WarpTCP - Solutions

WHITE PAPER

Page Load Bottleneck

BADU networks

- Improving the way the world connects -

Troubleshooting for Sluggish Applications

Sluggish applications are a common problem faced by network engineers and IT staff. Proper diagnosis of the true bottleneck is necessary to solve the problem with maximum benefit and lowest cost. While this principle is well known, it takes a careful analysis and proper domain knowledge to uncover the real bottleneck. Often the real bottleneck is not discovered, and an expensive and inefficient solution is applied as a patch.

In most cases, to hunt down the bottleneck, numerous processes and components have to be carefully considered. Often the troubleshooter is biased by his past experiences and forgets to consider causes that he has not encountered before.

In this white paper, we describe a process to diagnose sluggish page loading in a multi-office corporation. This case came from a customer engagement with Badu's technical staff for troubleshooting.

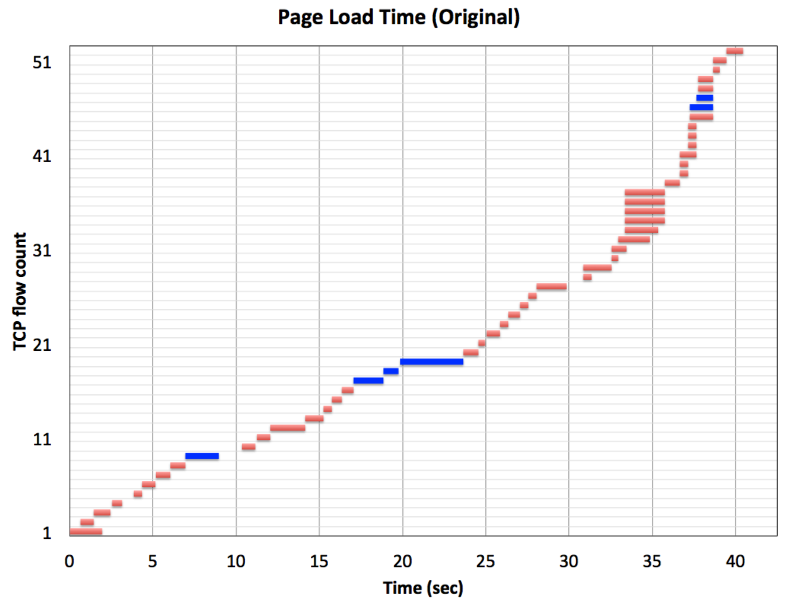
The company had a main office with servers that serve content to a number of overseas satellite offices. A slow page load problem was encountered in a satellite office for downloading a simple web page. The satellite office was located in a facility with unstable and large network RTT (round trip time). At the time of problem, downstream RTT from the main office varied between 400 ms and 800 ms. While the page contained only 52 separate small objects, it took more than 40 seconds to load the page. Therefore, the IT staff was called to solve the problem.

The IT staff quickly listed a number of possible causes

1. Competing traffic in the background
2. MTU/MSS issues (fragmentation)
3. Wi-Fi issues in the satellite office
4. DNS issues
5. Routing issues
6. Server issues
7. Client issues
8. TCP issues
9. Browser issues

After carefully checking through the list, the IT staff discovered that none of the items in the list had an issue that might have caused the page load time to go over 40 seconds. There was not much competing traffic in the background that might have caused congestion in the path. The MTU/MSS allowed 1342 bytes, which was expected for that environment. There were no TCP issues as each TCP session had only 3-5 data packets to transmit. The DNS was normal, as was the routing configuration. The server had plenty of resources to send data, while the client also had plenty of resources to make the requests and to receive data. The browser was operating normally. There were no Wi-Fi issues in the satellite office either.

After the Badu technical staff took a packet capture to study the TCP processes responsible for getting the page objects, this plot was obtained:



Altogether, there were 52 TCP flows. In the plot above, the blue bars indicate the server sending content data, while the red bars indicate the server sending no data. Out of the 52 HTTP requests, 5 had a 302 (data redirect) response, 6 had a 200 (successful data transfer) response, 40 had a 304 (data not modified) response, and 1 had a 404 (data not found) response.

HTTP staus code	Frequency	Percentage of flows
304	40	76.9%
200	6	11.5%
302	5	9.6%
404	1	1.9%

The distribution of parallelism in HTTP requests is given in this table.

Parallel HTTP requests	Frequency	Percentage
groups of 1	31	79.5%
groups of 2	5	12.8%
groups of 3	2	5.1%
groups of 5	1	2.6%

Out of the 52 HTTP requests, 21 (40.4%) of them were sent in parallel (with at least one other request at the same time), while 31 (59.6%) of them were sent strictly sequentially. None of HTTP requests that came back with content data was sent in parallel.

In the above table, a group of 2/3/5 parallel HTTP requests is counted as 1 parallel request. Then, the total number of parallel requests was 39, while about 80% of the requests were sent sequentially.

Therefore, the bottleneck of the page loading is the browser action of sending HTTP requests. When the download RTT is small, such a highly sequential request sequence is not noticeable by the user. However, in the current case, as the RTT is large (400 ms - 800 ms), the sequential request sequence caused the overall load time to be painfully long.

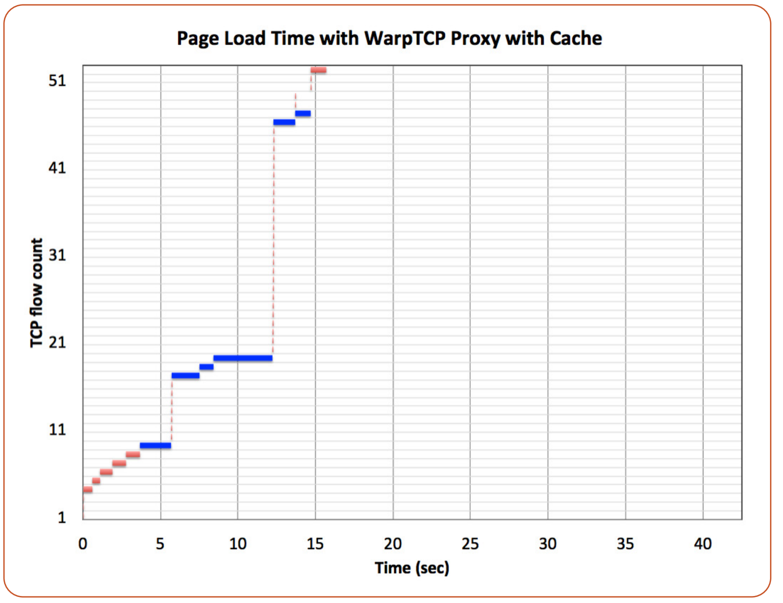
The highly sequential request sequence is a combined result of the HTML code and the browser implementation.

This is a bottleneck which is usually ignored by web developers. Web developers usually work in a networking environment where networking bottlenecks such as long RTT are absent from their QA testing.

Modern browsers do send parallel HTTP requests — however, most of them will send 2 requests or 4 requests in parallel at most. Unless the web developer makes a conscientious effort to parallelize the requests, the resulting download process can be highly sequential.

In the current example, the WAN bandwidth is obviously not the bottleneck. Upgrading the WAN bandwidth will have very little impact on the page load time.

Since 76.9% of the content is not modified when the data requests were sent, these data objects should be cached locally. Adding a *Warp*TCP proxy at the client site with cache option will drastically reduce the load time. The improved load time will be slightly more than 1/3 of the original load time, representing 2.5 times improvement.



Conclusion

Finding the real bottleneck is an art, as there are numerous potential trouble-causing sources and these sources can interact with one another in a nonlinear fashion.

Past experiences can cause biased opinions, while providing no guarantees for uncovering the real bottleneck. It is important not to make any assumptions.

Once the true bottleneck has been discovered, an effective solution with high improvement and low cost can be applied. The difference between knowing and not knowing the real bottleneck can be significant.

To request a demo, visit: **www.BADUnetworks.com**.

BADU
networks

ⓟ 949-310-5390
ⓕ 888-958-7697
ⓔ info@badunetworks.com

2640 Main Street Irvine CA 92614
http://www.BADUnetworks.com