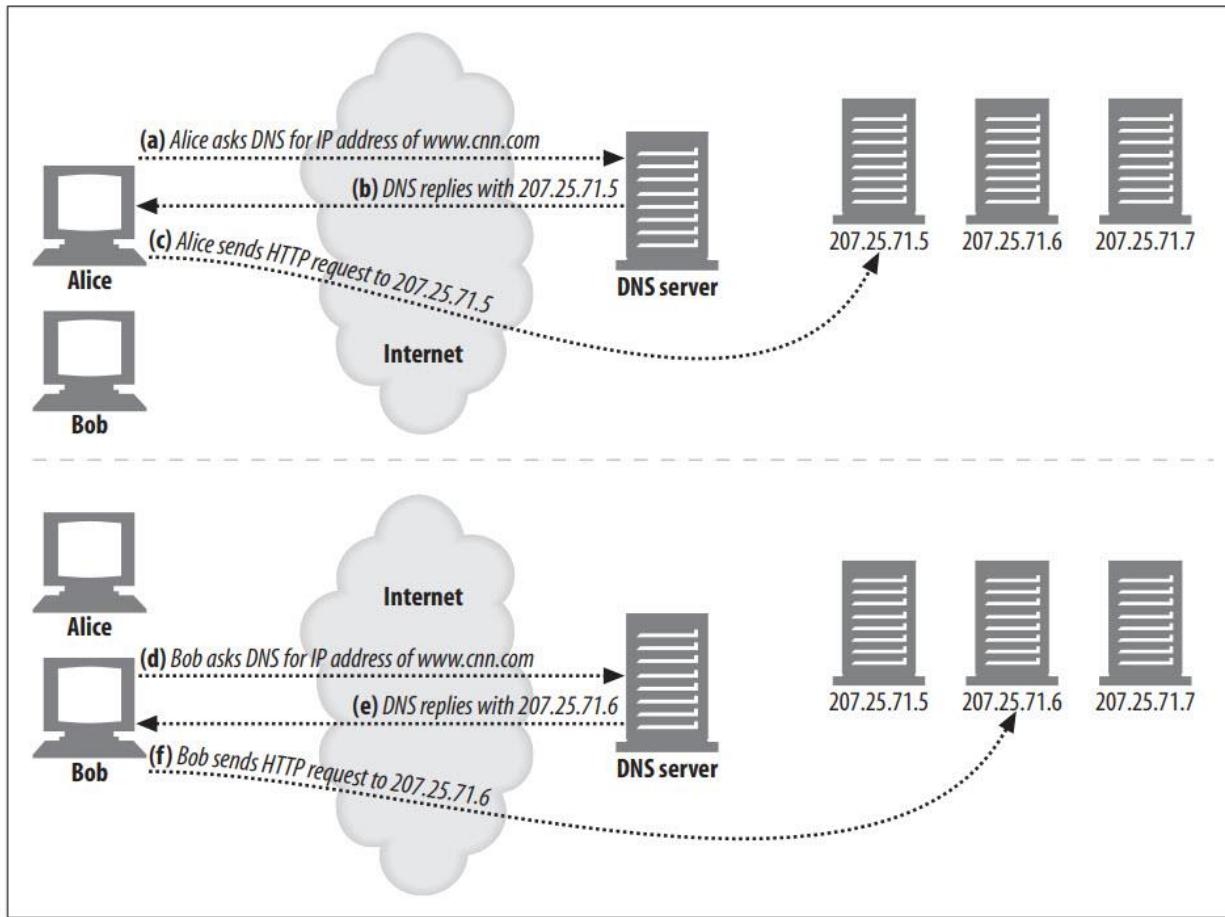
- اولین آدرس اولین جستجوی 207.25.71.5 است.
- آدرس اول دومین جستجوی DNS برابر با 207.25.71.6 است.
- اولین آدرس سومین جستجوی DNS برابر با 207.25.71.7 است.

DNS round robin for load balancing

از آنجایی که اکثر کلاینتها DNS فقط از اولین آدرس استفاده می‌کنند، DNS Rotation برای Load-Balancing بین سرورها عمل می‌کند. اگر آدرس‌ها را Rotate نمی‌کند، اکثر کلاینتها همیشه بار را به کلاینت اول ارسال می‌کنند.

شکل زیر نشان می‌دهد که چگونه Load-Balancing برای DNS Round-Robin Rotation عمل می‌کند:





وقتی آلیس سعی می‌کند به `www.cnn.com` متصل شود، آدرس IP را با استفاده از DNS جستجو نموده و `207.25.71.5` را به عنوان اولین آدرس IP دریافت می‌کند. آلیس به وب سرور `207.25.71.5` در بخش c شکل متصل می‌شود.

وقتی باب متعاقباً سعی می‌کند به `www.cnn.com` متصل شود، آدرس IP را با استفاده از DNS نیز جستجو می‌کند، اما نتیجه متفاوتی دریافت می‌کند زیرا فهرست آدرس‌ها بر اساس درخواست قبلی آلیس یک موقعیت چرخانده شده است. باب `207.25.71.6` را به عنوان اولین آدرس IP دریافت می‌کند و در بخش f شکل به این سرور متصل می‌شود.

The impact of DNS caching

DNS Rotation آدرس را در اطراف پخش می‌کند، زیرا هر جستجوی DNS به یک سرور، ترتیب متفاوتی از آدرس‌های سرور را دریافت می‌کند. با این حال، این توازن بار کامل نیست، زیرا نتایج جستجوی DNS ممکن است توسط برنامه‌ها، سیستم‌عامل‌ها و برخی از سرورهای DNS بدوفی به خاطر سپرده شده و دوباره مورد استفاده قرار گیرند. بسیاری از مرورگرهای وب، جستجوی DNS را برای یک میزبان انجام





می‌دهند، اما پس از آن بارها و بارها از همان آدرس استفاده می‌کنند تا هزینه جستجوی DNS را از بین ببرند و به این دلیل که برخی از سرورها ترجیح می‌دهند با همان کلاینت صحبت کنند. علاوه بر این، بسیاری از سیستم‌عامل‌ها جستجوی DNS را به صورت خودکار انجام می‌دهند و نتیجه را در حافظه پنهان می‌کنند، اما آدرس‌ها را نمی‌چرخانند. در نتیجه، DNS Round-Robin معمولاً یک کلاینت را متعادل نمی‌کند - یک کلاینت معمولاً برای مدت طولانی به یک سرور گیر می‌کند.

اما، حتی اگر DNS تراکنش‌های یک کلاینت را در بین نسخه‌های سرور انجام نمی‌دهد، اما کار مناسبی برای پخش بار کلی چندین کلاینت انجام می‌دهد. تازمانی که تعداد نسبتاً زیادی کلاینت با تقاضای مشابه وجود داشته باشد، بار به شکل مناسبی در بین سرورها توزیع می‌شود.

Other DNS-based redirection algorithms

ما قبلاً در مورد نحوه DNS Rotate لیست آدرس‌ها با هر درخواست بحث کردہ‌ایم. با این حال، برخی از سرورهای DNS پیشرفت‌های دیگری برای انتخاب ترتیب آدرس‌ها استفاده می‌کنند:

Load-balancing algorithms

برخی از سرورهای DNS بار روی سرورهای وب را پیگیری می‌کنند و سرورهای وب کم بارگذاری شده⁵⁹ را در جلوی لیست قرار می‌دهند.

Proximity-routing algorithms

سرورهای DNS می‌توانند سعی کنند کاربران را به وب سرورهای مجاور هدایت کنند، زمانی که مزرعه سرورهای وب از نظر جغرافیایی پراکنده است.

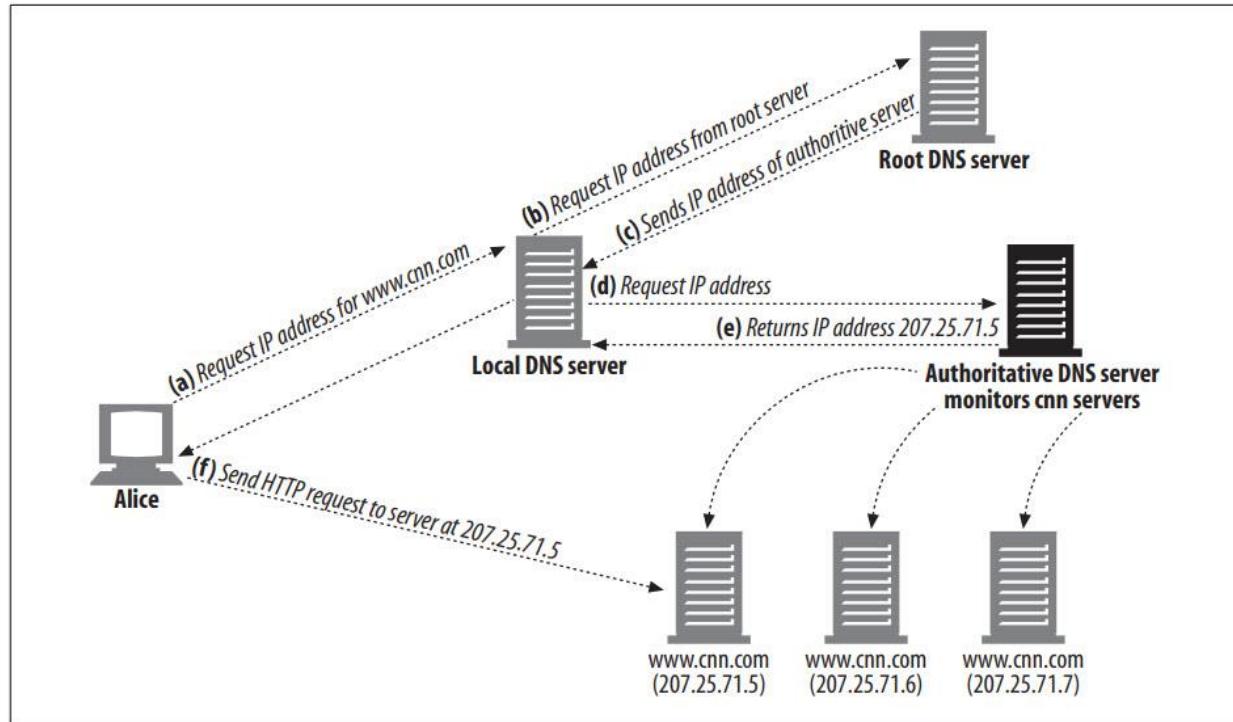
Fault-masking algorithms

سرورهای DNS می‌توانند سلامت شبکه را کنترل کنند و درخواست‌ها را به دور از وقفه‌های سرویس یا سایر خطاهای هدایت کنند.

به طور معمول، سرور DNS که الگوریتم‌های پیچیده ردیابی سرور را اجرا می‌کند، یک سرور معتبر است که تحت کنترل ارائه دهنده محتوا است (شکل زیر را ببینید).

⁵⁹ least loaded web servers





چندین سرویس میزبانی توزیع شده از این مدل تغییر مسیر DNS استفاده می‌کنند. یکی از اشکالات مدل برای سرویس‌هایی که به دنبال سرورهای مجاور می‌گردند این است که تنها اطلاعاتی که سرور DNS معتبر برای تصمیم‌گیری از آن استفاده می‌کند، آدرس IP سرور DNS محلی است، نه آدرس IP کلاینت.

Anycast Addressing

در آدرس‌دهی Anycast، چندین وب سرور پراکنده جغرافیایی دقیقاً آدرس IP یکسانی دارند و برای ارسال درخواست‌های کلاینت به سرور نزدیک به کلاینت، به قابلیت‌های مسیریابی «کوتاه‌ترین مسیر»⁶⁰ روترهای Backbone متکی هستند. یکی از راه‌هایی که این روش می‌تواند کار کند این است که هر وب سرور خود را به عنوان یک روت برای روت Backbone همسایه Advertise کند. وب سرور با استفاده از پروتکل ارتباطی روت با روت Backbone همسایه خود صحبت می‌کند. هنگامی که روت Backbone بسته‌هایی را دریافت می‌کند که به آدرس Anycast هستند، به دنبال نزدیک‌ترین «روت» می‌گردد که آن آدرس IP را می‌پذیرد. از آنجایی که سرور خود را به عنوان روت برای آن آدرس Backbone می‌کند، روت Backbone بسته را برای سرور ارسال می‌کند.

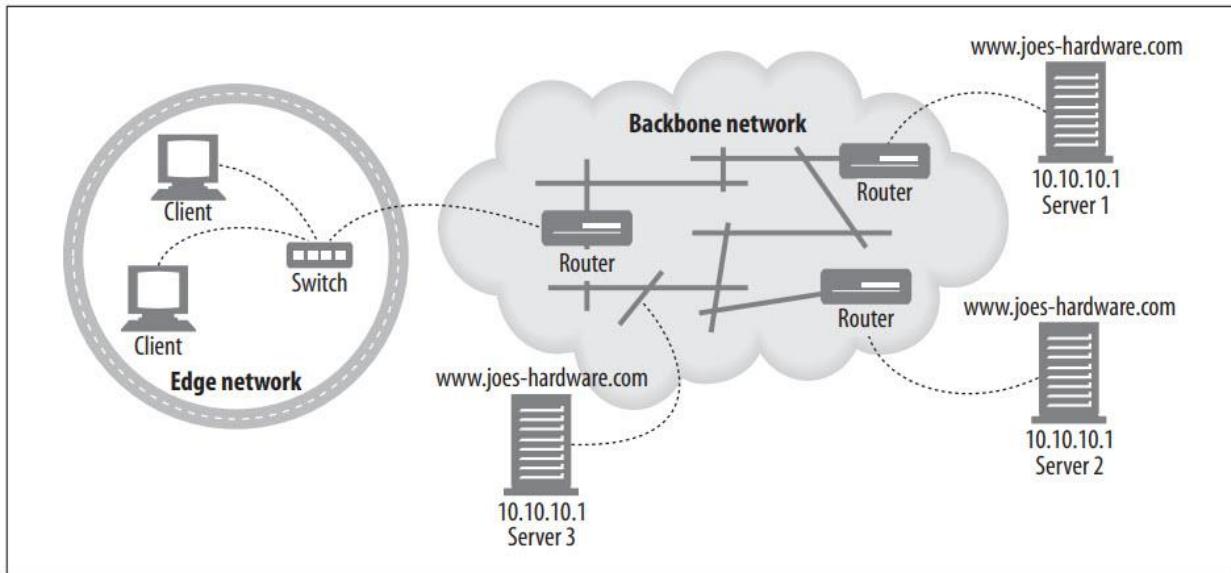
در شکل زیر، سه سرور با آدرس IP یکسان، ۱۰.۱۰.۱۰.۱ روبرو هستند. سرور لس آنجلس (LA) این آدرس را برای روت LA، سرور نیویورک (NY) همان آدرس را برای روت نیویورک و غیره Advertise

⁶⁰ shortest-path





می‌کند. سرورها با استفاده از پروتکل روتر با روترا ارتباط برقرار می‌کنند. روترا به طور خودکار درخواست‌های کلاینت را با هدف ۱۰.۱۰.۱۰.۱ به نزدیکترین سروری که آدرس را Advertise می‌کند هدایت می‌کنند. در شکل زیر، یک درخواست برای آدرس IP 10.10.10.1 به سرور ۳ هدایت می‌شود.



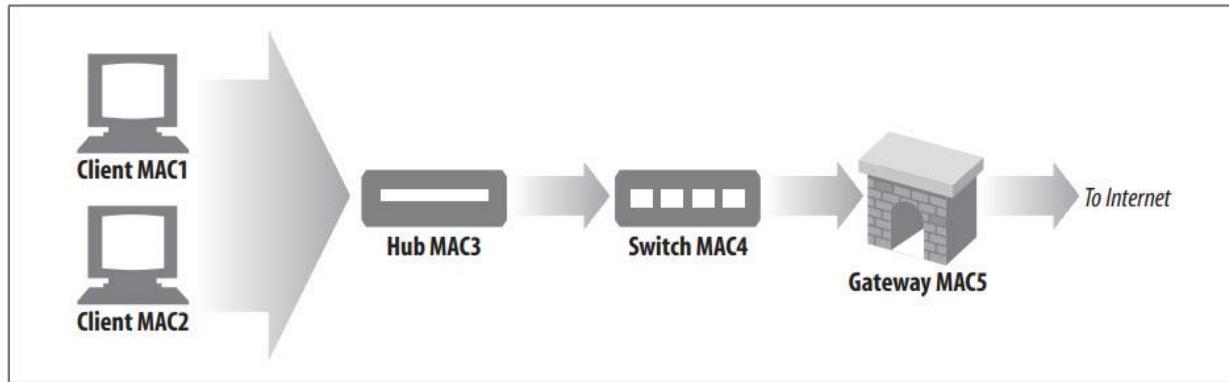
آدرس دهی Anycast هنوز یک تکنیک تجربی است. برای اینکه Anycast توزیع شده کار کند، سرورها باید به زبان روتر صحبت کنند و روترا باید بتوانند تضادهای احتمالی آدرس را مدیریت کنند، زیرا آدرس دهی اینترنتی اساساً یک سرور را برای یک آدرس فرض می‌کند. (اگر این کار به درستی انجام نشود، می‌تواند منجر به مشکلات جدی شود که به عنوان route leaks شناخته می‌شوند). Anycast توزیع شده یک فناوری در حال ظهرور است و ممکن است راه حلی برای ارائه دهنده‌گان محتوا باشد که شبکه‌های اصلی خود را کنترل می‌کنند.

IP MAC Forwarding

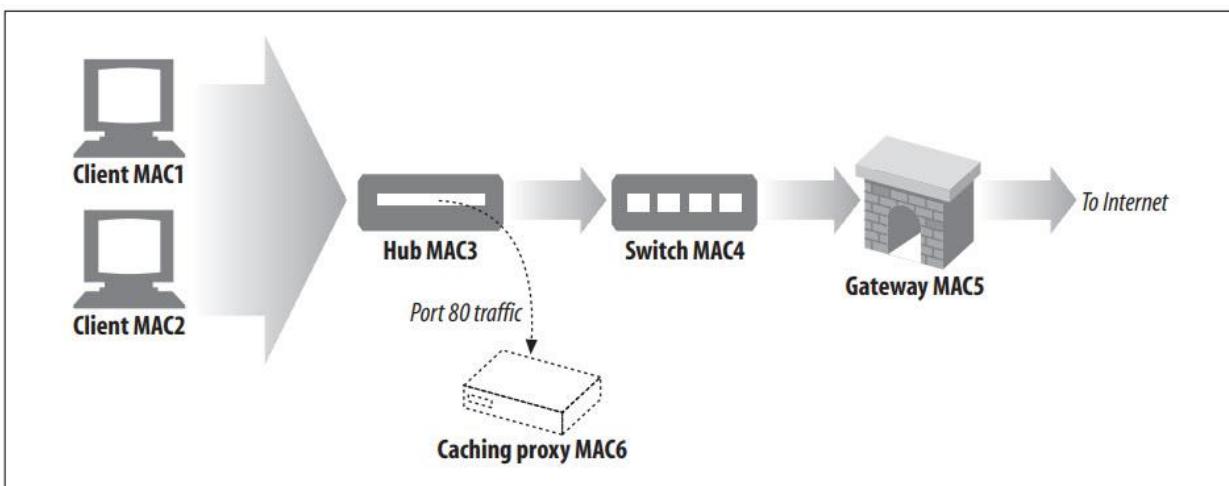
در شبکه‌های اترنت، پیام‌های HTTP در قالب بسته‌های داده آدرس دهی شده ارسال می‌شوند. هر بسته دارای یک آدرس لایه ۴ است که از آدرس IP مبدأ و مقصد و شماره پورت TCP تشکیل شده است. این آدرسی است که دستگاه‌های لایه ۴ به آن توجه می‌کنند. هر بسته همچنین دارای یک آدرس لایه ۲ به نام Media Access Control یا MAC است که دستگاه‌های لایه ۲ (معمولاً سوئیچ‌ها و هاب‌ها) به آن توجه می‌کنند. وظیفه دستگاه‌های لایه ۲ دریافت بسته‌هایی با آدرس‌های MAC ورودی خاص و ارسال آن‌ها به آدرس‌های MAC خروجی خاص است.

به عنوان مثال، در شکل زیر، سوئیچ طوری برنامه ریزی شده است که تمام ترافیک را از آدرس MAC3 به آدرس MAC4 ارسال کند.





سوئیچ لایه ۴ می‌تواند آدرس دهی لایه ۴ (آدرس‌های IP و شماره پورت‌های TCP) را بررسی کند و بر اساس این اطلاعات تصمیمات مسیریابی را اتخاذ کند. به عنوان مثال، یک سوئیچ لایه ۴ می‌تواند تمام ترافیک وب مقصد پورت ۸۰ را به یک پروکسی ارسال کند. در شکل زیر، سوئیچ طوری برنامه ریزی شده است که تمام ترافیک پورت ۸۰ را از MAC3 به MAC6 (یک حافظه پنهان پراکسی) ارسال کند. تمام ترافیک دیگر MAC3 به ۸۰ می‌رود.



به طور معمول، اگر محتوای HTTP درخواستی در Cache باشد و تازه باشد، Proxy Cache آن را ارائه می‌کند. در غیر این صورت، Proxy Cache یک درخواست HTTP را از طرف کلاینت به سرور مبدا برای محتوا ارسال می‌کند. سوئیچ درخواست‌های پورت ۸۰ را از پروکسی (MAC6) به دروازه اینترنت (MAC5) ارسال می‌کند.

سوئیچ‌های لایه ۴ که از ارسال MAC پشتیبانی می‌کنند معمولاً می‌توانند درخواست‌ها را به چندین Proxy Cache ارسال کنند و بار را بین آن‌ها متعادل کنند. به همین ترتیب، ترافیک HTTP نیز می‌تواند به سرورهای HTTP جایگزین هدایت شود.



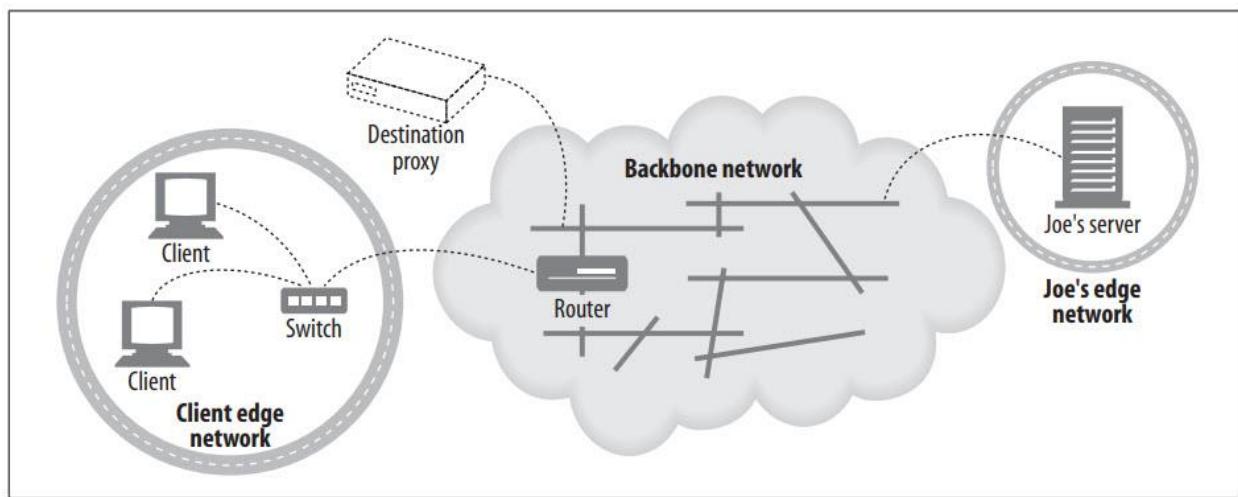


از آنجا که ارسال آدرس MAC فقط point-to-point است، سرور یا پروکسی باید یک Hop دورتر از سوئیچ قرار گیرد.

IP Address Forwarding

در انتقال آدرس IP، یک سوئیچ یا دستگاه لایه ۴، آدرسدهی TCP/IP را در بسته‌های ورودی بررسی می‌کند و بسته‌ها را بر اساس آن با تغییر آدرس IP مقصد، به جای آدرس MAC مقصد، مسیریابی می‌کند. یک مزیت نسبت به MAC Forwarding این است که سرور مقصد نیازی به فاصله یک Hop ندارد. فقط باید در بالادست سوئیچ قرار گیرد و مسیریابی معمول لایه ۳ اینترنت سرتاسر بسته را به مکان مناسب می‌رساند. به این نوع ارسال، NAT نیز گفته می‌شود.

با این حال یک نکته وجود دارد: تقارن مسیریابی^{۶۱}. سوئیچ که اتصال TCP ورودی را از کلاینت می‌پذیرد، آن اتصال را مدیریت می‌کند. سوئیچ باید پاسخ را به کلاینت در آن اتصال TCP ارسال کند. بنابراین، هر پاسخی از سرور مقصد یا پروکسی باید به سوییچ برگردد (شکل زیر را ببینید).



دو روش برای کنترل مسیر برگشت پاسخ عبارتند از:

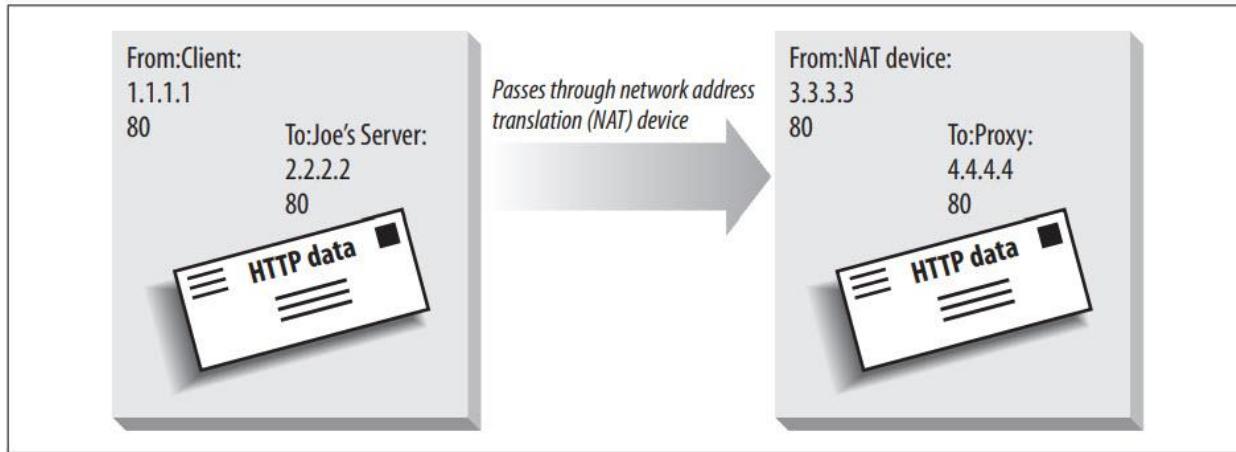
- آدرس IP منبع بسته را به آدرس IP سوئیچ تغییر دهید. به این ترتیب، صرف نظر از پیکربندی شبکه بین سوئیچ و سرور، بسته پاسخ به سوییچ می‌رود. این Full NAT نامیده می‌شود، جایی که دستگاه ارسال IP آدرس IP مقصد و مبدأ را ترجمه می‌کند. شکل زیر اثر Full NAT را بر روی دیتاگرام TCP/IP نشان می‌دهد. نتیجه این است که آدرس IP کلاینت برای وب سرور ناشناخته است، که ممکن است آن را برای احراز هویت بخواهد.

⁶¹ routing symmetry





- اگر آدرس IP منبع همچنان آدرس IP کلاینت باقی می‌ماند، مطمئن شوید (از دیدگاه سخت افزاری) هیچ مسیری مستقیماً از سرور به کلاینت وجود ندارد (با دور زدن سوئیچ). این گاهی اوقات Half NAT نامیده می‌شود. مزیت در اینجا این است که سرور آدرس IP کلاینت را دریافت می‌کند، اما نقطه ضعف آن نیاز به کنترل کل شبکه بین کلاینت و سرور است.



Network Element Control Protocol

NECP یا Network Element Control Protocol به عناصر شبکه^{۶۲} - دستگاههایی مانند روتراها و سوئیچهایی که بستههای IP را ارسال می‌کنند - اجازه می‌دهد با عناصر سرور^{۶۳} صحبت کنند - دستگاههایی مانند وب سرورها و کش‌های پراکسی که درخواستهای لایه Application را ارائه می‌کنند. NECP صریحاً از NE Load-Balancing پشتیبانی نمی‌کند. این فقط راهی را برای یک SE ارائه می‌دهد تا اطلاعات Load Balancing را ارسال کند تا NE بتواند تعادل را به دلخواه بارگذاری نماید. مانند NECP، WCCP، NE Load-Balancing برای ارسال بسته‌ها ارائه می‌دهد: .NAT encapsulation، MAC Forwarding و GRE encapsulation.

NECP از ایده استثنای^{۶۴} پشتیبانی می‌کند. SE می‌تواند تصمیم بگیرد که نمی‌تواند آدرس‌های IP منبع خاصی را سرویس دهد و آن آدرس‌ها را به NE ارسال کند. سپس NE می‌تواند درخواست‌ها را از آن آدرس‌های IP به سرور مبدأ ارسال کند.

Messages

پیام‌های NECP در جدول زیر توضیح داده شده است.

⁶² network elements

⁶³ server elements

⁶⁴ exceptions





Message	Who sends it	Meaning
NECP_NOOP		No operation—do nothing.
NECP_INIT	SE	SE initiates communication with NE. SE sends this message to NE after opening TCP connection with NE. SE must know which NE port to connect to.
NECP_INIT_ACK	NE	Acknowledges NECP_INIT.
NECP_KEEPALIVE	NE or SE	Asks if peer is alive.
NECP_KEEPALIVE_ACK	NE or SE	Answers keep-alive message.
NECP_START	SE	SE says "I am here and ready to accept network traffic." Can specify a port.
NECP_START_ACK	NE	Acknowledges NECP_START.
NECP_STOP	SE	SE tells NE "stop sending me traffic."
NECP_STOP_ACK	NE	NE acknowledges stop.
NECP_EXCEPTION_ADD	SE	SE says to add one or more exceptions to NE's list. Exceptions can be based on source IP, destination IP, protocol (above IP), or port.
NECP_EXCEPTION_ADD_ACK	NE	Confirms EXCEPTION_ADD.
NECP_EXCEPTION_DEL	SE	Asks NE to delete one or more exceptions from its list.
NECP_EXCEPTION_DEL_ACK	NE	Confirms EXCEPTION_DEL.
NECP_EXCEPTION_RESET	SE	Asks NE to delete entire exception list.
NECP_EXCEPTION_RESET_ACK	NE	Confirms EXCEPTION_RESET.
NECP_EXCEPTION_QUERY	SE	Queries NE's entire exception list.
NECP_EXCEPTION_RESP	NE	Responds to exception query.

Proxy Redirection Methods

تا اینجا در مورد روش‌های کلی تغییر مسیر صحبت کردیم. همچنین ممکن است نیاز به دسترسی به محتوا از طریق پراکسی‌های مختلف باشد (احتمالاً به دلایل امنیتی)، یا ممکن است یک Proxy Cache در شبکه وجود داشته باشد که کلاینت باید از آن استفاده کند (زیرا احتمالاً بازیابی محتوای Cache بسیار سریعتر از رفتن مستقیم به سرور اصلی خواهد بود).

اما کلاینت‌هایی مانند مرورگرهای وب چگونه می‌دانند که به یک پروکسی بروند؟ سه راه برای تعیین این وجود دارد: با پیکربندی صریح مرورگر، با پیکربندی خودکار پویا و با رهگیری شفاف^{۶۵}؛ در این بخش به این سه تکنیک خواهیم پرداخت.

⁶⁵ transparent interception





یک پروکسی به نوبه خود می‌تواند درخواست‌های کلاینت را به یک پروکسی دیگر هدایت کند. به عنوان مثال، یک Cache که محتوا را در Cache خود ندارد، ممکن است انتخاب کند که کلاینت را به دیگری هدایت کند. از آنجایی که این منجر به پاسخ از مکانی متفاوت از مکانی می‌شود که کلاینت منبع را از آن درخواست کرده است، ما همچنین چندین پروتکل مورد استفاده برای تغییر مسیر Proxy-Cache همتا را مورد CARP Cache Array Routing Protocol، ICP Internet Cache Protocol یا .HTCP Hyper Text Caching Protocol و

Explicit Browser Configuration

اغلب مرورگرها را می‌توان طوری پیکربندی کرد که برای محتوا با یک سرور پراکسی تماس بگیرند - یک منوی کشویی وجود دارد که کاربر می‌تواند نام پروکسی یا آدرس IP و شماره پورت را وارد کند. سپس مرورگر برای همه درخواست‌ها با پروکسی تماس می‌گیرد. برخی از ارائه دهندگان خدمات به جای اتکا به کاربران برای پیکربندی صحیح مرورگرهای خود برای استفاده از پراکسی، از کاربران می‌خواهند مرورگرهای از پیش پیکربندی شده را دانلود کنند. این مرورگرها آدرس پروکسی مورد نظر را می‌دانند.

پیکربندی صریح مرورگر دو عیب اصلی دارد:

- مرورگرهایی که برای استفاده از پراکسی‌ها پیکربندی شده‌اند، حتی اگر پراکسی پاسخ ندهد، با سرور اصلی تماس نمی‌گیرند. اگر پراکسی خاموش باشد یا اگر مرورگر به درستی پیکربندی نشده باشد، کاربر با مشکلات اتصال مواجه می‌شود.
- ایجاد تغییرات در معماری شبکه و انتشار آن تغییرات به همه کاربران نهایی دشوار است. اگر یک ارائه دهنده خدمات بخواهد پراکسی‌های بیشتری اضافه کند یا برخی از آن‌ها را از سرویس خارج کند، کاربران مرورگر باید تنظیمات پراکسی خود را تغییر دهند.

Proxy Auto-configuration

پیکربندی صریح مرورگرها برای تماس با پراکسی‌های خاص می‌تواند تغییرات در معماری شبکه را محدود کند، زیرا به کاربران بستگی دارد که مداخله کنند و مرورگرهای خود را دوباره پیکربندی کنند. یک روش پیکربندی خودکار که به مرورگرها اجازه می‌دهد تا به صورت پویا خود را برای تماس با سرور پراکسی صحیح پیکربندی کنند، این مشکل را حل می‌کند. چنین روش شناسی وجود دارد. پروتکل Proxy Auto-configuration یا PAC نامیده می‌شود. PAC توسط Netscape تعريف شده است و توسط مرورگرهای Microsoft Internet Explorer و Navigator





ایده اصلی پشت PAC این است که مرورگرها یک فایل خاص به نام فایل PAC را بازیابی کنند که پروکسی را برای تماس با هر URL مشخص می‌کند. مرورگر باید برای تماس با سرور خاصی برای فایل PAC پیکربندی شود. سپس مرورگر فایل PAC را هر بار که مجدداً راه اندازی می‌شود واکشی می‌کند.

فایل PAC یک فایل جاوا اسکریپت است که باید عملکرد زیر را مشخص کند:

```
function FindProxyForURL(url, host)
```

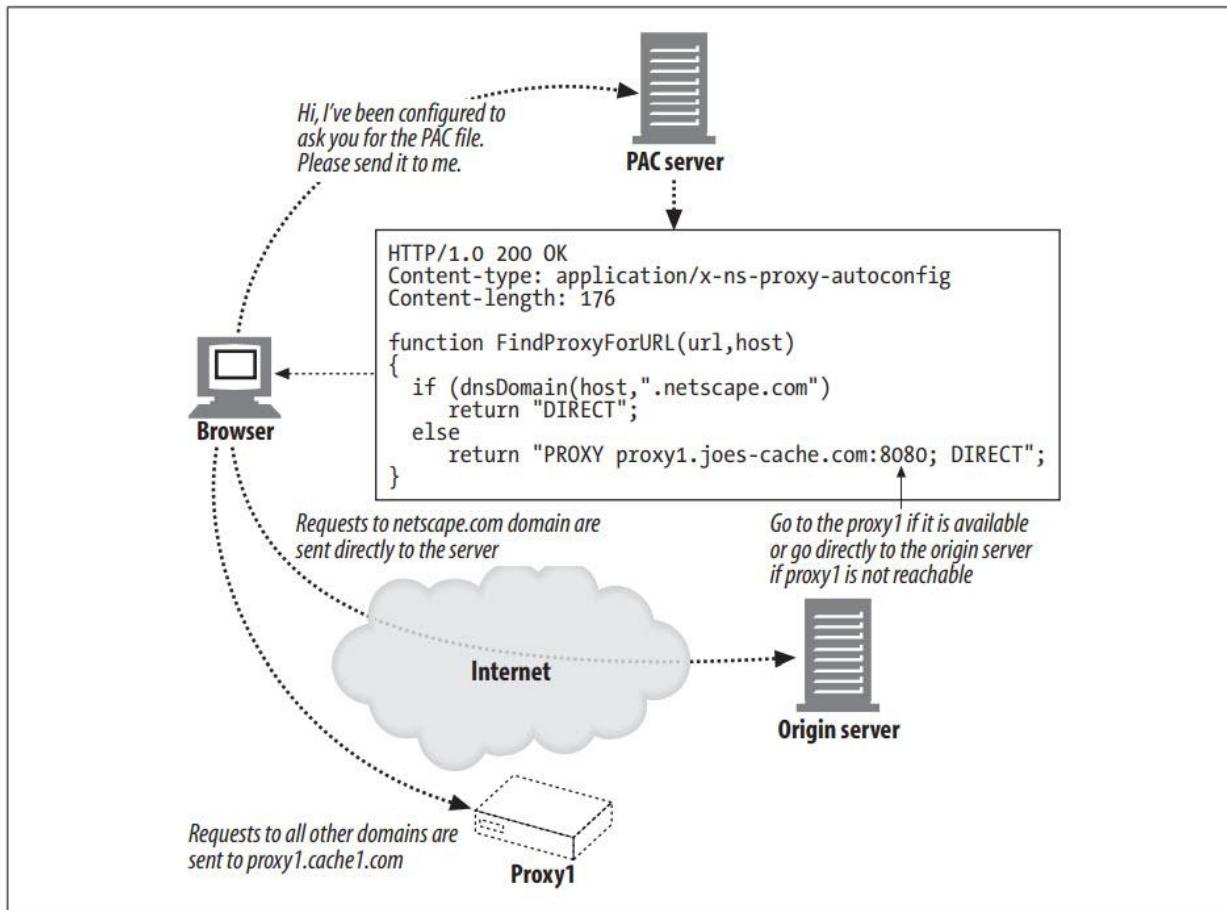
مرورگرها این تابع را برای هر URL درخواستی به صورت زیر فراخوانی می‌کنند:

```
return_value = FindProxyForURL(url_of_request, host_in_url);
```

که در آن مقدار بازگشتی رشته‌ای است که مشخص می‌کند مرورگر کجا باید این URL را درخواست کند. مقدار بازگشتی می‌تواند فهرستی از نامهای پراکسی‌هایی باشد که باید با آن‌ها تماس بگیرید (به عنوان مثال، "PROXY" یا "DIRECT") یا رشته "proxy1.domain.com; PROXY proxy2.domain.com" مرورگر باید مستقیماً به سرور مبدأ برود و هر پراکسی را دور بزند.

توالی عملیاتی که درخواست و پاسخ به درخواست مرورگر برای فایل PAC را نشان می‌دهد در شکل زیر نشان داده شده است. در این مثال، سرور یک فایل PAC را با یک برنامه جاوا اسکریپت پس می‌فرستد. برنامه جاوا اسکریپت تابعی به نام «FindProxyForURL» دارد که به مرورگر می‌گوید اگر میزبان در URL درخواستی در «proxy1.joes-cache.com» است، مستقیماً با سرور اصلی تماس بگیرد و به «netscape.com» برود. برای تمام درخواست‌های دیگر مرورگر این تابع را برای هر URLی که درخواست می‌کند فراخوانی می‌کند و با توجه به نتایجی که توسط تابع برمی‌گردد به آن متصل می‌شود.





پروتکل PAC بسیار قدرتمند است: برنامه جاوا اسکریپت می‌تواند از مرورگر بخواهد که یک پروکسی را بر اساس هر یک از تعدادی از پارامترهای مربوط به نام میزبان، مانند آدرس DNS و زیرشبکه، و حتی روز هفته یا ساعت روز، انتخاب کند. PAC به مرورگرها اجازه می‌دهد تا با تغییر در معماری شبکه، به طور خودکار با پروکسی مناسب تماس بگیرند، تا زمانی که فایل PAC در سرور بهروزرسانی می‌شود تا تغییرات مکان‌های پراکسی را منعکس کند. اشکال اصلی PAC این است که مرورگر باید پیکربندی شود تا بداند فایل PAC را از کدام سرور واکشی کند، بنابراین یک سیستم پیکربندی کاملاً خودکار نیست. WPAD، که در بخش بعدی مورد بحث قرار گرفت، به این مشکل می‌پردازد.

PAC، مانند مرورگرهای از پیش تنظیم شده، امروزه توسط برخی از ISP‌های اصلی استفاده می‌شود.

Web Proxy Autodiscovery Protocol

WPAD یا Web Proxy Autodiscovery Protocol با هدف ارائه راهی برای مرورگرهای وب برای یافتن و استفاده از پراکسی‌های مجاور، بدون نیاز به کاربر نهایی برای پیکربندی دستی تنظیمات پراکسی و بدون تکیه بر رهگیری ترافیک شفاف است. مشکل کلی تعریف یک WPAD به دلیل وجود





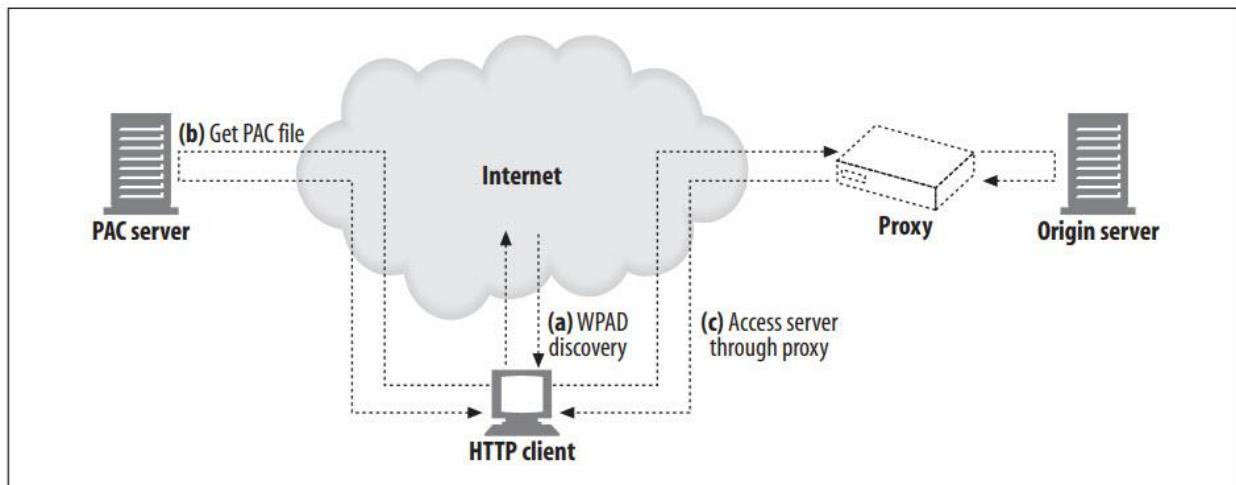
پروتکل‌های کشف بسیاری برای انتخاب و تفاوت در تنظیمات استفاده از پروکسی در مرورگرهای مختلف پیچیده است.

این بخش شامل یک نسخه کوتاه شده و کمی سازماندهی شده از پیش‌نویس اینترنت WPAD است. این پیش‌نویس در حال حاضر به عنوان بخشی از گروه کاری واسطه‌های وب IETF در حال توسعه است.

PAC file autodiscovery

به مشتریان HTTP WPAD امکان می‌دهد تا یک فایل PAC را پیدا کنند و از فایل PAC برای کشف نام یک سرور پراکسی مناسب استفاده کنند. WPAD مستقیماً نام سرور پراکسی را تعیین نمی‌کند، زیرا این امر باعث دور زدن قابلیت‌های اضافی ارائه شده توسط فایل‌های PAC (تعادل بار، درخواست مسیریابی به مجموعه‌ای از سرورها، خطای خودکار برای پشتیبان‌گیری از سرورهای پراکسی و غیره) می‌شود.

همانطور که در شکل زیر نشان داده است، پروتکل WPAD یک URL فایل PAC را کشف می‌کند که به عنوان URL پیکربندی (curl) نیز شناخته می‌شود. فایل PAC یک برنامه جاوا اسکریپت را اجرا می‌کند که آدرس یک سرور پراکسی مناسب را برمی‌گرداند.



یک سرویس گیرنده HTTP که پروتکل WPAD را پیاده سازی می‌کند:

- از WPAD برای یافتن فایل PAC curl استفاده می‌کند.
- فایل PAC (با نام مستعار فایل پیکربندی یا CFILE) مربوط به curl را واکشی می‌کند.
- فایل PAC را برای تعیین سرور پراکسی اجرا می‌کند.
- درخواست‌های HTTP را به سرور پراکسی ارسال شده توسط فایل PAC ارسال می‌کند.





WPAD algorithm

WPAD از یک سری تکنیک‌های کشف منبع برای تعیین PAC فایل CURL مناسب استفاده می‌کند. تکنیک‌های کشف چندگانه مشخص شده است، زیرا همه سازمان‌ها نمی‌توانند از همه تکنیک‌ها استفاده کنند. کلاینت‌های WPAD هر تکنیک را یک به یک امتحان می‌کنند تا زمانی که موفق به بدست آوردن CURL شوند.

مشخصات WPAD فعلی تکنیک‌های زیر را به ترتیب تعریف می‌کند:

- DHCP (Dynamic Host Discovery Protocol)
- SLP (Service Location Protocol)
- DNS well-known hostnames
- DNS SRV records
- DNS service URLs in TXT records

از بین این پنج مکانیسم، تنها تکنیک‌های Hostname و DNS معروف DHCP برای کلاینت‌های WPAD مورد نیاز است.

سرویس گیرنده WPAD یک سری درخواست‌های کشف منبع را با استفاده از مکانیسم‌های کشف ذکر شده در بالا به ترتیب ارسال می‌کند. کلاینت‌ها فقط مکانیسم‌هایی را امتحان می‌کنند که از آن‌ها پشتیبانی می‌کنند. هر زمان که یک تلاش برای کشف موفقیت آمیز باشد، کلاینت از اطلاعات به دست آمده برای ایجاد یک PAC CURL استفاده می‌کند.

اگر یک فایل PAC با موفقیت در آن CURL بازیابی شود، فرآیند تکمیل می‌شود. در غیر این صورت، کلاینت از همان جایی که در سری از پیش تعریف شده درخواست‌های کشف منبع متوقف شد، از سر می‌گیرد. اگر پس از امتحان همه مکانیسم‌های کشف، هیچ فایل PAC بازیابی نشد، پروتکل WPAD از کار می‌افتد و کلاینت طوری پیکربندی می‌شود که از هیچ سرور پراکسی استفاده نکند.

کلاینت ابتدا DHCP و سپس SLP را امتحان می‌کند. اگر هیچ فایل PAC بازیابی نشود، کلاینت به مکانیسم‌های مبتنی بر DNS می‌رود.

کلاینت چندین بار از طریق DNS SRV، DNS TXT و روش‌های ضبط DNS QNAME بار، پرس و جوی DNS QNAME کمتر و کمتر مشخص می‌شود. به این ترتیب، کلاینت می‌تواند خاص‌ترین اطلاعات پیکربندی ممکن را پیدا کند، اما همچنان می‌تواند به اطلاعات کمتر خاص بازگردد. هر جستجوی DNS دارای پیشوند QNAME با "wpad" است تا نوع منبع درخواستی را نشان دهد.





کلاینت با نام میزبان johns-desktop.development.foo.com را در نظر بگیرید. این دنباله‌ای از تلاش‌های کشف است که یک کلاینت WPAD کامل انجام می‌دهد:

- DHCP
- SLP
- DNS A lookup on “QNAME=wpad.development.foo.com”
- DNS SRV lookup on “QNAME=wpad.development.foo.com”
- DNS TXT lookup on “QNAME=wpad.development.foo.com”
- DNS A lookup on “QNAME=wpad.foo.com”
- DNS SRV lookup on “QNAME=wpad.foo.com”
- DNS TXT lookup on “QNAME=wpad.foo.com”

برای دریافت شبه کد دقیق که تمام توالی عملیات را نشان می‌دهد، به مشخصات WPAD مراجعه کنید. بخش‌های زیر دو مکانیسم مورد نیاز، DHCP و DNS A lookup را مورد بحث قرار می‌دهند. برای جزئیات بیشتر در مورد یادآوری روش‌های کشف CURL، به مشخصات WPAD مراجعه کنید.

CURL discovery using DHCP

برای اینکه این مکانیسم کار کند، CURL ها باید در سرورهای DHCP ذخیره شوند که کلاینت‌های WPAD می‌توانند پرس و جو کنند. کلاینت WPAD با ارسال یک درخواست DHCP به سرور CURL را به دست می‌آورد. CURL در کد گزینه 252 DHCP موجود است (اگر سرور DHCP با این اطلاعات پیکربندی شده باشد). تمام پیاده سازی‌های سرویس گیرنده WPAD برای پشتیبانی از DHCP مورد نیاز است. پروتکل DHCP در RFC 2131 به تفصیل آمده است. برای لیستی از گزینه‌های موجود DHCP به RFC 2132 مراجعه کنید.

اگر سرویس گیرنده WPAD قبلًا در طول Initialization خود، پرس و جوهای DHCP را انجام داده باشد، ممکن است سرور DHCP قبلًا آن مقدار را ارائه کرده باشد. اگر مقدار از طریق یک کلاینت OS API در دسترس نباشد، سرویس گیرنده یک پیام DHCPINFORM برای پرس و جو از سرور DHCP برای بدست آوردن مقدار ارسال می‌کند.

کد گزینه 252 DHCP برای WPAD از نوع STRING است و اندازه دلخواه دارد. این رشته حاوی یک URL است که به یک فایل PAC مناسب اشاره می‌کند. مثلا:

"<http://server.domain/proxyconfig.pac>"





DNS A record lookup

برای اینکه این مکانیسم کار کند، آدرس‌های IP سرورهای پراکسی مناسب باید در سرورهای DNS ذخیره شوند که کلاینت‌های WPAD می‌توانند پرس و جو کنند. کلاینت WPAD CURL را با ارسال یک جستجوی رکورد به یک سرور DNS به دست می‌آورد. نتیجه جستجوی موفق شامل یک آدرس IP برای سرور پروکسی مناسب است.

Implementation سرویس گیرنده WPAD برای پشتیبانی از این مکانیسم مورد نیاز است. این باید ساده باشد، زیرا فقط جستجوی اولیه DNS برای رکوردهای A مورد نیاز است. RFC 2219 را برای توضیح استفاده از "well known alias" DNS برای کشف منابع ببینید. برای WPAD، این مشخصات از "wpad" برای کشف خودکار پروکسی وب استفاده می‌کند.

کلاینت جستجوی DNS زیر را انجام می‌دهد:

```
QNAME=wpad.TGTDOM., QCLASS=IN, QTYPE=A
```

جستجوی موفق شامل یک آدرس IP است که سرویس گیرنده WPAD از آن CURL را می‌سازد.

Retrieving the PAC file

هنگامی که یک Candidate CURL ایجاد می‌شود، کلاینت WPAD معمولاً یک درخواست GET را به CFILER ارسال می‌کند. هنگام درخواست، کلاینت‌های WPAD ملزم به ارسال هدرهای Accept با اطلاعات فرمت مناسب هستند که قادر به مدیریت آن هستند. مثلاً:

```
Accept: application/x-nsp-proxy-autoconfig
```

علاوه بر این، اگر CURL منجر به تغییر مسیر شود، کلاینت‌ها ملزم به دنبال کردن تغییر مسیر به مقصد نهایی هستند.

When to execute WPAD

فرآیند کشف خودکار پروکسی وب باید حداقل به اندازه یکی از موارد زیر انجام شود:

- هنگام راه اندازی سرویس گیرنده وب - WPAD فقط برای شروع اولین نمونه انجام می‌شود. نمونه‌های بعدی تنظیمات را به ارث می‌برند.
- هر زمان که نشانه‌ای از پشت‌هش شبکه وجود دارد که نشانی IP میزبان کلاینت تغییر کرده است.





یک سرویس گیرنده وب می‌تواند از هر یک از گزینه‌ها استفاده کند، بسته به اینکه چه چیزی در محیط آن منطقی است. علاوه بر این، کلاینت باید یک چرخه کشف را پس از انقضای یک فایل PAC دانلود شده قبلی مطابق با انقضای HTTP انجام دهد. مهم است که کلاینت از زمان‌بندی‌ها پیروی کند و پس از انقضای فایل PAC، فرآیند WPAD را دوباره اجرا کند.

در صورت عدم موفقیت در پروکسی پیکربندی شده فعلی، در صورتی که فایل PAC جایگزینی ارائه نکند، به صورت اختیاری، کلاینت همچنین ممکن است اجرای مجدد فرآیند WPAD را اجرا کند.

هر زمان که کلاینت تصمیم گرفت فایل PAC فعلی را باطل کند، باید کل پروتکل WPAD را مجدداً اجرا کند تا مطمئن شود که CURL فعلی صحیح را کشف می‌کند. به طور خاص، هیچ شرطی در پروتکل برای انجام یک واکشی شرطی If-Modified-Since از فایل PAC وجود ندارد.

در طول پخش پروتکل WPAD و/یا ارتباطات چندپیخشی ممکن است تعدادی رفت و برگشت شبکه مورد نیاز باشد. پروتکل WPAD باید با سرعت بیشتر از آنچه مشخص شده است (مانند بازیابی هر URL) فراخوانی شود.

WPAD spoofing

پیاده سازی IE 5 WPAD به کلاینت‌های وب امکان می‌دهد تنظیمات پراکسی را به طور خودکار و بدون دخالت کاربر شناسایی کنند. الگوریتم مورد استفاده توسط WPAD نام میزبان "wpad" را به نام دامنه کاملاً واحد شرایط اضافه می‌کند و به تدریج زیر دامنه‌ها را حذف می‌کند تا زمانی که سرور WPAD را پیدا کند که نام میزبان a.b.microsoft.com را پاسخ می‌دهد یا به دامنه سطح سوم برسد. به عنوان مثال، کلاینت‌های وب در دامنه wpad.microsoft.com از wpad.b.microsoft.com و سپس wpad.a.b.microsoft پرس و جو می‌کنند.

این یک حفره امنیتی را آشکار کرد، زیرا در استفاده بین‌المللی (و برخی پیکربندی‌های دیگر)، ممکن است دامنه سطح سوم قابل اعتماد نباشد. یک کاربر مخرب می‌تواند یک سرور WPAD راه اندازی کند و دستورات پیکربندی پروکسی را به انتخاب خود ارائه دهد. نسخه‌های بعدی IE (۱۰.۰ به بعد) مشکل را برطرف کردند.

Timeouts

از چندین سطح کشف می‌گزارد و کلاینت‌ها باید مطمئن شوند که هر مرحله محدود به زمان است. در صورت امکان، محدود کردن هر فاز به ۱۰ ثانیه معقول تلقی می‌شود، اما پیاده کننده‌ها ممکن است مقدار متفاوتی را انتخاب کنند که با ویژگی‌های شبکه آن‌ها مناسب‌تر است. برای مثال، پیاده‌سازی





دستگاه، که از طریق یک شبکه بی‌سیم کار می‌کند، ممکن است از زمان‌بندی بسیار بزرگ‌تری برای محاسبه پهنه‌ای باند کم یا تأخیر بالا استفاده کند.

Administrator considerations

مدیران باید حداقل یکی از روش‌های جستجوی رکورد DNS A یا DHCP را در محیط‌های خود پیکربندی کنند، زیرا تنها دو روشی هستند که همه کلاینت‌های سازگار باید پیاده‌سازی کنند. فراتر از آن، پیکربندی برای پشتیبانی از مکانیسم‌های اولیه در ترتیب جستجو، زمان راهاندازی کلاینت را بهبود می‌بخشد.

یکی از انگیزه‌های اصلی این ساختار پروتکل، پشتیبانی از مکان کلاینت سرورهای پراکسی مجاور بود. در بسیاری از محیط‌ها، چندین سرور پروکسی (Gateway, Workgroup, ISP, Backbone) وجود دارد.

تعدادی از نقاط ممکن وجود دارد که در چارچوب WPAD می‌توان تصمیم‌های «نزدیک»^{۶۶} گرفت:

- سرورهای DHCP برای زیرشبکه‌های مختلف می‌توانند پاسخ‌های متفاوتی را برگردانند. آن‌ها همچنین می‌توانند تصمیمات را بر اساس فیلد کلاینت cipaddr یا گزینه شناسه کلاینت قرار دهند.
- سرورهای DNS را می‌توان به گونه‌ای پیکربندی کرد که سوابق منابع مختلف (RRs/A/TXT) را برای QNAMEs wpad.marketing.bigcorp.com و wpad.development.bigcorp.com پسوندهای دامنه مختلف (به عنوان مثال، wpad.marketing.bigcorp.com برگرداند).
- وب سروری که درخواست CURL را مدیریت می‌کند می‌تواند بر اساس هدر User-Agent، هدر Accept، آدرس IP کلاینت /subnet/name host، توزیع توپولوژیکی سرورهای پراکسی مجاور، و غیره تصمیم گیری کند. این می‌تواند در داخل یک فایل اجرایی CGI ایجاد شده برای مدیریت CURL رخ دهد. همانطور که قبل ذکر شد، حتی می‌تواند یک سرور پروکسی باشد که درخواست‌های CURL را مدیریت می‌کند و این تصمیمات را می‌گیرد.
- فایل PAC ممکن است به اندازه کافی رسا باشد که از میان مجموعه‌ای از گزینه‌ها در زمان اجرا روی کلاینت انتخاب شود. CARP بر اساس این فرض برای آرایه‌ای از Cache ها است. غیر قابل تصور نیست که فایل PAC بتواند برخی از معیارهای فاصله شبکه را با مجموعه‌ای از سرورهای پراکسی Candidate محاسبه کند و سپس "نزدیک ترین" یا "پاسخگوترین" سرور را انتخاب کند.

^{۶۶} nearness





Cache Redirection Methods

ما در مورد تکنیکهایی برای هدایت ترافیک به سرورهای عمومی و تکنیکهای تخصصی برای انتقال ترافیک به پراکسی‌ها و Gateway‌ها صحبت کردایم. این بخش پایانی برخی از تکنیکهای پیچیده‌تر تغییر مسیر مورد استفاده برای ذخیره سازی سرورهای پراکسی را توضیح می‌دهد. این تکنیکها پیچیده‌تر از پروتکل‌های مورد بحث قبلی هستند، زیرا سعی می‌کنند قابل اعتماد، با کارایی بالا و آگاه به محتوا باشند - درخواست‌ها را به مکان‌هایی که احتمالاً دارای محتوای خاصی هستند ارسال می‌کنند.

WCCP Redirection

سیسکو Web Cache Coordination Protocol را توسعه داد تا روتراها را قادر سازد تا ترافیک وب را به Proxy Cache‌ها هدایت کنند. WCCP ارتباطات بین روتراها و Cache‌ها را کنترل می‌کند به طوری که روتراها می‌توانند Cache‌ها را تأیید کنند (اطمینان حاصل شود که در حال اجرا هستند)، تعادل بار بین Cache‌ها را بارگذاری کنند و انواع خاصی از ترافیک را به Cache‌های خاص ارسال کنند. WCCP نسخه ۲ (WCCP2) یک Open Protocol است. ما در اینجا WCCP2 را مورد بحث قرار خواهیم داد.

How WCCP redirection works

در اینجا یک مرور مختصر از نحوه عملکرد تغییر مسیر WCCP برای HTTP ارائه شده است (WCCP پروتکل های دیگر را به طور مشابه تغییر مسیر می‌دهد):

- با شبکه‌ای شروع کنید که حاوی روتراها و Cache‌های دارای WCCP فعال است که می‌توانند با یکدیگر ارتباط برقرار کنند.
- مجموعه‌ای از روتراها و Cache‌های هدف آنها یک گروه سرویس WCCP را تشکیل می‌دهند. پیکربندی گروه سرویس مشخص می‌کند که چه ترافیکی به کجا ارسال می‌شود، چگونه ترافیک ارسال شده و چگونه بار باید بین Cache‌های گروه سرویس متعدد شود.
- اگر گروه سرویس برای تغییر مسیر ترافیک HTTP پیکربندی شده باشد، روتراهای گروه سرویس درخواست‌های HTTP را به Cache در گروه سرویس ارسال می‌کنند.
- هنگامی که یک درخواست HTTP به روترا در گروه سرویس می‌رسد، روترا یکی از Cache‌های گروه سرویس را برای ارائه درخواست انتخاب می‌کند (بر اساس هش آدرس IP درخواست یا طرح جفت شدن مجموعه .(mask/value).
- روترا بسته‌های درخواستی را با کپسوله کردن بسته‌ها با آدرس Cache IP یا با انتقال IP MAC به Cache ارسال می‌کند.



IP MAC Cache یا با انتقال IP یا با کپسوله کردن بسته‌ها با آدرس Cache IP





- اگر Cache نتواند درخواست را ارائه کند، بسته‌ها برای ارسال عادی به روتر بازگردانده می‌شوند.
- اعضای گروه سرویس پیام‌های Heartbeat را با یکدیگر رد و بدل می‌کنند و پیوسته در دسترس بودن یکدیگر را تأیید می‌کنند.

WCCP2 messages

چهار پیام WCCP2 وجود دارد که در جدول زیر توضیح داده شده است.

Message name	Who sends it	Information carried
WCCP2_HERE_I_AM	Cache to router	These messages tell routers that caches are available to receive traffic. The messages contain all of the cache's service group information. As soon as a cache joins a service group, it sends these messages to all routers in the group. These messages negotiate with routers sending WCCP2_I_SEE_YOU messages.
WCCP2_I_SEE_YOU	Router to cache	These messages respond to WCCP2_HERE_I_AM messages. They are used to negotiate the packet forwarding method, assignment method (who is the designated cache), packet return method, and security.
WCCP2_REDIRECT_ASSIGN	Designated cache to router	These messages make assignments for load balancing; they send bucket information for hash table load balancing or mask/value set pair information for mask/value load balancing.
WCCP2_REMOVAL_QUERY	Router to cache that has not sent WCCP2_HERE_I_AM messages for $2.5 \times \text{HERE_I_AM_T}$ seconds	If a router does not receive WCCP2_HERE_I_AM messages regularly, the router sends this message to see if the cache should be removed from the service group. The proper response from a cache is three identical WCCP2_HERE_I_AM messages, separated by $\text{HERE_I_AM_T}/10$ seconds.





The WCCP2_HERE_I_AM message format is:

- WCCP Message Header
- Security Info Component
- Service Info Component
- Web-cache Identity Info Component
- Web-cache View Info Component
- Capability Info Component (optional)
- Command Extension Component (optional)

The WCCP2_I_SEE_YOU message format is:

- WCCP Message Header
- Security Info Component
- Service Info Component
- Router Identity Info Component
- Router View Info Component
- Capability Info Component (optional)
- Command Extension Component (optional)

The WCCP2_REDIRECT_ASSIGN message format is:

- WCCP Message Header
- Security Info Component
- Service Info Component
- Assignment Info Component, or Alternate Assignment Component

The WCCP2_REMOVAL_QUERY message format is:

- WCCP Message Header
- Security Info Component
- Service Info Component
- Router Query Info Component

Message components

هر پیام WCCP2 از یک هدر و اجزا تشکیل شده است. اطلاعات هدر WCCP شامل نوع پیام (اینجا هستم، شما را می بینم، پرس و جوی تکلیف یا حذف)، نسخه WCCP و طول پیام (بدون در نظر گرفتن طول هدر) است. اجزاء هر کدام با یک هدر چهار هشتگانه شروع می شوند که نوع و طول مؤلفه را توصیف می کند. طول کامپوننت شامل طول هدر جزء نمی شود. اجزای پیام در جدول زیر توضیح داده شده است.



Component	Description
Security Info	Contains the security option and security implementation. The security option can be: WCCP2_NO_SECURITY (0) WCCP2_MD5_SECURITY (1) If the option is no security, the security implementation field does not exist. If the option is MD5, the security implementation field is a 16-octet field containing the message checksum and Service Group password. The password can be no more than eight octets.
Service Info	Describes the service group. The service type ID can have two values: WCCP2_SERVICE_STANDARD (0) WCCP2_SERVICE_DYNAMIC (1) If the service type is standard, the service is a well-known service, defined entirely by service ID. HTTP is an example of a well-known service. If the service type is dynamic, the following settings define the service: priority, protocol, service flags (which determine hashing), and port.
Router Identity Info	Contains the router IP address and ID, and lists (by IP address) all of the web caches with which the router intends to communicate.
Web Cache Identity Info	Contains the web cache IP address and redirection hash table mapping.
Router View Info	Contains the router's view of the service group (identities of the routers and caches).
Web Cache View Info	Contains the web cache's view of the service group.
Assignment Info	Shows the assignment of a web cache to a particular hashing bucket.
Router Query Info	Contains the router's IP address, address of the web cache being queried, and ID of the last router in the service group that received a Here I Am message from the web cache.
Capabilities Info	Used by routers to advertise supported packet forwarding, load balancing, and packet return methods; used by web caches to let routers know what method the web cache prefers.
Alternate Assignment	Contains hash table assignment information for load balancing.
Assignment Map	Contains mask/value set elements for service group.
Command Extension	Used by web caches to tell routers they are shutting down; used by routers to acknowledge a cache shutdown.

Service groups

یک گروه سرویس شامل مجموعه‌ای از روترها و Cache های دارای قابلیت WCCP است که پیام‌های WCCP را مبادله می‌کنند. روترها ترافیک وب را به Cache های گروه سرویس ارسال می‌کنند. پیکربندی گروه سرویس نحوه توزیع ترافیک به Cache در گروه سرویس می‌کند. روترها و Cache اطلاعات پیکربندی گروه سرویس را در پیام‌های Here I Am and I See You مبادله می‌کنند.

GRE packet encapsulation

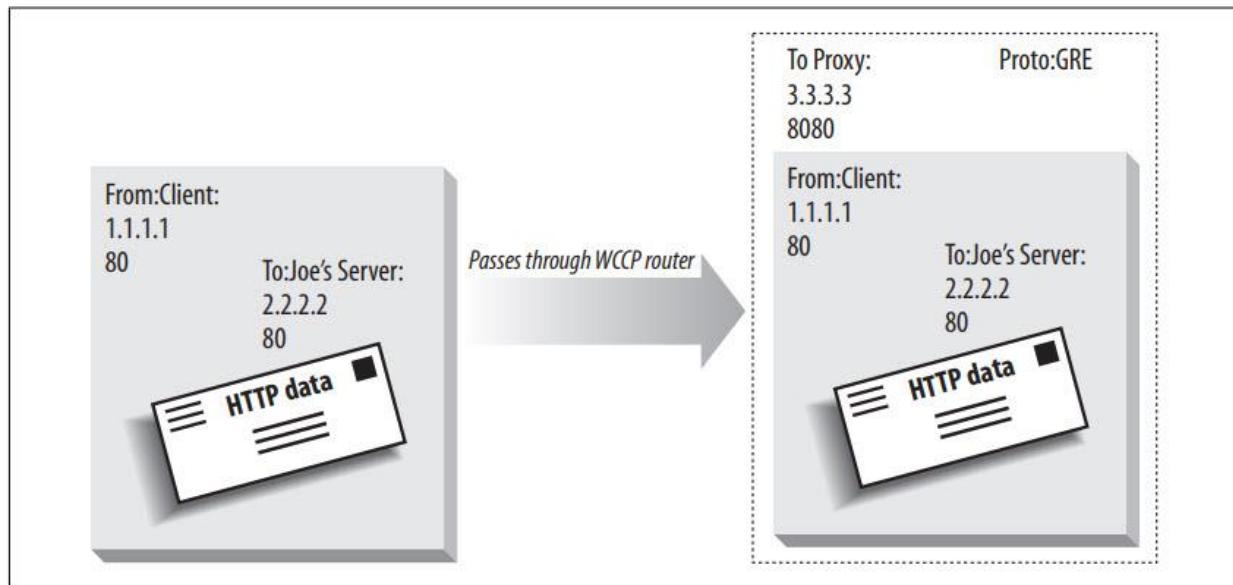
روترهایی که از WCCP پشتیبانی می‌کنند بسته‌های HTTP را با کپسوله کردن آن‌ها با آدرس IP سرور به یک سرور خاص هدایت می‌کنند. محصور کردن بسته همچنین حاوی یک فیلد proto هدر IP است





که نشان دهنده Encapsulation روتور عمومی (GRE) است. وجود فیلد proto به پروکسی دریافت کننده می‌گوید که یک بسته محصور شده دارد.

از آنجایی که بسته کپسوله شده است، آدرس IP کلاینت از بین نمی‌رود. شکل زیر کپسوله سازی بسته GRE را نشان می‌دهد.



WCCP load balancing

علاوه بر مسیریابی، روترهای WCCP می‌توانند بار را بین چندین سرور دریافت کننده متعادل کنند. روترهای WCCP و سرورهای دریافت کننده آنها، پیام‌های Heartbeat را رد و بدل می‌کنند تا به یکدیگر اطلاع دهند که در حال راهاندازی هستند. اگر یک سرور دریافت کننده خاص ارسال پیام‌های Heartbeat را متوقف کند، روتر WCCP به جای هدایت مجدد آن به آن گره، ترافیک درخواست را مستقیماً به اینترنت ارسال می‌کند. هنگامی که گره به سرویس باز می‌گردد، روتر WCCP دوباره شروع به دریافت پیام‌های Heartbeat می‌کند و ارسال ترافیک درخواست به گره را از سر می‌گیرد.

Internet Cache Protocol

ICP یا Internet Cache Protocol به کش‌ها اجازه می‌دهد تا به دنبال بازدیدهای محتوا در حافظه پنهان خواهر و برادر بگردند. اگر حافظه پنهان محتوا درخواست شده در یک پیام HTTP را نداشته باشد، می‌تواند بفهمد که آیا محتوا در Cache نزدیک است یا خیر، و اگر چنین است، محتوا را از آنجا بازیابی کند، امیدواریم از درخواست پرهزینه‌تر به سرور مبدا اجتناب شود. ICP را می‌توان به عنوان یک پروتکل





در نظر گرفت. این یک پروتکل تغییر مسیر است به این معنا که مقصد نهایی یک پیام درخواست HTTP را می‌توان توسط یک سری پرس و جوهای ICP تعیین کرد.

یک ICP Object Discovery Protocol است و از Cache های نزدیک می‌پرسد که آیا هر یک از آن‌ها URL خاصی در Cache خود دارند یا خیر. Cache های نزدیک یک پیام کوتاه می‌فرستند که می‌گوید «HIT» اگر آن URL را دارند یا «MISS» اگر ندارند. سپس Cache HTTP را به همسایه‌ای که به دارای شی است باز کند.

ICP ساده و سبک است. پیام‌های ICP ساختارهای بسته بندی شده ۳۲ بیتی به ترتیب بایت شبکه هستند که تجزیه آن‌ها را آسان می‌کند. آن‌ها برای کارایی در دیتاگرام‌های UDP حمل می‌شوند. UDP یک پروتکل اینترنتی غیرقابل اعتماد است، به این معنی که داده‌ها می‌توانند در حین انتقال از بین بروند، بنابراین برنامه‌هایی که به ICP صحبت می‌کنند باید برای شناسایی دیتاگرام‌های از دست رفته دارای Timeout باشند.

در اینجا توضیح مختصری از بخش‌های پیام ICP آورده شده است:

Opcode

یک مقدار ۸ بیتی است که معنای پیام ICP را توصیف می‌کند. کدهای اولیه عبارتند از پیام‌های درخواست ICP_OP_QUERY و پیام‌های پاسخ ICP_OP_HIT و ICP_OP_MISS.

Version

شماره نسخه ۸ بیتی شماره نسخه پروتکل ICP را توصیف می‌کند. نسخه ICP مورد استفاده Squid که در اینترنت RFC 2186 مستند شده است، نسخه ۲ است.

Message length

اندازه کل پیام ICP بر حسب بایت. از آنجایی که تنها ۱۶ بیت وجود دارد، اندازه پیام ICP نمی‌تواند بزرگتر از ۱۶۳۸۳ بایت باشد. URL ها معمولاً کوتاهتر از ۱۶ کیلوبایت هستند. اگر طولانی تر از آن باشند، بسیاری از برنامه‌های کاربردی وب آن‌ها را پردازش نمی‌کنند.

Request number

دارای Cache از شماره درخواست برای پیگیری چندین درخواست و پاسخ به طور همزمان استفاده می‌کند. یک پیام پاسخ ICP همیشه باید دارای همان شماره درخواست پیام درخواست ICP باشد که پاسخ را آغاز کرده است.





Options

فیلد گزینه‌های ۳۲ بیتی ICP یک بردار بیت حاوی Flag هایی است که رفتار ICP را تغییر می‌دهد. Flag دو ICPv2 را تعریف می‌کند که هر دو درخواست‌های ICP_OP_QUERY را تغییر می‌دهند. ICP_FLAG_HIT_OBJ Flag بازگشت داده‌های سند را در پاسخ‌های ICP فعال و غیرفعال می‌کند. ICP_FLAG_SRC_RTT Flag تخمینی از زمان رفت و برگشت به سرور مبدا را درخواست می‌کند، همانطور که توسط یک Sibling Cache اندازه گیری می‌شود.

Option data

داده‌های گزینه ۳۲ بیتی برای ویژگی‌های اختیاری رزرو شده است. ICPv2 از ۱۶ بیت پایین داده‌های گزینه استفاده می‌کند تا تخمین زمان رفت و برگشت اختیاری را از Sibling به سرور مبدا انجام دهد.

Sender host address

یک فیلد تاریخی که دارای آدرس IP ۳۲ بیتی فرستنده پیام است. در عمل استفاده نمی‌شود.

Payload

محتویات Payload بسته به نوع پیام متفاوت است. برای ICP_OP_QUERY یک آدرس میزبان درخواست‌کننده اصلی ۴ بایتی است که به دنبال آن یک URL پایان‌یافته NUL است. برای payload، ICP_OP_HIT_OBJ یک URL پایان‌یافته NUL است که یک اندازه شی ۱۶ بیتی و به دنبال آن داده‌های شی است.

برای اطلاعات بیشتر در مورد ICP، به RFC‌های اطلاعاتی ۲۱۸۶ و ۲۱۸۷ مراجعه کنید. مرجع عالی ICP و همتا نیز در آزمایشگاه ملی ایالات متحده برای تحقیقات شبکه کاربردی (<http://www.nlanr.net/Squid>) موجود است.

Cache Array Routing Protocol

سرورهای پروکسی با رهگیری درخواست‌های کاربران و ارائه کپی‌های Cache شده از اشیاء وب درخواستی، ترافیک اینترنت را تا حد زیادی کاهش می‌دهند. با این حال، با افزایش تعداد کاربران، حجم بالای ترافیک می‌تواند خود سرورهای پروکسی را بارگیری کند.

یک راه حل برای این مشکل استفاده از چندین سرور پراکسی برای توزیع بار در مجموعه‌ای از سرورها است. Microsoft CARP یا Cache Array Routing Protocol استانداردی است که توسط Netscape Communication Corporation و Corporation برای مدیریت مجموعه‌ای از

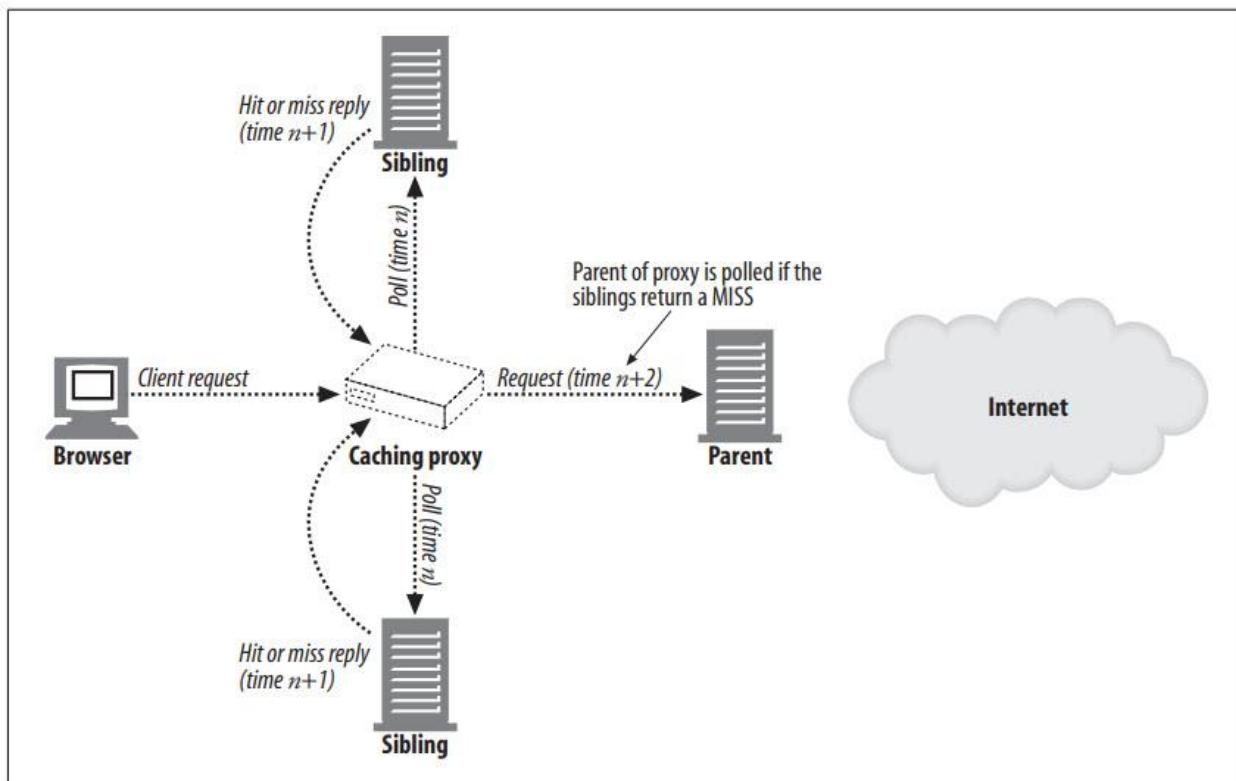




سرورهای پروکسی پیشنهاد شده است به طوری که آرایه‌ای از سرورهای پراکسی به عنوان یک Cache منطقی برای کلاینت‌ها به نظر می‌رسد.

CARP جایگزینی برای ICP است. هر دو CARP و ICP به مدیران اجازه می‌دهند تا با استفاده از چندین سرور پراکسی عملکرد را بهبود بخشنند. این بخش در مورد تفاوت CARP با ICP، مزایا و معایب استفاده از CARP نسبت به ICP و جزئیات فنی نحوه پیاده سازی پروتکل CARP بحث می‌کند.

پس از از دست رفتن Cache در ICP، سرور پروکسی با استفاده از قالب پیام Cache، از ICP مجاور پرس و جو می‌کند تا در دسترس بودن شی وب را تعیین کند. Cache های مجاور با یک "MISS" یا "HIT" پاسخ می‌دهند و سرور پروکسی درخواست کننده از این پاسخ‌ها برای انتخاب مناسب‌ترین مکان برای بازیابی شی استفاده می‌کند. اگر سرورهای پروکسی ICP به صورت سلسله مرتبی مرتب شده باشند، خطای Parent افزایش می‌یابد. شکل زیر به صورت نموداری نشان می‌دهد که چگونه HIT ها و اشتباهات با استفاده از ICP حل می‌شوند.

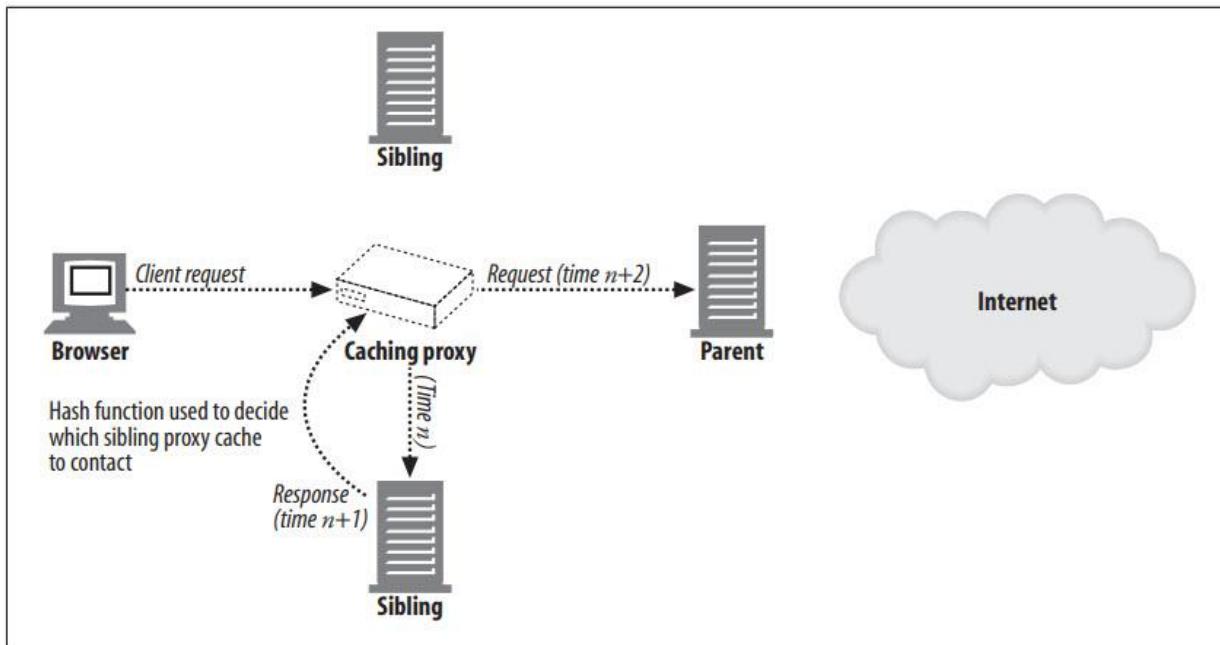


توجه داشته باشید که هر یک از سرورهای پراکسی که با استفاده از پروتکل ICP به یکدیگر متصل شده‌اند، یک سرور Mirror Cache مستقل با Mirror های اضافی از محتوا هستند، به این معنی که ورودی‌های تکراری اشیاء وب در سرورهای پراکسی امکان پذیر است. در مقابل، مجموعه‌ای از سرورهای متصل با استفاده از CARP به عنوان یک سرور واحد و بزرگ عمل می‌کند که هر سرور جزء تنها بخشی از کل اسناد Cache می‌باشد.





شده را شامل می‌شود. با اعمال یک تابع هش به URL یک شی وب، CARP اشیاء وب را به یک سرور پراکسی خاص Map می‌کند. از آنجا که هر شی وب دارای یک خانه منحصر به فرد است، ما می‌توانیم مکان شی را با یک جستجوی واحد تعیین کنیم، نه اینکه هر یک از سرورهای پراکسی پیکربندی شده در مجموعه را Poll کنیم. شکل زیر رویکرد CARP را خلاصه می‌کند.



اگرچه شکل بالا Caching Proxy را به عنوان واسطه بین کلاینتها و سرورهای پراکسی نشان می‌دهد که بار را بین سرورهای پراکسی مختلف توزیع می‌کند، ممکن است این عملکرد توسط خود توسعه کلاینتها ارائه شود. مرورگرهای تجاری مانند Netscape Navigator و Internet Explorer را می‌توان برای محاسبه عملکرد هش در قالب یک افزونه پیکربندی کرد که سرور پراکسی را که درخواست باید به آن ارسال شود، تعیین می‌کند.

وضوح قطعی سرور پراکسی در CARP به این معنی است که نیازی به ارسال پرس‌وجوها به همه همسایگان نیست، به این معنی که این روش به پیام‌های intercache کمتری برای ارسال نیاز دارد. همانطور که سرورهای پراکسی بیشتری به پیکربندی اضافه می‌شوند، سیستم Cache جمعی نسبتاً خوب مقیاس می‌شود. با این حال، یک نقطه ضعف CARP این است که اگر یکی از سرورهای پراکسی در دسترس نباشد، تابع هش باید اصلاح شود تا این تغییر را منعکس کند، و محتویات سرورهای پراکسی باید در میان سرورهای پراکسی موجود تغییر شکل دهند.

اگر سرور پروکسی اغلب از کار بیفتند، ممکن است هزینه بر باشد. در مقابل، محتوای اضافی در سرورهای پروکسی ICP به این معنی است که نیازی به تغییر شکل نیست. مشکل بالقوه دیگر این است که چون CARP یک پروتکل جدید است، سرورهای پراکسی موجود که فقط پروتکل ICP را اجرا می‌کنند ممکن است به راحتی در مجموعه CARP گنجانده نشوند.





پس از توضیح تفاوت بین CARP و ICP، اجازه دهد اکنون CARP را با جزئیات بیشتری شرح دهیم. روش تغییر مسیر CARP شامل وظایف زیر است:

- جدولی از سرورهای پراکسی شرکت کننده داشته باشد. این سرورهای پروکسی به صورت دوره‌ای Poll می‌شوند تا بینند کدام یک هنوز فعال هستند.
 - برای هر سرور پراکسی شرکت کننده، یکتابع هش را محاسبه کنید. مقدار بازگردانده شده توسط تابع هش میزان باری را که این پروکسی می‌تواند تحمل کند را در نظر می‌گیرد.
 - یکتابع هش جداگانه تعریف کنید که بر اساس URL شی وب درخواستی، عددی را برمی‌گرداند.
 - مجموع تابع هش URL و تابع هش سرورهای پراکسی را برای بدست آوردن آرایه‌ای از اعداد در نظر بگیرید. حداقل مقدار این اعداد، سرور پراکسی مورد استفاده برای URL را تعیین می‌کند. از آنجایی که مقادیر محاسبه شده قطعی هستند، درخواست‌های بعدی برای همان شی وب به همان سرور پراکسی ارسال می‌شوند.
- این چهار کار را می‌توان بر روی مرورگر، در یک افزونه یا بر روی یک سرور میانی محاسبه کرد.

برای هر مجموعه‌ای از سرورهای پراکسی، جدولی ایجاد کنید که تمام سرورهای مجموعه را فهرست می‌کند. هر ورودی در جدول باید حاوی اطلاعاتی در مورد فاکتورهای بار، مقادیر Time To Live (TTL) و پارامترهای کلی مانند تعداد دفعات نظرسنجی اعضا باشد. ضریب بار نشان می‌دهد که دستگاه چقدر بار می‌تواند تحمل کند، که به سرعت پردازنه و ظرفیت هارد دیسک آن دستگاه بستگی دارد. جدول را می‌توان از راه دور از طریق رابط RPC نگهداری کرد. هنگامی که فیلدهای جداول توسط RPC به روز شدن، می‌توان آنها را برای کلاینتها و پراکسی‌های پایین دستی، در دسترس یا منتشر کرد. این انتشار در HTTP انجام می‌شود و به هر کلاینت یا سرور پراکسی اجازه می‌دهد تا اطلاعات جدول را بدون معرفی پروتکل بین پراکسی دیگری مصرف کند. کلاینتها و سرورهای پروکسی به سادگی از یک URL شناخته شده برای بازیابی جدول استفاده می‌کنند.

تابع هش مورد استفاده باید اطمینان حاصل کند که اشیاء وب از نظر آماری در سراسر سرورهای پراکسی شرکت کننده توزیع شده‌اند. ضریب بار سرور پراکسی باید برای تعیین احتمال آماری تخصیص یک شی وب به آن پروکسی استفاده شود.

به طور خلاصه، پروتکل CARP به گروهی از سرورهای پروکسی اجازه می‌دهد تا به عنوان یک Cache جمعی، به جای گروهی از Cache‌های همکار اما جداگانه (مانند ICP) مشاهده شوند. یک مسیر قطعی درخواست قطعی، خانه یک شی وب خاص را در یک جهش واحد پیدا می‌کند. این امر ترافیک بین پراکسی را که اغلب برای یافتن شی وب در گروهی از سرورهای پراکسی در ICP ایجاد می‌شود، حذف می‌کند. CARP همچنین از ذخیره کپی‌های تکراری اشیاء وب در سرورهای پراکسی مختلف جلوگیری می‌کند، که این





مزیت را دارد که سیستم Cache مجموعاً ظرفیت بیشتری برای ذخیره‌سازی اشیاء وب دارد، اما این عیب را نیز دارد که شکست در هر یک از پراکسی‌ها مستلزم تغییر دادن برخی از محتویات Cache به پراکسی‌های موجود است.

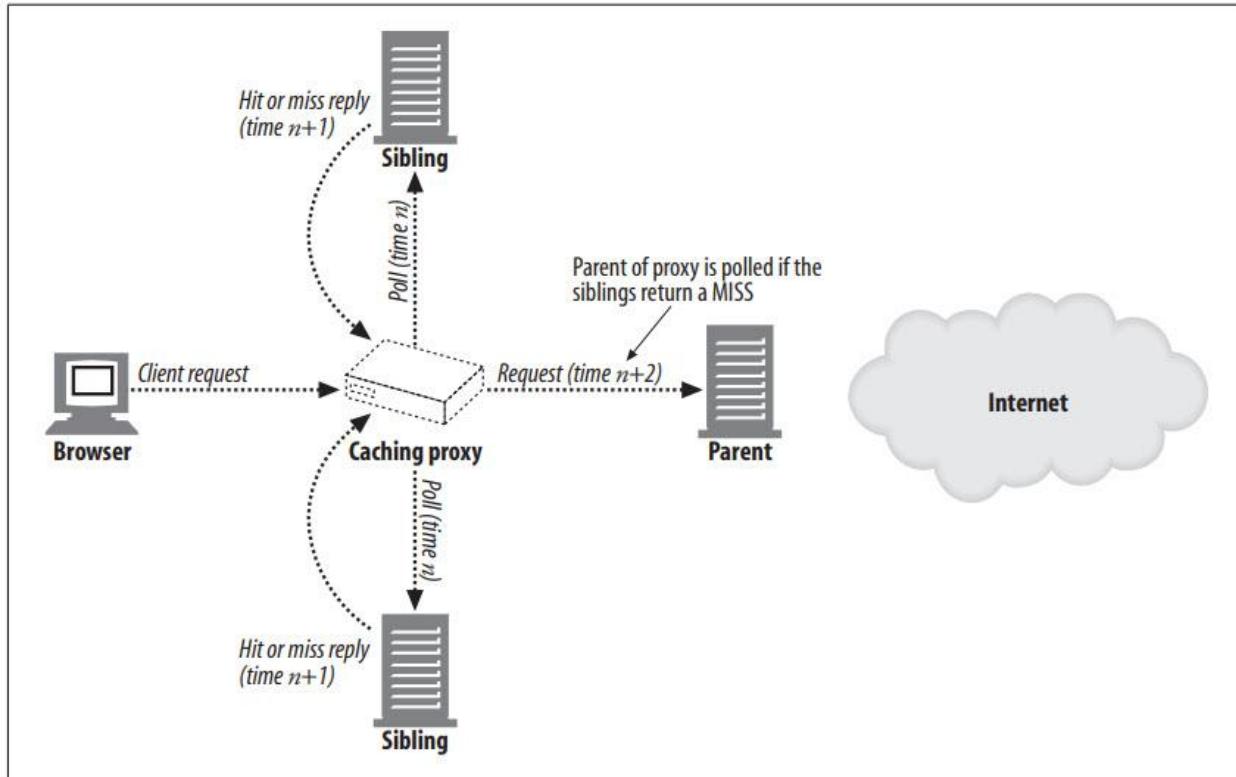
Hyper Text Caching Protocol

پیش از این، ICP را مورد بحث قرار دادیم، پروتکلی که به Cache‌های پراکسی اجازه می‌دهد تا از Sibling در مورد وجود اسناد پرس و جو کنند. با این حال، ICP با در نظر گرفتن HTTP/0.9 طراحی شده است و بنابراین به Cache اجازه می‌دهد فقط URL را هنگام پرس و جو از Sibling در مورد وجود یک منبع ارسال کنند. نسخه‌های 1.0 و 1.1 HTTP بسیاری از هدرهای درخواست جدید را معرفی کردند که همراه با URL، برای تصمیم‌گیری در مورد تطابق اسناد استفاده می‌شوند، بنابراین ارسال URL در یک درخواست ممکن است منجر به پاسخ‌های دقیق نشود.

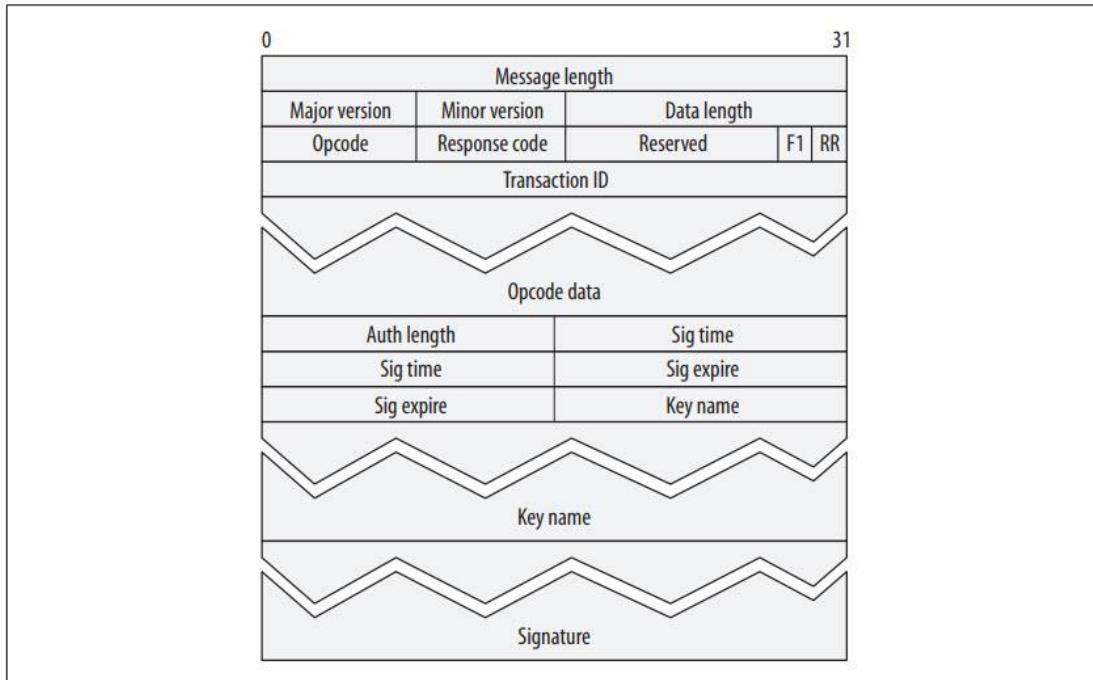
پروتکل Hyper Text Caching Protocol (HTCP) برای اسناد با اجازه دادن به Sibling برای پرس و جو از یکدیگر برای حضور اسناد با استفاده از URL و همه هدرهای درخواست و پاسخ، احتمال بازدیدهای نادرست را کاهش می‌دهد. علاوه بر این، HTCP به Cache‌های Sibling اجازه می‌دهد تا اضافه و حذف اسناد انتخاب شده را در Cache یکدیگر نظارت و درخواست کنند و در سیاست‌های Cache اسناد ذخیره‌شده یکدیگر تغییراتی ایجاد کنند.

شکل زیر که یک تراکنش ICP را نشان می‌دهد، همچنین می‌تواند برای نشان دادن تراکنش HTCP استفاده شود—HTCP فقط یک پروتکل کشف شی دیگر است. اگر یک Cache نزدیک سند را داشته باشد، درخواست کننده می‌تواند یک اتصال HTTP را به Cache باز کند تا یک کپی از سند دریافت کند. تفاوت بین ICP و تراکنش HTCP در سطح جزئیات درخواست‌ها و پاسخ‌ها است.





ساختار پیام‌های HTCP در شکل زیر نشان داده شده است.





قسمت هدر شامل طول پیام و نسخه پیام است. بخش داده با طول داده شروع می‌شود و شامل کدهای عملیاتی، کدهای پاسخ، و برخی Flag‌ها و شناسه‌ها می‌شود و با داده‌های واقعی خاتمه می‌یابد. یک بخش تأیید هویت اختیاری ممکن است بعد از بخش داده باشد.

جزئیات فیلدهای پیام به شرح زیر است:

Header

بخش Header شامل یک طول پیام ۳۲ بیتی، یک نسخه پروتکل اصلی ۸ بیتی و یک نسخه پروتکل فرعی ۸ بیتی است. طول پیام شامل تمام اندازه‌های هدر، داده‌ها و احراز هویت است.

Data

بخش Data حاوی پیام HTCP است و ساختار آن در شکل بالا نشان داده شده است. اجزای داده در جدول زیر توضیح داده شده‌اند.





Component	Description
Data length	A 16-bit value of the number of bytes in the Data section including the length of the Length field itself.
Opcode	The 4-bit operation code for the HTCP transaction. The full list of opcodes is provided in Table 20-7.
Response code	A 4-bit key indicating the success or failure of the transaction. The possible values are: <ul style="list-style-type: none">• 0—Authentication was not used, but is needed• 1—Authentication was used, but is not satisfactory• 2—Unimplemented opcode• 3—Major version not supported• 4—Minor version not supported• 5—Inappropriate, disallowed, or undesirable opcode
F1	F1 is overloaded—if the message is a request, F1 is a 1-bit flag set by the requestor indicating that it needs a response (F1=1); if the message is a response, F1 is a 1-bit flag indicating whether the response is to be interpreted as a response to the overall message (F1=1) or just as a response to the Opcode data fields (F1=0).
RR	A 1-bit flag indicating that the message is a request (RR=0) or a response (RR=1).
Transaction ID	A 32-bit value that, combined with the requestor's network address, uniquely identifies the HTCP transaction.
Opcode data	Opcode data is opcode-dependent. See Table 20-7.

جدول زیر opcode های HTCP و انواع داده مربوط به آنها را فهرست می‌کند.





Opcode	Value	Description	Response codes	Opcode data
NOP	0	Essentially a “ping” operation.	Always 0	None
TST	1		0 if entity is present, 1 if entity is not present	Contains the URL and request headers in the request and just response headers in the response
MON	2		0 if accepted, 1 if refused	
SET	3	The SET message allows caches to request changes in caching policies. See Table 20-9 for a list of the headers that can be used in SET messages.	0 if accepted, 1 if ignored	
CLR	4		0 if I had it, but it's now gone; 1 if I had it, but I am keeping it; and 2 if I didn't have it	

HTCP Authentication

بخش احراز هویت پیام HTCP اختیاری بوده که اجزای آن در جدول زیر توضیح داده شده است.

Component	Description
Auth length	The 16-bit number of bytes in the Authentication section of the message, including the length of the Length field itself.
Sig time	A 32-bit number representing the number of seconds since 00:00:00 Jan 1, 1970 GMT at the time that the signature is generated.
Sig expire	A 32-bit number representing the number of seconds since 00:00:00 Jan 1, 1970 GMT when the signature will expire.
Key name	A string that specifies the name of the shared secret. The Key section has two parts: the 16-bit length in bytes of the string that follows, followed by the stream of uninterrupted bytes of the string.
Signature	The HMAC-MD5 digest with a B value of 64 (representing the source and destination IP addresses and ports), the major and minor HTCP versions of the message, the Sig time and Sig expires values, the full HTCP data, and the key. The Signature also has two parts: the 16-bit length in bytes of the string, followed by the string.

Setting Caching Policies

پیام Cache ها اجازه می‌دهد تا تغییراتی را در سیاست‌های ذخیره سازی اسناد ذخیره شده در Cache درخواست کنند. هدرهای قابل استفاده در پیام‌های SET در جدول زیر توضیح داده شده است.





Header	Description
Cache-Vary	The requestor has learned that the content varies on a set of headers different from the set in the response Vary header. This header overrides the response Vary header.
Cache-Location	The list of proxy caches that also may have copies of this object.
Cache-Policy	The requestor has learned the caching policies for this object in more detail than is specified in the response headers. Possible values are: "no-cache," meaning that the response is not cacheable but may be shareable among simultaneous requestors; "no-share," meaning that the object is not shareable; and "no-cache-cookie," meaning that the content may change as a result of cookies and caching therefore is not advised.
Cache-Flags	The requestor has modified the object's caching policies and the object may have to be treated specially and not necessarily in accordance with the object's actual policies.
Cache-Expiry	The actual expiration time for the document as learned by the requestor.
Cache-MD5	The requestor-computed MD5 checksum of the object, which may be different from the value in the Content-MD5 header, or may be supplied because the object does not have a Content-MD5 header.
Cache-to-Origin	The requestor-measured round-trip time to an origin server. The format of the values in this header is <origin server name or ip> <average round-trip time in seconds> <number of samples> <number of router hops between requestor and origin server>.

با اجازه دادن به هدرهای درخواست و پاسخ برای ارسال پیام‌های جستجو به HTCP Sibling Cache می‌تواند نرخ false-hit را در کوئری‌های Cache کاهش دهد. با اجازه دادن بیشتر به Sibling Cache برای تبادل اطلاعات خط مشی با یکدیگر، HTCP می‌تواند توانایی Sibling Cache را برای همکاری با یکدیگر بهبود بخشد.



For More Information

DNS and Bind - Cricket Liu, Paul Albitz, and Mike Loukides, O'Reilly & Associates, Inc.

<http://www.wrec.org/Drafts/draft-cooper-webi-wpad-00.txt>

<http://home.netscape.com/eng.mozilla/2.0/relnotes/demo/proxy-live.html>

<http://www.ietf.org/rfc/rfc2186.txt>

<http://icp.ircache.net/carp.txt>

<http://www.ietf.org/rfc/rfc2756.txt>

<http://www.ietf.org/internet-drafts/draft-wilson-wrec-wccp-v2-00.txt>

<http://www.ietf.org/rfc/rfc2131.txt?number=2131>

<http://www.ietf.org/rfc/rfc2132.txt?number=2132>

<http://www.ietf.org/rfc/rfc2608.txt?number=2608>

<http://www.ietf.org/rfc/rfc2219.txt?number=2219>





فصل بیست و یکم - Logging and Usage Tracking

تقریباً همه سرورها و پروکسی‌ها خلاصه‌ای از تراکنش‌های HTTP پردازش شده را ثبت می‌کنند. این کار به دلایل مختلفی انجام می‌شود: ردیابی استفاده، امنیت، Billing، تشخیص خطا و غیره. در این فصل، ما یک تور مختصر از گزارش‌گیری را بررسی می‌کنیم و بررسی می‌کنیم که معمولاً چه اطلاعاتی در مورد تراکنش‌های HTTP ثبت می‌شود و برخی از قالب‌های گزارش رایج شامل چه مواردی هستند.

What to Log

در بیشتر موارد، Logging به دو دلیل انجام می‌شود: جستجوی مشکلات در سرور یا پروکسی (به عنوان مثال، درخواست‌هایی که با شکست مواجه می‌شوند)، و ایجاد آمار در مورد نحوه دسترسی به وبسایتها. آمار برای بازاریابی، صورتحساب و برنامه‌ریزی ظرفیت (به عنوان مثال، تعیین نیاز به سرورهای اضافی یا پهنهای باند) مفید است.

شما می‌توانید تمام هدرها را در یک تراکنش HTTP ثبت کنید، اما برای سرورها و پروکسی‌هایی که میلیون‌ها تراکنش را در روز پردازش می‌کنند، بخش عمده‌ای از تمام آن داده‌ها به سرعت از کنترل خارج می‌شوند. همچنین در نهایت اطلاعات زیادی را ثبت می‌شوند که واقعاً به آن‌ها اهمیت داده نشده و ممکن است هرگز به آن‌ها نگاه نکنید.

به طور معمول، فقط اصول اولیه یک تراکنش ثبت می‌شود. چند نمونه از فیلدهایی که معمولاً ثبت شده‌اند عبارتند از:

- HTTP method
- HTTP version of client and server
- URL of the requested resource
- HTTP status code of the response
- Size of the request and response messages (including any entity bodies)
- Timestamp of when the transaction occurred
- Referer and User-Agent header values

متد HTTP و URL نشان می‌دهد که درخواست چه کاری را انجام داده است - به عنوان مثال، دریافت یک منبع یا ارسال فرم سفارش. URL را می‌توان برای ردیابی محبوبیت صفحات در وب سایت استفاده کرد.





رشته‌های نسخه^{۶۷} نکاتی را در مورد کلاینت و سرور ارائه می‌دهند که در رفع اشکال تعاملات عجیب یا غیرمنتظره بین کلاینت‌ها و سرورها مفید است. برای مثال، اگر درخواست‌ها با نرخی بالاتر از حد انتظار شکست بخورند، اطلاعات نسخه ممکن است به نسخه جدیدی از مرورگری اشاره کند که قادر به تعامل با سرور نیست.

کد وضعیت HTTP می‌گوید که چه اتفاقی برای درخواست افتاده است: آیا موفقیت آمیز بود، تلاش مجوز ناموفق بود، منبع پیدا شد، و غیره.

اندازه درخواست/پاسخ و مهر زمانی عمدتاً برای اهداف حسابداری استفاده می‌شود. به عنوان مثال، برای ردیابی تعداد بایت‌هایی که به داخل، خارج یا از طریق برنامه وارد شده‌اند. مهر زمان همچنین می‌تواند برای مرتبط کردن مشکلات مشاهده شده با درخواست‌هایی که در آن زمان انجام می‌شد استفاده شود.

Log Formats

چندین فرمت گزارش استاندارد شده‌اند و ما در این بخش به برخی از رایج‌ترین فرمت‌ها خواهیم پرداخت. اکثر برنامه‌های تجاری و منبع باز Logging از HTTP⁶⁸ را در یک یا چند مورد از این قالب‌های رایج پشتیبانی می‌کنند. بسیاری از این برنامه‌ها همچنین از توانایی مدیران برای پیکربندی فرمت‌های گزارش و ایجاد فرمت‌های سفارشی خود پشتیبانی می‌کنند.

یکی از مزایای اصلی پشتیبانی (برای برنامه‌ها) و استفاده (برای مدیران) از این فرمت‌های استاندارددتر در توانایی استفاده از ابزارهایی است که برای پردازش و تولید آمار اولیه از این گزارش‌ها ساخته شده‌اند.

Common Log Format

یکی از رایج‌ترین فرمت‌های گزارش که امروزه مورد استفاده قرار می‌گیرد، به طور مناسب، Common Log Format یا فرمت گزارش مشترک نامیده می‌شود. این فرمت در ابتدا توسط NCSA⁶⁹ تعریف شده بود، بسیاری از سرورها از این فرمت log⁷⁰ به عنوان پیش فرض استفاده می‌کنند. اکثر سرورهای تجاری و منبع باز را می‌توان برای استفاده از این فرمت پیکربندی کرد و بسیاری از ابزارهای تجاری و رایگان برای کمک به تجزیه فایل‌های گزارش رایج وجود دارد. جدول زیر فیلدهای قالب گزارش مشترک را به ترتیب فهرست می‌کند.

⁶⁷ version strings





Field	Description
remotehost	The hostname or IP address of the requestor's machine (IP if the server was not configured to perform reverse DNS or cannot look up the requestor's hostname)
username	If an <i>ident</i> lookup was performed, the requestor's authenticated username ^a
auth-username	If authentication was performed, the username with which the requestor authenticated
timestamp	The date and time of the request
request-line	The exact text of the HTTP request line, "GET /index.html HTTP/1.1"
response-code	The HTTP status code that was returned in the response
response-size	The Content-Length of the response entity—if no entity was returned in the response, a zero is logged

مثال زیر چند نمونه از ورودی‌های فرمات گزارش رایج را نشان می‌دهد.

```
209.1.32.44 - - [03/Oct/1999:14:16:00 -0400] "GET / HTTP/1.0" 200 1024
http-guide.com - dg [03/Oct/1999:14:16:32 -0400] "GET / HTTP/1.0" 200 477
http-guide.com - dg [03/Oct/1999:14:16:32 -0400] "GET /foo HTTP/1.0" 404 0
```

در این مثال‌ها، فیلد‌ها به صورت زیر اختصاص داده می‌شوند:

Field	Entry 1	Entry 2	Entry 2
remotehost	209.1.32.44	http-guide.com	http-guide.com
username	<empty>	<empty>	<empty>
auth-username	<empty>	dg	dg
timestamp	03/Oct/1999:14:16:00 -0400	03/Oct/1999:14:16:32 -0400	03/Oct/1999:14:16:32 -0400
request-line	GET / HTTP/1.0	GET / HTTP/1.0	GET /foo HTTP/1.0
response-code	200	200	404
response-size	1024	477	0

توجه داشته باشید که فیلد میزبان راه دور می‌تواند یک نام میزبان، مانند http-guide.com، یا یک آدرس IP مانند ۲۰۹.۱.۳۲.۴۴ باشد.

خط تیره فیلد‌های دوم (username) و سوم (auth-username) نشان دهنده خالی بودن فیلد‌ها است. این نشان می‌دهد که یا جستجوی شناسه رخ نداده است (فیلد دوم خالی است) یا احراز هویت انجام نشده است (فیلد سوم خالی است).





Combined Log Format

یکی دیگر از فرمتهای گزارش رایج، Combined Log Format یا فرمت گزارش ترکیبی است. این فرمت توسط سرورهایی مانند آپاچی پشتیبانی می‌شود. فرمت Combined Log بسیار شبیه به فرمت Common Log است. در واقع، آن را دقیقاً با افزودن دو فیلد منعکس می‌کند (در جدول زیر فهرست شده است). فیلد-User-Agent برای مشخص کردن اینکه کدام برنامه‌های سرویس گیرنده HTTP درخواست‌های ثبتشده را ارسال می‌کنند مفید است، در حالی که فیلد Referer جزئیات بیشتری درباره جایی که درخواست‌کننده این URL را پیدا کرده است، ارائه می‌دهد.

Field	Description
Referer	The contents of the Referer HTTP header
User-Agent	The contents of the User-Agent HTTP header

مثال زیر مثالی از ورودی فرمت گزارش ترکیبی را ارائه می‌دهد.

209.1.32.44 - - [03/Oct/1999:14:16:00 -0400] "GET / HTTP/1.0" 200 1024 "http://www.joes-hardware.com/" "5.0: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)"

در مثال بالا، فیلدهای Referer و User-Agent به صورت زیر اختصاص داده شده‌اند:

Field	Value
Referer	http://www.joes-hardware.com/
User-Agent	5.0: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)

هفت فیلد اول نمونه ورودی فرمت گزارش ترکیبی در مثال بالا دقیقاً مانند فرمت گزارش مشترک هستند. دو فیلد جدید، User-Agent و Referer، در انتهای ورودی گزارش قرار می‌گیرند.

Netscape Extended Log Format

هنگامی که Netscape وارد فضای برنامه تجاری HTTP شد، فرمتهای گزارش بسیاری را برای سرورهای خود تعریف کرد که توسط سایر توسعه دهندگان برنامه HTTP پذیرفته شده است. فرمتهای Netscape از قالب NCSA Common Log مشتق شده‌اند، اما آن‌ها این قالب را برای ترکیب فیلدهای مربوط به برنامه‌های HTTP مانند پراکسی‌ها و Cache و وب گسترش می‌دهند.





هفت فیلد اول در Common Log با فرمت Netscape Extended Log Format یکسان است. جدول زیر فیلدهای جدیدی را که قالب Netscape Extended Log معرفی می‌کند، به ترتیب فهرست می‌کند.

Field	Description
proxy-response-code	If the transaction went through a proxy, the HTTP response code from the server to the proxy
proxy-response-size	If the transaction went through a proxy, the Content-Length of the server's response entity sent to the proxy
client-request-size	The Content-Length of any body or entity in the client's request to the proxy
proxy-request-size	If the transaction went through a proxy, the Content-Length of any body or entity in the proxy's request to the server
client-request-hdr-size	The length, in bytes, of the client's request headers
proxy-response-hdr-size	If the transaction went through a proxy, the length, in bytes, of the proxy's response headers that were sent to the requestor
proxy-request-hdr-size	If the transaction went through a proxy, the length, in bytes, of the proxy's request headers that were sent to the server
server-response-hdr-size	The length, in bytes, of the server's response headers
proxy-timestamp	If the transaction went through a proxy, the elapsed time for the request and response to travel through the proxy, in seconds

مثال زیر نمونه‌ای از یک ورودی Netscape Extended Log Format را نشان می‌دهد.

```
209.1.32.44 - - [03/Oct/1999:14:16:00-0400] "GET / HTTP/1.0" 200 1024 200 1024 0 0 215 260
279 254 3
```

در این مثال، فیلدهای توسعه یافته به صورت زیر اختصاص داده می‌شوند:



Field	Value
proxy-response-code	200
proxy-response-size	1024
client-request-size	0
proxy-request-size	0
client-request-hdr-size	215
proxy-response-hdr-size	260
proxy-request-hdr-size	279
server-response-hdr-size	254
proxy-timestamp	3

هفت فیلد اول نمونه ورودی در مثال بالا ورودی‌های نمونه Common Log Format را منعکس می‌کند.

Netscape Extended 2 Log Format

یکی دیگر از فرمتهای گزارش Netscape Extended Log Format، فرمت Extended Log را می‌گیرد و اطلاعات بیشتری مربوط به پروکسی HTTP و برنامه‌های Cache و براحتی اضافی به ترسیم تصویر بهتری از تعاملات بین یک سرویس گیرنده HTTP و یک برنامه پراکسی HTTP کمک می‌کند.

فرمت Netscape Extended 2 Log مشتق شده است و فیلدهای اولیه آن با موارد ذکر شده در جدول پیشین یکسان است.

جدول زیر فیلدهای اضافی فرمت Log Extended 2 Netscape را به ترتیب فهرست می‌کند.

Field	Description
route	The route that the proxy used to make the request for the client (see Table 21-5)
client-finish-status-code	The client finish status code; specifies whether the client request to the proxy completed successfully (FIN) or was interrupted (INTR)
proxy-finish-status-code	The proxy finish status code; specifies whether the proxy request to the server completed successfully (FIN) or was interrupted (INTR)
cache-result-code	The cache result code; tells how the cache responded to the request ^a

مثال زیر مثالی از ورودی Netscape Extended 2 Log Format را نشان می‌دهد.





209.1.32.44 - - [03/Oct/1999:14:16:00-0400] "GET / HTTP/1.0" 200 1024 200 1024 0 0 215 260
279 254 3 DIRECT FIN FIN WRITTEN

فیلد های توسعه یافته در این مثال به صورت زیر اختصاص داده می شوند:

Field	Value
route	DIRECT
client-finish-status-code	FIN
proxy-finish-status-code	FIN
cache-result-code	WRITTEN

۱۶ فیلد اول در ورودی Log در مثال بالا ورودی های نمونه فرمت Extended Netscape را منعکس می کند.

جدول زیر کدهای مسیر Netscape معتبر را فهرست می کند.

Value	Description
DIRECT	The resource was fetched directly from the server.
PROXY(host:port)	The resource was fetched through the proxy "host."
SOCKS(socks:port)	The resource was fetched through the SOCKS server "host."

جدول زیر کدهای پایان معتبر Netscape را فهرست می کند.

Value	Description
-	The request never even started.
FIN	The request was completed successfully.
INTR	The request was interrupted by the client or ended by a proxy/server.
TIMEOUT	The request was timed out by the proxy/server.

جدول زیر کدهای Cache معتبر Netscape را فهرست می کند.





Code	Description
-	The resource was uncacheable.
WRITTEN	The resource was written into the cache.
REFRESHED	The resource was cached and it was refreshed.
NO-CHECK	The cached resource was returned; no freshness check was done.
UP-TO-DATE	The cached resource was returned; a freshness check was done.
HOST-NOT-AVAILABLE	The cached resource was returned; no freshness check was done because the remote server was not available.
CL-MISMATCH	The resource was not written to the cache; the write was aborted because the Content-Length did not match the resource size.
ERROR	The resource was not written to the cache due to some error; for example, a timeout occurred or the client aborted the transaction.

برنامه‌های Netscape، مانند بسیاری از برنامه‌های HTTP دیگر، فرمتهای گزارش دیگری نیز دارند، از جمله فرمت گزارش انعطاف‌پذیر و ابزاری برای مدیران برای خروجی فیلدهای ثبت سفارشی. این قالب‌ها به مدیران امکان کنترل بیشتر و توانایی سفارشی‌سازی گزارش‌های خود را با انتخاب بخش‌هایی از تراکنش HTTP (هدرهای، وضعیت، اندازه‌ها و غیره) در گزارش‌های خود می‌دهند.

امکان پیکربندی فرمتهای سفارشی برای مدیران اضافه شد زیرا پیش‌بینی اینکه مدیران علاقه‌مند به دریافت چه اطلاعاتی از گزارش‌های خود هستند دشوار است. بسیاری از پروکسی‌ها و سرورهای دیگر نیز توانایی انتشار گزارش‌های سفارشی را دارند.

Squid Proxy Log Format

به یکی از پروژه‌های وب اولیه (Proxy Cache) Squid Cache (http://www.squid-cache.org) Squid Proxy Cacheftp://ftp.cs.colorado.edu/ (pub/techreports/schwartz/Harvest.Conf.ps.Z) بر می‌گردد. Squid یک پروژه متن باز است که در طول سال‌ها توسط جامعه منبع باز گسترش یافته و ارتقا یافته است. ابزارهای زیادی برای کمک به مدیریت برنامه Squid نوشته شده‌اند، از جمله ابزارهایی برای کمک به پردازش، ممیزی و استخراج گزارش‌های آن. بسیاری از Squid های بعدی فرمت Squid را برای لاغهای خود به کار گرفتند تا بتوانند از این ابزارها استفاده کنند.

فرمت ورودی Squid log نسبتاً ساده است. زمینه‌های آن در جدول زیر خلاصه شده است.





Field	Description
timestamp	The timestamp when the request arrived, in seconds since January 1, 1970 GMT.
time elapsed	The elapsed time for request and response to travel through the proxy, in milliseconds.
host ip	The IP address of the client's (requestor's) host machine.
result code/status	The result field is a Squid-ism that tells what action the proxy took during this request ^a ; the code field is the HTTP response code that the proxy sent to the client.
size	The length of the proxy's response to the client, including HTTP response headers and body, in bytes.
method	The HTTP method of the client's request.
url	The URL in the client's request. ^b
rfc931 ident ^c	The client's authenticated username. ^d
hierarchy/from	Like the route field in Netscape formats, the hierarchy field tells what route the proxy used to make the request for the client. ^e The from field tells the name of the server that the proxy used to make the request.
content type	The Content-Type of the proxy response entity.

مثال زیر مثالی از یک ورودی Squid Log Format را نشان می‌دهد.

99823414 3001 209.1.32.44 TCP_MISS/200 4087 GET http://www.joes-hardware.com - DIRECT/ proxy.com text/html

فیلدها به صورت زیر تخصیص داده می‌شوند:





Field	Value
timestamp	99823414
time elapsed	3001
host ip	209.1.32.44
action code	TCP_MISS
status	200
size	4087
method	GET
URL	http://www.joes-hardware.com
RFC 931 ident	-
hierarchy	DIRECT ^a
from	proxy.com
content type	text/html

^a The DIRECT Squid hierarchy value is the same as the DIRECT route value in Netscape log formats.

جدول زیر کدهای مختلف نتیجه Squid را فهرست می کند.





Action	Description
TCP_HIT	A valid copy of the resource was served out of the cache.
TCP_MISS	The resource was not in the cache.
TCP_REFRESH_HIT	The resource was in the cache but needed to be checked for freshness. The proxy revalidated the resource with the server and found that the in-cache copy was indeed still fresh.
TCP_REF_FAIL_HIT	The resource was in the cache but needed to be checked for freshness. However, the revalidation failed (perhaps the proxy could not connect to the server), so the “stale” resource was returned.
TCP_REFRESH_MISS	The resource was in the cache but needed to be checked for freshness. Upon checking with the server, the proxy learned that the resource in the cache was out of date and received a new version.
TCP_CLIENT_REFRESH_MISS	The requestor sent a Pragma: no-cache or similar Cache-Control directive, so the proxy was forced to fetch the resource.
TCP_IMS_HIT	The requestor issued a conditional request, which was validated against the cached copy of the resource.
TCP_SWAPFAIL_MISS	The proxy thought the resource was in the cache but for some reason could not access it.
TCP_NEGATIVE_HIT	A cached response was returned, but the response was a negatively cached response. Squid supports the notion of caching errors for resources—for example, caching a 404 Not Found response—so if multiple requests go through the proxy-cache for an invalid resource, the error is served from the proxy cache.
TCP_MEM_HIT	A valid copy of the resource was served out of the cache, and the resource was in the proxy cache’s memory (as opposed to having to access the disk to retrieve the cached resource).
TCP_DENIED	The request for this resource was denied, probably because the requestor does not have permission to make requests for this resource.
TCP_OFFLINE_HIT	The requested resource was retrieved from the cache during its <i>offline</i> mode. Resources are not validated when Squid (or another proxy using this format) is in offline mode.
UDP_*	The UDP_* codes indicate that requests were received through the UDP interface to the proxy. HTTP normally uses the TCP transport protocol, so these requests are not using the HTTP protocol. ^a
UDP_HIT	A valid copy of the resource was served out of the cache.
UDP_MISS	The resource was not in the cache.
UDP_DENIED	The request for this resource was denied, probably because the requestor does not have permission to make requests for this resource.
UDP_INVALID	The request that the proxy received was invalid.
UDP_MISS_NOFETCH	Used by Squid during specific operation modes or in the cache of frequent failures. A cache miss was returned and the resource was not fetched.
NONE	Logged sometimes with errors.
TCP_CLIENT_REFRESH	See TCP_CLIENT_REFRESH_MISS.
TCP_SWAPFAIL	See TCP_SWAPFAIL_MISS.
UDP_RELOADING	See UDP_MISS_NOFETCH.

^a Squid has its own protocol for making these requests: ICP. This protocol is used for cache-to-cache requests. See <http://www.squid-cache.org> for more information.





Hit Metering

سرورهای مبدأً اغلب گزارش‌های دقیق را برای اهداف صورتحساب نگه می‌دارند. ارائه‌دهندگان محتوا باید بدانند که هر چند وقت یکبار به URL‌ها دسترسی پیدا می‌کنند، تبلیغ‌کنندگان می‌خواهند بدانند تبلیغاتشان چند بار نشان داده می‌شود و نویسنده‌گان وب می‌خواهند بدانند محتوای آن‌ها چقدر محبوب است. هنگامی که کلاینت‌ها مستقیماً از سرورهای وب بازدید می‌کنند، ورود به سیستم برای ردیابی این موارد خوب عمل می‌کند.

با این حال، Cache‌ها بین کلاینت‌ها و سرورها قرار می‌گیرند و مانع از دسترسی بسیاری از سرورها می‌شوند (هدف Cache). از آنجایی که Cache بسیاری از درخواست‌های HTTP را مدیریت می‌کند و آن‌ها را بدون بازدید از سرور اصلی برآورده می‌کند، سرور هیچ سابقه‌ای مبنی بر دسترسی کلاینت به محتوای آن، ایجاد حذفیات در فایل‌های گزارش، ندارد.

داده‌های گزارش از دست رفته باعث می‌شود ارائه‌دهندگان محتوا برای مهمترین صفحات خود به حذف Cache متولّش شوند. مخدوش کردن Cache به تولید کننده محتوا اشاره دارد که عمداً محتوای خاصی را غیرقابل ذخیره می‌کند، بنابراین همه درخواست‌ها برای این محتوا باید به سرور مبدا بروند. این به سرور مبدا اجازه می‌دهد تا دسترسی را ثبت کند. Defeating Caching ممکن است گزارش‌های بهتری را به همراه داشته باشد، اما درخواست‌ها را کند می‌کند و بار روی سرور و شبکه اصلی را افزایش می‌دهد.

از آنجایی که Proxy Cache‌ها (و برخی از کلاینت‌ها) گزارش‌های خود را نگه می‌دارند، اگر سرورها بتوانند به این گزارش‌ها دسترسی داشته باشند - یا حداقل روشی خام برای تعیین اینکه محتوای آن‌ها توسط یک Proxy Cache چند بار ارائه می‌شود - از تخریب Cache جلوگیری می‌شود. پروتکل Hit Metering پیشنهادی، یک Cache برای HTTP، راه حلی را برای این مشکل پیشنهاد می‌کند. پروتکل Hit Metering به همراه Extension‌ها نیاز دارد تا به صورت دوره‌ای آمار دسترسی Cache را به سرورهای مبدا گزارش دهنند.

RFC 2227 پروتکل Hit Metering را با جزئیات تعریف می‌کند.

Overview

پروتکل Hit Metering یک Extension برای HTTP تعریف می‌کند که چند تسهیلات اساسی را فراهم می‌نماید که Cache‌ها و سرورها می‌توانند برای اشتراک‌گذاری اطلاعات دسترسی و تنظیم تعداد دفعات استفاده از منابع ذخیره‌شده، پیاده‌سازی کنند.

Hit Metering، از نظر طراحی، راه حل کاملی برای مشکلات logging Cache برای این است، اما وسیله‌ای اساسی برای به دست آوردن معیارهایی که سرورها می‌خواهند ردیابی کنند، فراهم می‌کند.





پروتکل Hit Metering به طور گسترده اجرا یا مستقر نشده است (و ممکن است هرگز اجرا نشود). با این اوصاف، یک طرح مشارکتی مانند Hit Metering نوید ارائه آمار دسترسی دقیق را در عین حفظ دستاوردهای عملکرد Cache دارد. امیدواریم که این انگیزه‌ای برای پیاده سازی پروتکل Hit Metering به جای علامت گذاری محتوای غیرقابل ذخیره باشد.

The Meter Header

اضافه کردن یک هدر جدید به نام Cache را پیشنهاد می‌کند که می‌تواند از آن برای ارسال دستورالعمل‌های مربوط به استفاده و گزارش‌دهی به یکدیگر استفاده کنند، دقیقاً مانند هدر Cache-Control که اجزه می‌دهد دستورالعمل‌های ذخیره‌سازی را مبادله کنند.

جدول زیر دستورالعمل‌های مختلف را تعریف می‌کند و مشخص می‌کند که چه کسی می‌تواند آن‌ها را در هدر ارسال کند Meter

Directive	Abbreviation	Who	Description
will-report-and-limit	w	Cache	The cache is capable of reporting usage and obeying any usage limits the server specifies.
wont-report	x	Cache	The cache is able to obey usage limits but won't report usage.
wont-limit	y	Cache	The cache is able to report usage but won't limit usage.
count	c	Cache	The reporting directive, specified as "uses/reuses" integers—for example, ":count=2/4". ^a
max-uses	u	Server	Allows the server to specify the maximum number of times a response can be used by a cache—for example, "max-uses=100".
max-reuses	r	Server	Allows the server to specify the maximum number of times a response can be reused by a cache—for example, "max-reuses=100".
do-report	d	Server	The server requires proxies to send usage reports.
dont-report	e	Server	The server does not want usage reports.
timeout	t	Server	Allows the server to specify a timeout on the metering of a resource. The cache should send a report at or before the specified timeout, plus or minus 1 minute. The timeout is specified in minutes—for example, "timeout=60".
wont-ask	n	Server	The server does not want any metering information.

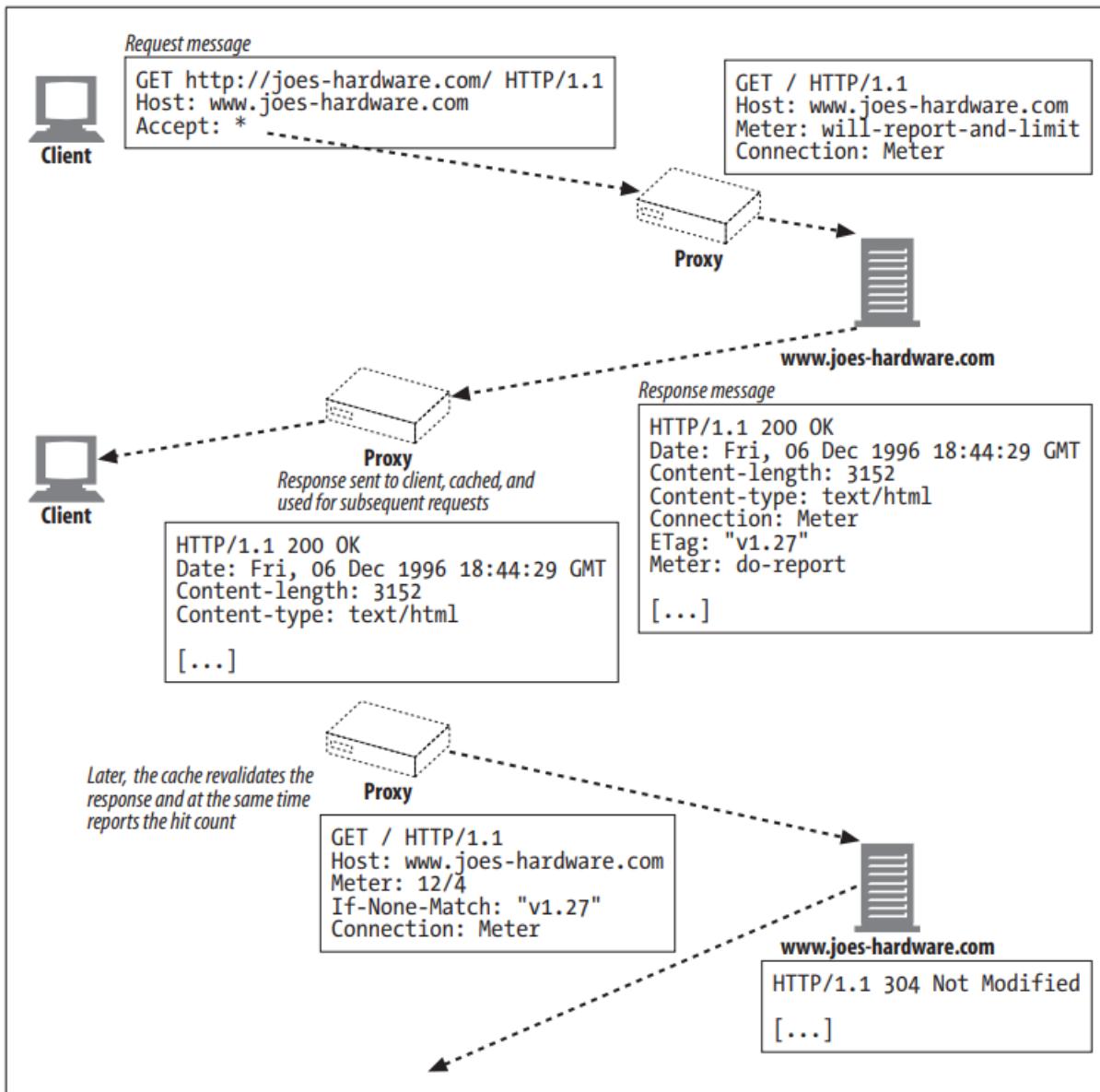
^a Hit Metering defines a *use* as satisfying a request with the response, whereas a *reuse* is revalidating a client request.

شکل زیر نمونه‌ای از Hit Metering را در عمل نشان می‌دهد. بخش اول تراکنش فقط یک تراکنش معمولی HTTP بین Cache مشتری و پراکسی است، اما در درخواست پراکسی، به درج هدر





و پاسخ سرور توجه کنید. در اینجا، پروکسی به سرور اطلاع می‌دهد که می‌تواند Hit Metering را انجام دهد، و سرور نیز به نوبه خود از پروکسی می‌خواهد تا تعداد ضربه‌های خود را گزارش دهد.



از دیدگاه کلاینت، درخواست همانطور که معمولاً انجام می‌شود تکمیل می‌گردد و پروکسی از طرف سرور شروع به ردیابی بازدیدهای آن منبع می‌کند. بعداً، پروکسی سعی می‌کند منبع را مجدداً با سرور تأیید کند. پروکسی اطلاعات اندازه گیری شده‌ای را که ردیابی کرده است در درخواست مشروط به سرور جاسازی می‌کند.

A Word on Privacy

از آنجا که Logging واقعاً یک عملکرد مدیریتی است که سرورها و پراکسی‌ها انجام می‌دهند، کل عملیات برای کاربران شفاف است. اغلب، وی ممکن است حتی از ثبت تراکنش‌های HTTP خود آگاه





نباشند - در واقع، بسیاری از کاربران احتمالاً حتی نمی‌دانند که از پروتکل HTTP هنگام دسترسی به محتوا در وب استفاده می‌کنند.

توسعه دهنده‌گان و مدیران برنامه‌های کاربردی وب باید از پیامدهای ردیابی تراکنش‌های HTTP کاربر آگاه باشند. بر اساس اطلاعاتی که کاربر بازیابی می‌کند، می‌توان چیزهای زیادی در مورد کاربر به دست آورد. بدینهی است که از این اطلاعات می‌توان استفاده بدی کرد - تبعیض، آزار و اذیت، باج گیری، و غیره مواردی است که در این خصوص متصور خواهد بود. سرورهای وب و پروکسی‌هایی که فرآیند Logging را انجام می‌دهند باید در حفاظت از حریم خصوصی کاربران نهایی خود هوشیار باشند.

مدیران همچنین باید این واقعیت را به اطلاع عموم برسانند که تراکنش‌های افراد تحت نظرات است.

به طور خلاصه، Logging ابزار بسیار مفیدی برای مدیر و توسعه‌دهنده است—فقط از نقض حریم خصوصی که لاغ‌ها می‌توانند بدون اجازه یا اطلاع کاربرانی که اقدامات آن‌ها ثبت می‌شود، آگاه باشید.

For More Information

<http://httpd.apache.org/docs/logs.html>

<http://www.squid-cache.org/Doc/FAQ/FAQ-6.html>

<http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format>

<http://www.w3.org/TR/WD-logfile.html>

<http://www.ietf.org/rfc/rfc2227.txt>

