Fastening of the data transmission in high latency satellite networks

Robert KRATZ[a] and Hubertus OSTERWIND
[a] *idea meets market Beteiligungsgesellschaft mbH*

Abstract. The internet onboard of ships is very slow and uncomfortable. This is caused by the high latency of the used satellite networks. Most internet connections are based on connection-oriented protocols, like TCP/IP, which require positive acknowledgments for each data packet sent. To fasten the data transfer, the use of a connectionless protocol without positive acknowledgments would solve the latency problem. But such a protocol, like UDP, is not secure enough nor supported by most of the web servers. The aim of the present development is therefore to provide a fast and secure data transmission in networks having a high latency. This object is solved by a connectionless data protocol having a parallel information channel between the client and a server which is independent of transmission of the payload. Said information channel is e.g. used for negative acknowledgements for lost packets or connection relevant information like the available bandwidth. Furthermore, to avoid lots of time consuming singular connections for each object of a webpage, a proxy server may be used that collects all relevant data of an external web server, renders the data in a headless browser, and only sends the completely rendered web-page to the client in one data stream.

Keywords. Ship digitalization, Transport protocol, Congestion control, UDP, High performance data transfer.

1. Introduction

The increase of network bandwidth has enabled data transfer at speed of gigabits per second. Computational and data grids have been among the first generation infrastructures to utilize the abundant network resources [1], followed by modern video platforms. Even though the network bandwidth is increasing, the usability of the network bandwidth theoretically available for the users should be improved. The problem is mainly based on the fact that the existing possible bandwidth to transport data is not fully used. This is based on a problem resulting from on the one hand sub-optimal application layer protocol HTTP combined with a transport layer protocol that is facing several own significant problems. In addition, the problems of the mainly used transportation layer protocol TCP increases with the network bandwidth and delay product (BDP), which is one of the problems in connections with high latency like via a satellite.

The other aspect is that HTTP is the most used application level protocol over the internet [2]. According to recent studies HTTP even itself will represent the bottle neck of the future internet [3]. Why HTTP will be one of the bottle will be discussed later on in section 2. Within the last years HTTP has experienced an extensive growth mainly fueled by the wide diffusion of HTTP-based infrastructure such as Content Distribution

Networks (CDNs), proxies, caches, and other middle-boxes. This growth is driven mainly by video content delivered using HTTP, which is today the main source of internet traffic.

Video over HTTP is the mainstream choice employed by all major video distribution platforms including the ones based on the recent MPEG-DASH, HLS standards, and the VoD systems such as Alphabet´s YouTube and Netflix [4].

Therefore, we are facing on the one hand the problem of a blocked up internet due to the increasing video traffic and therefore with lower available bandwidth and we still have significant unsolved problems in already now "bad quality" networks having a high BDP already now.

The problem of solving this objection is that it has to be focused on the one hand to the transportation layer, as the used TCP is facing serious challenges depending on the BDP, and on the other hand it had to be focused on the more and more important HTTP on the application layer, as HTTP itself is more part of the problem than the solution at present.

It has as well to be noted that TCP as connection-oriented protocol is providing services which goes beyond the pure transportation of data. TCP provides e.g. a congestion control and a loss recovery. The most obvious alternative for solving the BDP problem of TCP, the connection-less protocol UDP, does not provide such services itself. Connectionless protocols do not face the latency problems of connection-oriented protocols, but objectives solved by a connection oriented protocol like TCP will have to be solved in the application layer instead to provide a stable and secure connection.

Therefore, it will be a necessity to look on both protocols of the transportation and the application layer at once to solve the given objections of a more and more blocked network and the parts of said network already having a high BDP.

An important aspect is thereby to provide a feasible solution which can be easily implemented and used by both server and clients without significant changes on the overall structure of the internet.

2. Challenges of TCP and HTTP

At first it is of relevance to look at the data transport protocol to analyze whether the existing protocols could be optimized to overcome the problems cited above. Starting with the widely used TCP on the internet, TCP has exposed its inefficiencies as the network bandwidth and delay product (BDP) increases [5, 6]. The window based congestion control of TCP suffers in particular from its "slow start" mechanism in networks having a high latency like satellite connections or oversea connections. During the congestion avoidance phase, TCP increases the sending window approximately 1 byte per round trip time (RTT). It will take a long time for a TCP flow to occupy a high BDP link (e.g. satellite connection) or recover from a loss [7].

Moreover, due to this slow increase it cannot be guaranteed that the TCP flow will at the end claim the full available bandwidth in the long run. Random loss of the physical link prevents TCP from increasing and makes its sending window drop to half (and the subsequent need for several round trips to recover). Theoretical analysis has shown that to reach high speed over high BDP networks, TCP requires an extremely low link loss error rate, which is hard to archive for the current technology, in

particular in high latency networks [7]. The problem is worse over wireless networks such as satellite link, which has a much higher link error rate than wired networks.

What as well is of importance that since TCP increases its window per RTT, flows with shorter RTT will increase faster and cause unfairness between flows with different RTTs [8]. This unfairness cannot be ignored as it leads to serious performance degradation in a distributed computing application that involves both local nodes and remote nodes across the ocean. To give one example of such kind of applications is streaming join [5], whose performance is limited by the slowest data stream.

Therefore, there is a need to look at an alternative for TCP. TCP may still be a good choice in high performance networks having a low latency, but is not able to work in blocked networks or bad quality networks.

To search for a solution, there is a need to look not only at the transportation layer, but also on the application layer as described above. This is due to the fact that TCP has integrated functions as a connection oriented protocol like congestion control and lost recovery which must be provided by the application layer in case of a change to a different protocol. Furthermore, what is important regarding a blocking of the internet, more and more traffic is transported via HTTP.

If we look at HTTP, despite its widespread use, some inefficiencies of HTTP are hindering the development of a faster internet. To improve HTTP/1.1 Google that has proposed SPDY [11] within the IETF working group HTTPbis. SPDY is designed to overcome three of the main inefficiencies of HTTP; at first that fact that a HTTP client can only fetch one resource at a time, second, a client-server pull-based communication model is used; if a server knows that a client needs a resource, there is no mechanism to push the content to the client, and third, it redundantly sends several headers on the same channel.

HTTP/1.1 web browsers attempt to address these issues by opening multiple concurrent HTTP connections, while the HTTP/1.1 RFC suggests a limit of two concurrent connections for each client-server pair.

One disadvantage of SPDY is that an out-of-order packet delivery for TCP induces head of line blocking for all the SPDY streams multiplexed on that TCP connection. Moreover, SPDY connection startup latency depends on TCP handshake that requires one RTT and in the case SSL/TLS is employed up to three RTT.

Most of the inefficiencies mentioned above are caused by the use of TCP as transport protocol. Google is still aiming at reducing TCP handshake time [13] and, more recently in [14], at improving TCP loss recovery mechanism. Unfortunately, the improvements above-mentioned are not implemented in the default version of TCP.

As it seems hard to optimize TCP enough the closed alternative is to replace TCP by UDP as a connectionless protocol. This would require to integrated the missing TCP functionality into the application layer. To do so, Akamai has proposed a Hybrid HTTP and UDP content delivery protocol that is employed in its CDN [12].

As the goal of optimizing TCP was not as successful as expected, Google has as well started on an alternative to TCP called QUIC [14]. QUIC is as well based on UDP and aimed to replace HTTP over TCP. Google has already deployed QUIC in their servers, such as the ones powering YouTube. As well, QUIC can be activated in the Web client Chrome browser that runs on billions of devices. This puts Google in the position of driving a switch of a sizable amount of traffic from HTTP over TCP to QUIC over UDP.

QUIC incorporates congestion control and loss recovery features similar to TCP provided on the application layer. Moreover, QUIC decreases network latency by

offering fewer RTTs for the connection setup. QUIC as well provides the negotiation features of Transport Layer Security (TLS). At the end to further fasten the data transmission, QUIC also comprises features of SPDY such as multi-streaming without the problem of head of line blocking occurred in SPDY via TCP because all packets must be delivered in order.

Google at present is trying therefore for several years to replace the presents standard “HTTP via TCP” by “QUIC via UDP” and pushes this protocol into the market. Nevertheless, besides the use of QUIC on their own system world, QUIC has not been successful yet and is facing its own problems.

The most important problems of QUIC have been discovered by Carlucci et a. [15] in a comparison between HTTP, SPDY and QUIC. It was shown that QUIC has a goodtime of 1.5 over TCP in case of no data losses, but as soon as data losses occur the load time of web pages was even higher with QUICK than with UDP (loss rate of 2%). This was assumed to be caused by the fact that in the case of HTTP/1.1, the browser opens 6 parallel TCP connections and the effect of the random losses will be distributed among the 6 TCP flows, which has less impact than in the case of a 6 streams multiplexed over a single UDP connection. The learning of Carlucci is that the higher the bandwidth and/or the packet loss rate the lower is the advantage of QUIC over UDP over HTTP/1.1 over TCP and quickly the conventional protocols are faster.

This combination is in our view fatal. If we have fast networks, QUIC has no real advantage over TCP, as the advantages gets lower and lower as faster the connection is. If we have BDP of disadvantage such a poor connection goes along in most case, like satellite connections, with a high error loss rate. And we learned as well that in this case QUIC is even slower than TCP. That means that there seem to be no real use case for QUIC and this may be the background that even the complete power of Google is behind QUICK for years the system is not really used.

This background motivated us to design and develop a lightweight high performance hybrid data transportation and application level data transfer protocol which is faster than QUIC and is as reliable as HTTP over TCP.

3. HTTP over UDP via Onestreme

As we have discussed before, the use of a connectionless protocol would be in most cases more efficient than the use of connection-oriented protocols because this would solve the latency problem. We have decided to take an approach which is still using both UDP and HTTP as they are, but build up an intermediary protocol called ONESTREME. ONESTREME uses part of the UDP payload to include the necessary services that are included in TCP.

At the beginning, we do have one initial handshake protocol where the ONESTREME client gets an identifier of the server for a unique connection. This has the advantage that the client-server connection is independent of the IP address and a handover in a wireless network is easy to handle. We are therefore not losing the connection in case of a handover in a wireless network. The next is that we use the first handshake to estimate the available bandwidth and the maximum transfer unit (MTU) right at the beginning. Both parameters are adapted during the whole connection time dynamically, but based on this initial determination. Therewith it will be secured that we are not having a high packet loss rates from the beginning on, but we are starting with the right range.

For each file or object to be transferred we include an identifier for the type of data and a data pointer in the UDP packet payload. Therewith the receiver of the data learns on the fly how many packets should be received. The order of receiving is totally irrelevance, as we have initially and constantly adopted the relevant parameters like the latency and the bandwidth so the time until a re-transmission should be requested with the help of a negative acknowledgement is clear for each party. There is therefore no risk that a missing packet blocks the whole data transfer.

As well, the negotiation features of TLS for security reasons are included into ONESTREME being part of the state of the art.

We have as well learned that it is of high importance to split the data streams to several UDP ports. If we use just one UDP port, the speed of the data transfer is lower than by using several parallel UDP ports.

QUIC multiplexes the data over just one UDP port and is therefore in several cases slower than HTTP/1.1 over TCP using 6 parallel TCP ports for data transmission. We have made measurements with several test cases and identified that at least two parallel UDP ports have to be used to improve the transmission via TCP, see table 1:

Table 1. Showing the influence of a given latency on the data transmission using Onestreme compared to TCP (1x stream = 1 UDP port)

Scenario	Latency (ms)	1x stream (kbps)	2x stream	3x stream	TCP
1	15	710	1100	1500	1010
2	20	510	880	1020	815
3	25	390	520	810	500
4	50	200	395	550	370
5	75	140	270	400	280
6	100	90	170	260	175

As can be directly see in table 1 that using one UDP port in all cases is slower than using TCP, while starting with 2 TCP ports the data transmission is approximately equal. For this test we have tested four test scenarios:

1. Three types of photo galleries having 4, 20 and 50 pictures on it;
2. Three types of wiki pages with different complexity
3. File repository
4. News sites (small, medium, complex)

The results in table 1 shows the average measurement results. Therefore, ONESTREME is not only multiplexing the data like QUIC over one UDP port, but over several UDP ports. As well, we have learned that the bandwidth should be divided onto different ports with a maximum bandwidth of 6 mps per port to be able to be faster than TCP.

We therefore have learned and shown that the data transport could be improved over TCP in case of splitting the multiplexed UDP stream onto parallel UDP ports, while at least two UDP ports are needed to be equal to TCP and starting with three parallel ports so TCP connections could be improved in any case.

4. Onestreme and the Forward Proxy Cache

In the first step we have improved the data transmission as such over our ONESTREME protocol. But we have further improved the overall system especially in

high latency networks. The idea is to use a proxy server with a forward cache and a headless browser. The client asks the Onestreme server by sending the first request to provide a webpage and the Onestreme server fetches all relevant data, renders the webpage in a headless browser and only provides the already rendered webpage to the client.

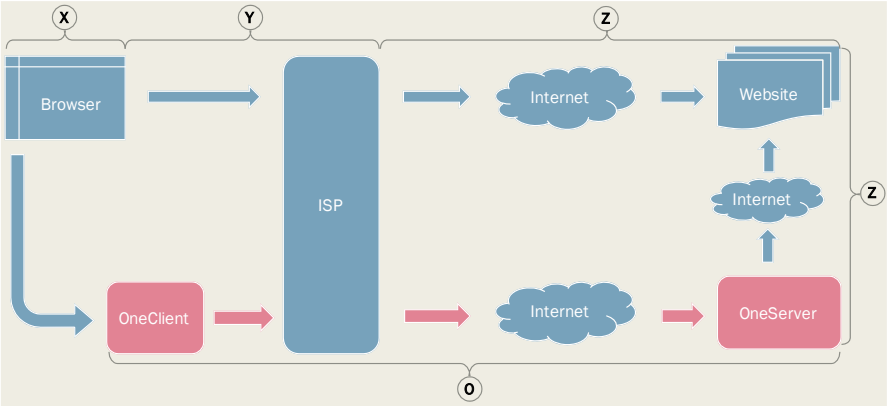

Figure 1. The assumption for using a proxy server to fasten the web traffic

The basic assumption is that the times X + Y + Z > X + O + Z. If we delete the not necessary parts; Y > O is the important remaining element.

This means that the use of a proxy server makes only sense if the network quality of the client is poor and the time the server needs to fetch all the data, render the webpage and send the same to the client is lower than the case the browser does it itself.

First test shows that in high latency networks we could significantly reduce the load time depending on the complexity of the web pages between the factor 2 to 8 in normal networks if we store pre-rendered pages onto our server while the time to fetch the data and render the webpages in real time results in slow down the speed. Networks having a latency up to 50ms seem not to be improved even by using a server directly located close to a central internet backbone.

The real-time rendering of web pages would therefore only make sense in high latency networks having a latency of 300ms or higher. Such a latency is measured by using satellite connections. Here we assume a great fastening.

As well, we use a cloned client cache on the Onestreme server. This means we are ensuring that only new data are transmitted to the client that reduces the overall amount of data significantly if the user is opening the same webpage more than ones.

Finally, we have stored an archive of already pre-rendered pages onto our Onestreme server, which is kept up to date, based on the usage of the users and the type of page. That means by using Onestreme we are always checking whether we do have an already archived pre-rendered version of the page that is requested on our server, if so, the transmission of the same would be faster between the factor 2 and 8 even in networks having a latency of 10 to 15 ms. If not, the Onestreme system is checking if it would be faster to retrieve the data via the server or not based on the data connection information that are known. If it is more efficient to use HTTP over TCP that way would be chosen automatically.

5. Conclusion

We have shown that the protocols used at present like HTTP over TCP are facing more and more problems. The increasing web traffic in particular caused by video streaming reduces the available bandwidth in at present excellent networks while the basic problems in networks with a high BDP are still unsolved. We have discussed the ideas of SPDY to improve TCP and as well the idea of Google to use QUIC over UDP.

The last approach of QUIC over UDP faces difficulties as QUIC multiplexes all data over one UDP port and seems not to improve the data transfer in case of a high bandwidth in a significant way and even slows down the transmission in case of higher packet loss rates that occur in particular in high latency networks.

We have shown that we could solve this problem with our ONESTREME protocol that is somewhere similar to QUIC but besides other differences split the multiplexed data stream onto several UDP ports to be faster than TCP under any circumstance. We could show that as soon as we are using at least three UDP ports in parallel we are about at least 50% faster than TCP and we could even more than double the bandwidth by using more UDP ports in parallel.

As well, one additional problem is that the complexity of web pages increase and more and more singular data packets has to be transferred. We proposed to use a forward proxy server to fetch all relevant data by a server located at a location having a high bandwidth and pre-render the web pages onto said server in a headless browser so that the user sends the HTTP request only and receives a completely rendered webpage back. By the use of a cloned client cache on the server we have been able to reduce the amount of data to be transferred even further as only completely new data is transmitted. As the result, depending on different test case, our ONESTREME solution is between 2 and 8 times faster (having a latency of 100ms at the maximum, it is expected to be higher with higher latency) than the conventional HTTP over TCP approach.

References

[1] Foster, C. Kesselman, S. Tuecke: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[2] Global Internet Phenomena Report, 2014. https://www.sandvine.com/trends/global-internet- phenomena/.

[3] L. Popa et al. Http as the narrow waist of the future internet. In Proc. of ACM SIGCOMM Workshop on HotNets, pages 6:1–6:6, Monterey, California, 2010.

[4] Cisco. Cisco Visual Networking Index:Forecast and Methodology 2013-2018. White Paper, June 2014.

[5] Y. Zhang, E. Yan, and S. K. Dao: A Measurement of TCP over Long-Delay Network, Proc. of 6th Intl. Conf. on Telecommunication Systems.

[6] W. Feng and P. Tinnakornsrisuphap: The Failure of TCP in High-Performance Computational Grids, Proc. of SuperComputing 2002.

[7] T. V. Lakshman and U. Madhow: The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. IEEE/ACM Trans. on Networking, vol. 5 no 3, July 1997, pp. 336-350.

[8] D. Katabi, M. Hardley, and C. Rohrs: Internet Congestion Control for Future High Bandwidth-Delay Product Environments, ACM SIGCOMM 2002, Pittsburg, PA.

[9] Floyd, S., Mahdavi, J., Mathis, M., and Podolsky, M.: An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883, Proposed Standard, July 2000.

[10] T. Dunigan, M. Mathis and B. Tierney: A TCP Tuning Daemon, Proc. of IEEE SuperComputing 2002.

[11] M. Belshe and R. Peon. Spdy protocol. IETF Draft, 2012.

[12] M. Ponec and A. Alness. Hybrid http and udp content delivery, Aug. 23 2013. US Patent App. 13/974,087.
[13] S. Radhakrishnan et al. Tcp fast open. In in Proc. of CoNEXT '11, pages 21:1–21:12, Tokyo, Japan, 2011.
[14] T. Flach et al. Reducing web latency: The virtue of gentle aggression. In Proc. of ACM SIGCOMM '13, Hong Kong.
[15] G. Carlucci et a. HTTP over UDP: an experimental investigation of QUIC, Proceedings of the 30th Annual ACM Symposium on Applied Computing, Pages 609-614, Salamanca, Spain 2015