Wormhole – An Active HTTP Tunnel

Konrad Miller

System Architecture Group
Karlsruhe Institute of Technology, Germany
miller@kit.edu

Abstract

Browsing the world wide web over a high-latency network connection is frustrating. We propose an “active” HTTP Tunnel, *Wormhole*, to significantly reduce webpage load times in such a scenario.

1 Motivation

Advances in laptop and smartphone technology as well as in so called 3G networks have made it possible to be connected to the Internet at all times. Probably the most frequently used network application on such systems is the World Wide Web (WWW). Commonly used networks such as UMTS and HSDPA offer high data throughput, but are unfortunately prone to high latencies.

The protocols used most widely for browsing the WWW today are HTTP over TCP. Those protocols were designed in a time when webpages consisted of only a single text file. They are far from optimal for today’s webpages often requiring tens to hundreds of files, e.g., js, css, and images, for proper display.

Fetching a webpage can be split into at least four steps:

Domain name resolution: one UDP roundtrip, the mapping may be cached for future requests

TCP connection setup: one roundtrip, as the ACK packet may piggyback data

Get index file: one roundtrip, cannot be parallelized with page requisites

Get page requisites: fetch *page requisites*, supplementary files needed to display the webpage. Usually over 6-8 parallel connections, possibly revealing further requisites (e.g. via css, js)

Using TCP as a transport protocol in such a way is problematic: TCP’s slowstart mechanism hinders the utilization of the full bandwidth for many roundtrips leading to major slowdowns if roundtrips take a long time [3]. When fetching only one small file per connection, the aggregated latencies introduced by connection setups and slow-starts are significant; the user cannot profit from the available high bandwidth.

2 Related Work

Techniques to mitigate this and similar problems have been developed and integrated into the HTTP standard in the past: keep-alive [4] saves many roundtrips by reusing existing connections for multiple files and pipelining [5] reduces idle periods which are due to the stop-and-wait behavior of keep-alive once the connection is established. However, the implemented improvements do not solve all of the existing problems: The client still has to resolve the domain name, and to wait for TCP connection setup following the slowstart mechanism. This procedure leads to wasted bandwidth right after initiating an HTTP request and thus to longer webpage load times.

Other proposed improvements include changing the transport protocol to UDP, which leads to problems due to the lack of congestion control [6]. Google recently suggested the session layer protocol Spdy [2], which introduces the possibility for servers to push content to clients. Spdy does not address the DNS and TCP slowstart problems and is wasting bandwidth by repeatedly retransferring data, rendering the web-caching mechanisms ineffective. Furthermore, it requires modification of existing server and client software.

3 Our Approach

We propose to use an “active” HTTP Tunnel through the high-latency network as a solution to the problems stated above. One tunnel endpoint, *WormholeEntry*, resides on the browser’s side, possibly on the same node as the browser itself. The other tunnel endpoint, *WormholeExit*, resides on the far side of the high-latency network. The *WormholeEntry* acts as an HTTP proxy for the web client.

All communication is serialized through a persistent TCP connection between *WormholeEntry* and *WormholeExit*. This connection may be established before the first webpage is requested, so the connection setup does not play a role for the user-perceived latency when fetching a webpage.

DNS look-ups are not performed by the client directly anymore. Instead, the HTTP request including the headers is sent to the *WormholeExit*, which performs domain name resolution, connects to the target webserver, and fetches the requested data. Before sending the content and return-headers back to the *WormholeEntry*, the *WormholeExit* determines if the received content is a cascading style sheet or an HTML document. If so, it parses the file to discover further *page requisites*. A list of such page requisites is then appended to the reply. These links are also appended to a fetch queue and the data is unsolicitedly sent to the *WormholeEntry* as soon as it has been fetched by the *WormholeExit*. Each of them may in turn require additional objects, which are also announced to the *WormholeEntry* and added to the fetch queue recursively.

Receiving a reply, the *WormholeEntry* learns which page requisites will be sent shortly and delivers the received content to the client. When the client asks for a known page requisite, the request is not forwarded to the *WormholeExit* but blocked until the requested data has been unsolicitedly received.

The depicted protocol leads to redundant transfers of files which already reside in the client's cache if the webpage is refreshed or sub-pages with overlapping contents (e.g., same style sheet, images) are visited, since the *WormholeExit* is lacking information about data that is already present in the *WormholeEntry* or web client. To counter this effect, we propose two different mechanisms. Firstly, the *WormholeEntry* appends a "Date header" if it requests a URL which has been received before. This header is passed to the webserver, whose standard behavior is to reply with a "200 OK" status and the same date, omitting the actual data, if it has not changed. This in turn leads to no further requisites being found in the empty body. Secondly, a self-synchronizing cache is used to circumvent the retransmission of previously sent data. The cache contains hash values of data which has been sent before. When data needs to be retransmitted, an index into the cache is transmitted instead. In addition to these techniques, all messages passing through the tunnel are compressed using the gzip algorithm, to reduce the amount of transmitted data.

4 Initial Results and Conclusion

We have implemented *Wormhole* using C++/Qt4.5 and built a test-setup using a current Firefox browser in its default configuration.

Initial benchmarks against the first 20 webpages of the Alexa Top 500 list [1] show very promising results. Where standard HTTP over TCP seldomly used the available bandwidth, slowing down the webpage load times significantly, *Wormhole* uses all of the available bandwidth right after the first roundtrip time. The main open problem we have encountered was with JavaScript, which is not interpreted in the *WormholeExit* but may lead to additional page requisites.

Wormhole has currently only passed initial feasibility tests. A thorough evaluation remains to be done. We plan to benchmark the latency reduction in different scenarios, comparing it to different proxy configurations. A question which needs to be answered in the ongoing work is how much each of the techniques contributes, e.g., if the compression is worth the CPU cycles, or if the introduced cache is effective. Moreover, the work raises the question if the bandwidth saving techniques are surpassed by the regular caching mechanisms, whose effectivity has been limited by our approach. Further investigations must show how well *Wormhole* performs, and if the *WormholeExit* scales to a large number of *WormholeEntries* in realistic scenarios.

References

[1] Alexa top 500 sites, http://www.alexa.com, February 2010.

[2] Spdy: An experimental protocol for a faster web, http://dev.chromium.org/spdy, 2010.

[3] M. Allman, C. Hayes, H. Kruse, and S. Ostermann. TCP performance over satellite links. *5th International Conference on Telecommunication Systems*, 1997.

[4] T. Berners-Lee, R. Riedlding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol – HTTP/1.0, May 1996.

[5] R. T. Fielding, J. Gettys, and J. M. et al. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, June 1999.

[6] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Trans. Netw.*, 1999.