B6: GET /started/with/ HTTP Analysis

Robert Bullen
Application Performance Engineer
Blue Cross Blue Shield of Minnesota
robert_d_bullen@bluecrossmn.com
The BCBSMN Experience

- Who is Blue Cross Blue Shield of Minnesota?
  - We are the first “Blue” health plan in the nation & the largest in Minnesota.
  - We have 2.6 million members across all 50 states and 3,500 employees.
  - Our administrative costs are less than 10 cents on the dollar, among lowest in the country.

- What do I do there?
  - I perform deep dive packet analysis for a few reasons:
    - To comprehend application functionality for modeling in our APM solution.
    - Troubleshooting.
    - Troubleshooting.
    - Troubleshooting.
  - I co-architect, implement, and administer our Shared Visibility Fabric (SVF).
  - I implement and administer our packet capture appliances.
  - I code in “down” time.
HTTP Is...

- Simple
  - It is stateless.
  - It is a ping/pong request/response protocol (ignore pipelining).
  - It uses human-readable requests, responses, headers, and sometimes payloads.
HTTP Is...

- Distributed/Multitiered
  - Services can be load balanced.
  - Connections can be forward and/or reverse proxied.
  - Static content can be separated and cached in a different tier from dynamic content.
    - Content can be localized through a CDN.
  - Resources can be redirected (e.g. URL shrinkers rely on this).
  - Applications might be composites that pull from multiple sites.
HTTP Is…

- Flavored
  - HTTP 1.0
  - HTTP 1.1 (this is the important one)
  - WebSockets (sorta)
  - SPDY/HTTP 2.0
HTTP Is…

- Ubiquitous
  - Web and application servers serving HTML.
  - Middle tier application servers publishing SOAP services.
  - Back-end SOA buses accepting SOAP/XML calls as a façade to legacy services.
  - Internet RESTful APIs to database-like resources.
  - Clients and servers are readily available as standalone programs or as libraries in most programming/scripting languages.
HTTP Is…

- Complex
  - Applications can utilize cookies or HTML hidden fields for statefulness
  - Applications can add caching for performance
  - Applications can add concurrency for throughput
  - Applications can choose to encode content:
    - Compressed (Content-Encoding)
    - Chunked (Transfer-Encoding)
  - More and more often encryption using SSL/TLS is in place at every tier (a.k.a. HTTPS)
    - Analysis gets trickier but is still possible.
    - Remember all those distributed/multitier hops? You’ll need keys for each of those tiers you wish to analyze.
HTTP Is...
URLs

http://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose
HTTP Request Methods

- Three most common:
  - GET
    - Requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
    - I equate this to a deterministic, non-modifying function (idempotent).
  - POST
    - Requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, as examples, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.
    - I like to think of this as a nondeterministic, modifying procedure invocation.
  - CONNECT
    - Instructs an intermediate proxy to create a tunnel to the remote host.

- Others:
  - HEAD, PUT, DELETE, TRACE, OPTIONS, PATCH
HTTP Response Status Codes

- **1xx—Informational**
  - 100 Continue—The request header is valid and the client may proceed with sending the request payload.

- **2xx—Successful**
  - 200 OK—Need I say more?
  - 202 Accepted—The request has been queued; check back later.

- **3xx—Redirection**
  - 302 Found—The requested resource has been temporarily moved and the browser should issue a request to the URL supplied in the Location response header.
  - 304 Not Modified—The requested resource has not been modified and the browser should read from its local cache instead.

- **4xx—Client Error**
  - 401 Unauthorized—Anonymous clients are not authorized to view the requested content and must provide authentication information in the WWW-Authenticate request header.
  - 404 Not Found—The requested resource does not exist on the server.

- **5xx - Server Error**
  - 500 Internal Server Error—Oftentimes this is the result of an uncaught exception (i.e. an unexpected and unhandled condition or a system error such as out of memory).
HTTP Is Layer 7

Layer 7—Application
Layer 6—Presentation
Layer 5—Session
Layer 4—Transport
Layer 3—Network

HTTP
SSL (records)
TCP (segments)
IP (fragments)
SSL Decryption

- You must be in possession of the private key.
  - Wireshark supports PEM or PKCS#12 format. I wrote a paper covering terminology, key file formats, and extracting private keys from those file formats, which you can download at http://goo.gl/w2r7kt.
  - The negotiated cryptography algorithm must not be Diffie-Hellman.

- You must configure Wireshark with server:port to private keys mappings.

- The client key exchange must be present in the capture.
  - The client key exchange occurs during the SSL handshake.
  - Rarely you may see a client and server renegotiate in the middle of an established connection.
  - SSL has a performance optimization called session caching where a client and server can reuse previously agreed upon session keys from different conversations.
## URL Redirection

http://www.hanselman.com/blog/ThisURLShortenerSituationIsOfficiallyOutOfControl.aspx

<table>
<thead>
<tr>
<th>#</th>
<th>Result</th>
<th>Protocol</th>
<th>Host</th>
<th>URL</th>
<th>Body</th>
<th>Cache</th>
<th>Content-Length</th>
<th>Process</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>301</td>
<td>HTTP</td>
<td>slate.me</td>
<td>/1h0svt8</td>
<td>120</td>
<td>privat</td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>301</td>
<td>HTTP</td>
<td>slate.trib.al</td>
<td>/8OfbdvM</td>
<td>101</td>
<td></td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>301</td>
<td>HTTP</td>
<td>slate.me</td>
<td>/1tByJQz</td>
<td>120</td>
<td>privat</td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>301</td>
<td>HTTP</td>
<td>slate.trib.al</td>
<td>/IOYnwof</td>
<td>101</td>
<td></td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>301</td>
<td>HTTP</td>
<td>slate.me</td>
<td>/1kZ76jq</td>
<td>120</td>
<td>privat</td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>301</td>
<td>HTTP</td>
<td>slate.trib.al</td>
<td>/QjWEhrI</td>
<td>95</td>
<td></td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>301</td>
<td>HTTP</td>
<td>goo.gl</td>
<td>/qF0xUk</td>
<td>323</td>
<td>no-cache</td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>200</td>
<td>HTTP</td>
<td><a href="http://www.slate.com">www.slate.com</a></td>
<td>/blogs/future_tense/...</td>
<td>175</td>
<td></td>
<td>text/html</td>
<td>fiddler</td>
<td></td>
</tr>
</tbody>
</table>
The Waterfall Diagram

- The Waterfall diagram is the best way to start analyzing single client web page performance.
- All the major browsers now come with debugging tools baked right in (“F12” tools) that present a waterfall diagram of (among many other things).
- Third party tools are also available:
  - HTTP Watch—”HTTP Sniffer” ([http://httpwatch.com](http://httpwatch.com))
  - Fiddler—”Web application debugging proxy” ([http://www.telerik.com/fiddler](http://www.telerik.com/fiddler))
- The information gathered by “F12” tools can be saved to an HTTP archive (HAR) file.
- A Python script called pcap2har ([https://github.com/andrewf/pcap2har](https://github.com/andrewf/pcap2har)) can be used to convert PCAPs to HAR files.
Reverse Proxies & Client Identification

1. Client ephemeral port propagation
2. X-Forwarded-For header
3. Proxy-added cookie header (encoded)
4. Payload matching
Reverse Proxies & Client Identification

1. Client ephemeral port propagation
2. X-Forwarded-For header
3. Proxy-added cookie header (encoded)
4. Payload matching
XFF, BIGIP

- X-Forwarded-For: 192.168.1.1
- BIGipServerLive_pool=375537930.544.0000
  - Decoded: IP Address: 10.65.98.22 Port: 34
Top Performance Bottlenecks

- HTML Content
  - Improper caching of static objects
  - Requiring authentication for every object on a page

- Client/Server Configuration
  - Low concurrency
  - Poor TCP connection reuse
  - Poor SSL session caching

- Busy server
  - High think time
  - High response transmission time (mid-stream delays)

- Intermediate Devices
  - HTTP proxies or WAFs introducing latency
  - Load balancer challenges
    - Unsynchronized object tags on pool servers
    - Client port collisions
Resources

- HTTP Introduction—http://www.httpwatch.com/httpgallery/
- SSL Analysis—
  http://sharkfest.wireshark.org/sharkfest.09/AU2_Blok_SSL_Troubles_hooting_with_Wireshark_and_Tshark.pps