Accelerating Data Transmission Rate of Transport Connections with Lightweight Proxying

Nikita Voynov
Lomonosov Moscow State University
Moscow, Russia
voynov@nikscorp.com

Evgeniy Stepanov
Lomonosov Moscow State University
Moscow, Russia
estepanov@lvk.cs.msu.su

Eugene Chemeritskiy
Lomonosov Moscow State University
Moscow, Russia
eugene.chemeritskiy@gmail.com

Introduction

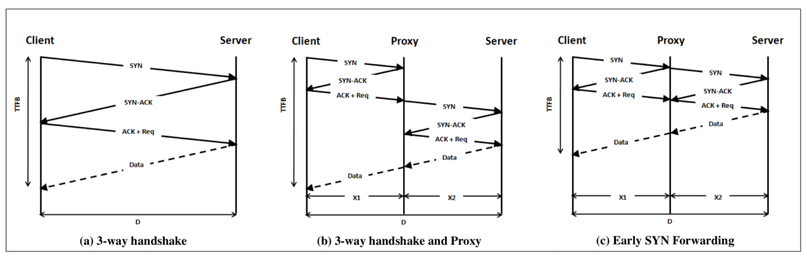
TCP congestion control mechanisms do not respond quickly enough to network environment changes on channels with large RTT. This problem is especially actual for short-lived connections, which duration comparable to the slow start duration.

The SplitTCP approach suggests splitting the connection into several consecutive ones with smaller RTTs. To implement such approach, a connection should be passed through one or more proxy servers. This solution allows to quickly retransmit lost packets and adjust the size of the congestion window.

However, this solution allocates additional resources (like open file descriptors, CPU time for flow control and congestion control for each connection). Thus, this paper is focused on the task to implement a lightweight proxying method which does not require establishing intermediate connections between proxy servers.

It may be undesirable to make kernel modifications on the proxy server (e.g. due to security policy restrictions) or even impossible (e.g. when using cloud-based solutions with container virtualization). Therefore, it is proposed to investigate the feasibility of implementing lightweight TCP connections proxying in the user space.

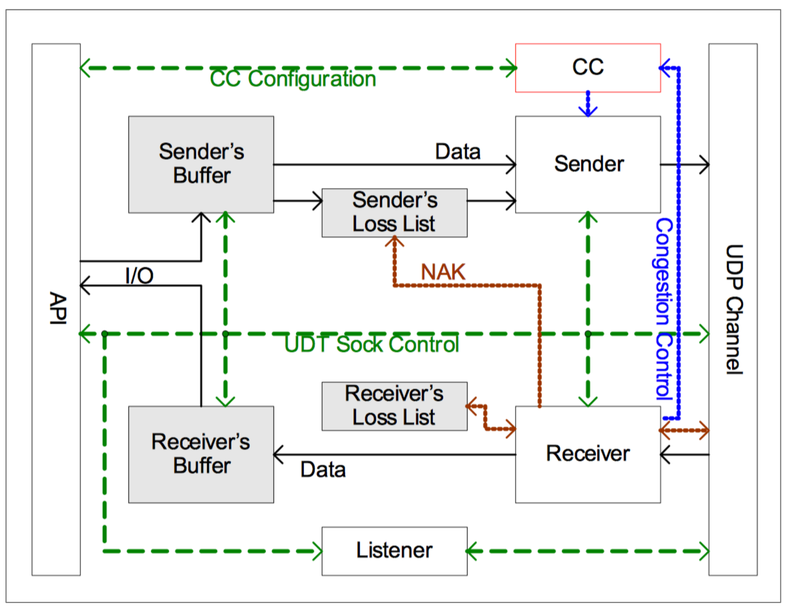
Protocol selection for inter-proxy communication



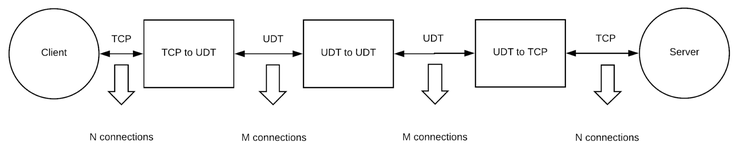
Despite the fact that Split TCP technology allows to accelerate data transfer compared to the standard TCP protocol, the connection establishment time is significantly increased, because 3-handshake is performed consecutively between each neighboring pair of proxies. The ESF approach allows to solve this problem by establishing connections between neighboring servers immediately after receiving the first SYN packet. However, this solution requires modification of the OS kernel. In addition, both the Split TCP approach and its optimization with ESF have above-said scaling problem, since each new connection requires the establishment of several additional ones between proxies. It is possible to solve this problem by using a reliable modification of the UDP protocol, without connection establishment, but it's difficult to implement congestion control in such solution.

Thus, we propose to use UDT protocol to transmit data between proxies, which establishes connection and provides all TCP facilities.

Moreover, UDT utilizes available bandwidth better than the classic TCP protocol and reduces the number of ACK packets transmitted over the network.Due to the fact that the UDT protocol is implemented in user space, it provides a number of features such as creating its own congestion control algorithm. Since proxy servers can be located on the boundaries of network segments with different characteristics, this feature is often in demand.



The proposed method



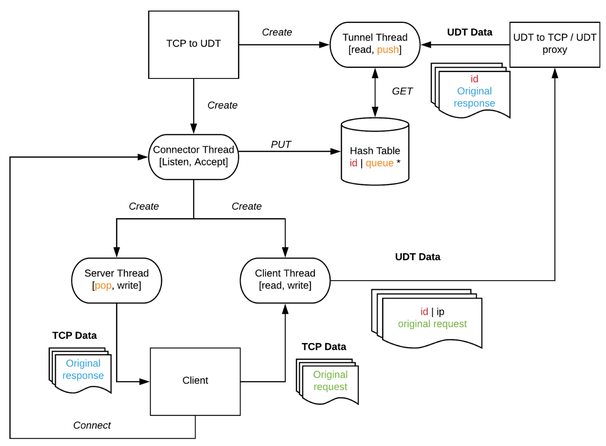
The proposed method consists of 3 parts:

1. TCP-to-UDT proxy: intercepts TCP connections received from the client and transmits data to UDT-to-UDT or UDT-to-TCP proxy, depending on the number of proxy servers.
2. UDT-to-UDT proxy: receives UDT packets and passes them to the next proxy in the chain, can be used to reduce RTT between proxy servers
3. UDT-to-TCP proxy: receives UDT packets, transforms it to TCP and transmits data to the server.

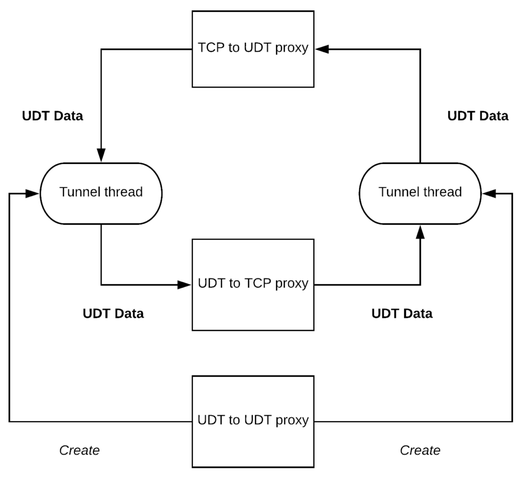
To prevent the creation of additional connections between proxies, the following approach is used.

At the start, each proxy server establishes a specified number of connections to neighboring proxy servers, these are the "tunnels" through which data will be sent.

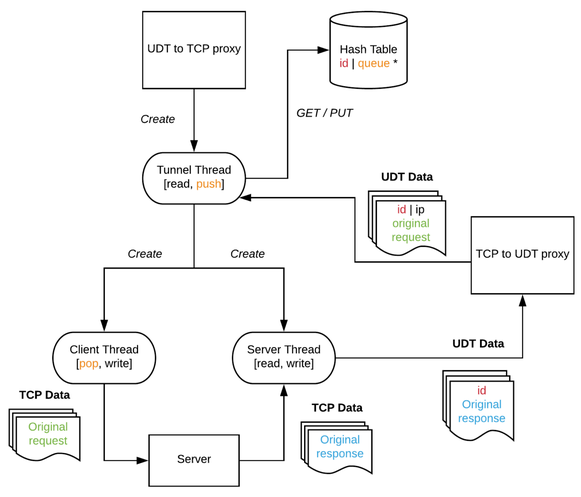
When the **TCP-to-UDT** proxy receives a new client connection, it assigns the unique ID to it and stores the original IP source address and port. After that, proxy application creates the queue to store data received from server. Then two threads are created: the first one receives client TCP packets, adds headers with the connection ID and original source IP and port and passes the UDT data to the chosen tunnel, the second one receives UDT data from tunnel (through the queue created earlier), drops the additional headers, identifying the connections by ID, and transmits TCP packet to client.



UDT-to-UDT proxy is the basic implementation of Split TCP approach, that uses UDT sockets.



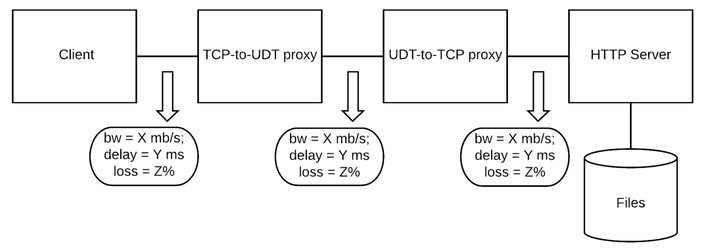
UDT-to-TCP proxy is built similar to TCP-to-UDT one, but with backward logic.



Experimental study

Most experiments were made for topology shown below.

The testbed was built with the help of Linux network virtualization tools, like network namespaces, veth, TC, netem, TBF.



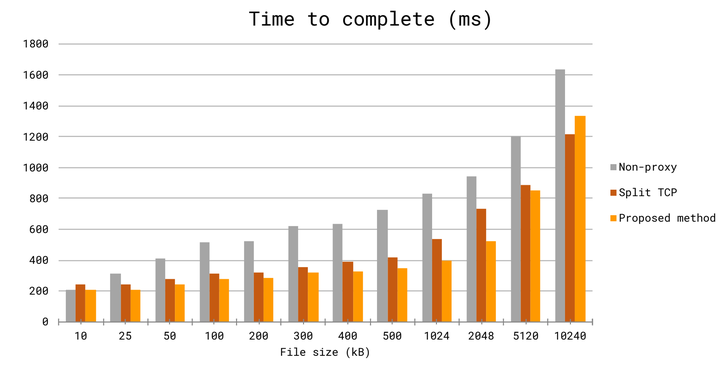
Different cases were considered, setting different values of throughput X, delay Y and percentage of packet loss on each line Z.

To make the conclusion there were considered 3 main cases, with different file sizes for each case:

1. $RTT = 100ms, p_{loss} = 0\%, v = 100MB/s, N_{clients} = 1$
2. $RTT = 100ms, p_{loss} = 5\%, v = 100MB/s, N_{clients} \geq 1$
3. $RTT = 100ms, p_{loss} = 0\%, v = 100MB/s, N_{clients} > 1$

All experiments showed some advantage of proposed method in comparison with non-proxying and Split TCP approaches.

The result of the first experiment shown below.



Conclusion

Experiments showed that the proposed method works better than non-proxy and split TCP approaches for transfer size up to 5 MB, but degrades with increasing numbers of client threads per tunnel. The profit is seen up to 4 threads per tunnel in comparison with Split TCP approach.

Besides, this method allows to reduce channel utilization, by sending fewer acknowledgments, than TCP. Assuming proxy servers located on boundaries of different networks segments, user can achieve better performance by managing (or use its own) congestion control algorithms, without modifying the kernel space.

Bibliography

[1] M.-H. Wang, L.-W. Chen, P.-W. Chi, and C.-L. Lei, "Sdudp: A reliable udp-based transmission protocol over sdn," IEEE Access, vol. 5, pp. 5904–5916, 2017.

[2] F. Le, S. H. Wong, R. Raghavendra, V. Pappas, and E. Nahum, "Removing tcp congestion control on the last hop in split tcp environments," in Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2016 14th International Symposium on. IEEE, 2016, pp. 1–8.

[3] F. Baccelli, G. Carofiglio, and S. Foss, "Proxy caching in split tcp: Dynamics, stability and tail asymptotics," in INFOCOM 2008. The 27th Conference on Computer Communications. IEEE. IEEE, 2008, pp. 131–135.

[4] G. Siracusano, R. Bifulco, S. Kuenzer, S. Salsano, N. B. Melazzi, and F. Huici, "On the fly tcp acceleration with miniproxy," in
Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization. ACM, 2016, pp. 44–49.

[5] Y. Gu and R. L. Grossman, "Udt: Udp-based data transfer for high-speed wide area networks," Computer Networks, vol. 51, no. 7, pp. 1777–1799, 2007.

[6] ——, "Supporting configurable congestion control in data transport services," in Proceedings of the 2005 ACM/IEEE Conference on Supercomputing. IEEE Computer Society, 2005, p. 31.