Wormhole
An Active HTTP-Tunnel for High-Latency Networks
Konrad Miller <miller@kit.edu>

Motivation
High-latency networks (UMTS, VPN, TOR) have recently become very popular due to mobility, privacy, and anonymity considerations. Browsing the www over high-latency networks is frustrating as the underlying protocols are not designed to work well in such a scenario.

Problem Analysis
- Today's webpages contain embedded objects (DOM-tree/page requisites)
- These may recursively embed objects
- Embedded css file may embed images
- Objects are typically spread across multiple domains
  - Static images, scripts, advertisements on different domains
  - 20 most popular Alexa Top 500 pages: 443 domain names
- Fetching requires the following sequential steps per domain
  - Domain name resolution 1 round-trip
  - TCP connection setup 1 round-trip
  - Get "index.html" file 1 round-trip
  - Get page requisites (mult. connections, keep-alive, pipelining)

Network:
- Campus: 12 ms
- DSL: 27 ms
- UMTS: 326 ms
- TOR: 1228 ms

- Interpretation of DOM-tree is done in the client browser
  - Hinders parallelization (e.g., "index.html" needs to be interpreted to convey embedded contents)
  - Aggravates high-latency problem (stop-and-wait behavior)
  - Leads to poor bandwidth utilization
  - Causes a large fraction of the total delay

Initial part of an HTTP GET request with a RTT of 300ms

Proposed Solution
- Active HTTP-Tunnel
  - Wormhole entry at high-latency, low-bandwidth link
  - Wormhole exit at low-latency, high-bandwidth link
  - Wormhole entry acts as a web proxy for the browser
    - Passes browser queries through the tunnel
    - Serializes all traffic through a single TCP connection
    - Keeps connection alive between requests
  - Wormhole exit fetches and parses objects
    - Resolves all domain names
    - Returns object data to Wormhole entry
    - Piggybacks a list of page requisites that will be pushed subsequently
  - Wormhole entry can hold back future requests for announced contents until the data arrives unsolicitedly

Server push vs. client cache
- Wormhole exit is unaware of browser cache’s state
- Redundant data is pushed to the Wormhole entry
- Wormhole implements a self synchronizing cache to mitigate this effect
  - Entry caches received objects
  - Exit has knowledge of entry’s cache contents
  - Object hash is saved on exit-side instead of full object
  - Only an index into the cache needs to be transferred for a cache-hit

Evaluation
- Implementation with C++/Qt4
- Measurement with vanilla Mozilla Firefox and tcpdump
- Link traffic shaping with tc (netem, htb)

Initial Results
- Latency is reduced greatly

<table>
<thead>
<tr>
<th>Cache</th>
<th>No Proxy</th>
<th>Wormhole</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX: cold</td>
<td>190.4 Kib</td>
<td>48.5 Kib</td>
<td>0.25</td>
</tr>
<tr>
<td>RX: cold</td>
<td>1089.1 Kib</td>
<td>1049.3 Kib</td>
<td>0.96</td>
</tr>
<tr>
<td>TX: hot</td>
<td>49.2 Kib</td>
<td>21.7 Kib</td>
<td>0.44</td>
</tr>
<tr>
<td>RX: hot</td>
<td>148.0 Kib</td>
<td>196.8 Kib</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Future Work/Next Steps
- Thorough evaluation (e.g., scalability of Wormhole exit)
- Compare different scenarios
  - No proxy, local proxy, remote proxy
  - Cold, warm, hot caches; different caching parameters
  - Compression on and off
  - Different latencies/data rates