Reducing the Latency-Tail of Short-Lived Flows: Adding Forward Error Correction in Data Centers

Klaus-Tycho Foerster, Demian Jaeger, David Stolz, and Roger Wattenhofer
ETH Zurich, Switzerland
{foklaus, jaegerde, stolzda, wattenhofer}@ethz.ch

***Abstract*—TCP handles packet loss in the network by retransmitting lost packets, which in turn increases latency. Many connections in data centers are short-lived and consist only of a few packets (e.g., RPCs). Such connections suffer disproportionately from packet retransmissions. We address this issue by introducing a new transport layer protocol called ATP: ATP uses ample forward error correction at the beginning of a connection, allowing short-lived flows to recover from packet loss without retransmissions – but at the same time not congesting long-lived flows. Our experiments show that in an environment with background traffic, the latency's 99th percentile can be reduced by a factor of almost 20 while being fair to other TCP connections.**

***Index Terms*—FEC, Latency Tail, TCP, Data Centers**

I. Introduction

When Microsoft Research examined the traffic of 1500 servers in a cluster for data mining [1], they found that while more than 50 % of all the flows did not last for more than 100 ms, they did not contribute to more than 1 % of the transferred data. Many flows were transmitting at a small rate, 50 % at 10 kB/s or less (cf. also [2], [3]). Even though such little flows do not contribute significantly to the overall traffic volume, they are important and especially affected by a lost packet. The added latency of an additional RTT due to the retransmission is proportionately larger for a small flow than a large one. The authors of [1] state: "*We do believe that TCP's inability to recover from even a few packet drops without resorting to timeouts in low bandwidth delay product settings is a fundamental problem that needs to be solved.*" [1]

Similarly, a review of Facebook's data center network architecture showed that a single HTTP request can result in dozens of cache and database lookups and almost 400 remote procedure calls (RPCs) [4]. We assume that the vast amount of small flows discovered is at least partially caused by RPCs.

As thus, we aim to reduce the latency of *small* flows within data centers by introducing a new transport layer protocol called *Another Transport Protocol (ATP)*. Our goal is to reduce the latency of small flows that is caused by packet loss. While TCP *always* needs to perform a retransmission in the case of a packet loss and the data transfer thus takes an additional Round-trip Time (RTT), our protocol uses Forward Error Correction (FEC) in order to avoid this latency.

Organisation of our short paper. We start by describing the design of ATP in Section II, before providing a positive performance evaluation of ATP in Section III. We then discuss related work in Section IV, and lastly conclude in Section V.

II. Design of ATP: Another Transport Protocol

In this section, we will describe the two most important features of ATP, packet loss handling and congestion control.

A. ATP's Packet Loss Handling

ATP has three different measures to handle lost packets:

Sender Timeouts. A timeout on the sender side occurs if a byte of the stream does not get acknowledged within a specified time. The timeout duration depends on the RTT. If a byte times out, it will be retransmitted by the sender. The protocol tries to avoid these timeouts when possible, as such timeouts are comparatively large and so is the added latency.

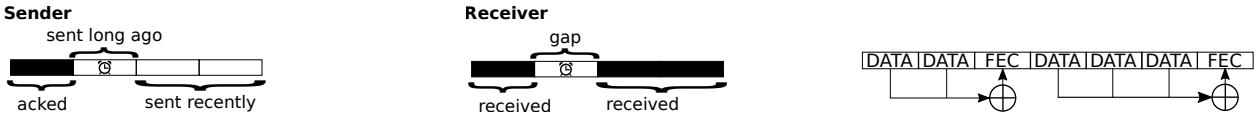
Receiver Timeouts. If the receiver receives out-of-order packets, the received data stream will have gaps. If these gaps exist for too long, they will trigger a timeout and the receiver sends a retransmission request to the sender which then retransmits the missing data. Subfigures 1a and 1b illustrate the two different types of timeouts.

Forward Error Correction. In order to avoid timeouts even in the case of packet loss, the protocol uses a simple systematic block coding. Systematic codes contain the input data in the output. This leads to no decoding overhead if all data is received correctly. After sending multiple packets with plain data, the protocol will insert a FEC packet which contains the XOR of the previously sent data packets as illustrated in Subigure 1c. If one of these packets is lost, the receiver can restore the missing packet. These additional packets result in an overhead leading to higher bandwidth requirement for the same goodput. To minimize the overhead, the rate of the FEC is adjusted by the protocol. At the start of a connection the rate will be high, but it is decreased as the transfer speeds up and there is no loss.

B. Congestion Control

An important feature of a reliable transport layer protocol is its congestion control algorithm. If there is congestion in the network, the protocol must adjust its send rate. The basic principle is illustrated in Figure 2 and is the same for ATP and TCP. The window size is the amount of unacknowledged bytes the sender is allowed to transmit. Adjustment of this window allows control of the transmission speed.

ATP's Congestion Control. Our protocol increases its window based on a RTT without any congestion events. If during

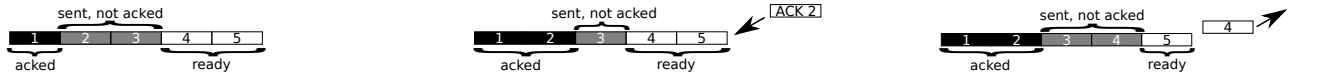


(a) If a packet does not get acknowledged within a certain time period, it will trigger a timeout and the sender will send it again.

(b) If a gap in the receiver stream persists for a specified timeout period, the receiver will ask the sender to resend the packet.

(c) The FEC packets contain the bitwise XOR of the plain data packets sent before.

Fig. 1. Timeouts can either happen at the sender or the receiver.



(a) Packet 1 is already acknowledged, while 2 and 3 are not. Even though the sender has additional data in its buffer, it is not allowed to send it.

(b) The sender now receives an acknowledgment for segment 2. It now has only one unacknowledged packet, while its window size is 2.

(c) The sender now sends packet 4, as there is still space in the window. After sending the fourth packet, all allowed packets have been sent.

Fig. 2. In this example the window size is the length of two packets. The sender is only allowed to send two packets and must wait until an acknowledgment of a packet is received before the next packet can be sent.

a whole RTT no such event occurs and at least some data gets acknowledged, the window will be increased by a constant value corresponding to the maximum packet size. Since this increase might not be fast enough for links with high latency, at the start of a new connection the window is increased with each newly acknowledged segment. However, in data centers the RTT is usually very small, and an increase of the RTT is most probably due to increased buffer latency. Since Another Transport Protocol (ATP) then increases its window size slower, the buffer will not fill up too fast. In the case of a lost packet, there are, as mentioned in Section II-A, three possible ways to get the data to the receiver: Either by using the FEC information, by sending a retransmission request to the sender, or if the sender experiences a timeout. If the receiver needs to use a FEC packet due to a lost packet, it will inform the sender about this event – the sender then decreases its window by a small factor. If the receiver sends a retransmission request or the sender has a local timeout, it will decrease its window by a larger factor.

As the window is increased, the protocol will simultaneously reduce the FEC rate – since the network is not in a congested state with losses. Conversely, if the window size is decreased, the FEC rate is increased.

We note that the specific implementation details of ATP (in *C*) are omitted in this short paper due to space constraints. The main difference of ATP to TCP is the included forward error correction, which is our central focus in this short paper.

III. Performance Evaluation

In this section, we first describe our testbed setup, before comparing the performance of ATP and TCP for small flows under various background traffic settings in Subsection III-A. We analyze a large number of connections, showing that ATP indeed reduces the tail latency of small flows. Lastly, we also briefly show the fairness of ATP to TCP in Subsection III-B.

Testbed Setup. We evaluated the implementation of ATP on a small testbed using up to four laptops and a single desktop machine, depicted in Figure 3. All machines ran on Ubuntu 15.10 and had a Gigabit Ethernet interface. In each experiment, the Ethernet pause frame functionality was disabled. When comparing TCP with ATP, all the hardware helpers of the NIC to support TCP were disabled.

ATP relies on a mechanism called *Random Early Detection (RED)*, which is a widely available in high performance switches used in data centers. RED detects if a queue in a switch gets larger and starts to drop packets before it is completely full. This results in a fairer packet drop distribution across multiple flows, and does usually not result in a drop of consecutive packets – an essential property for ATP. The testbed's switches (*Planet GSD-805*) do not support RED, hence the desktop machine is used to provide RED.

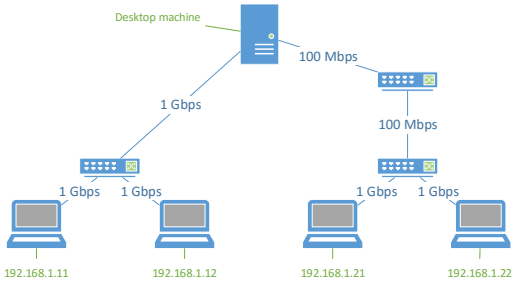
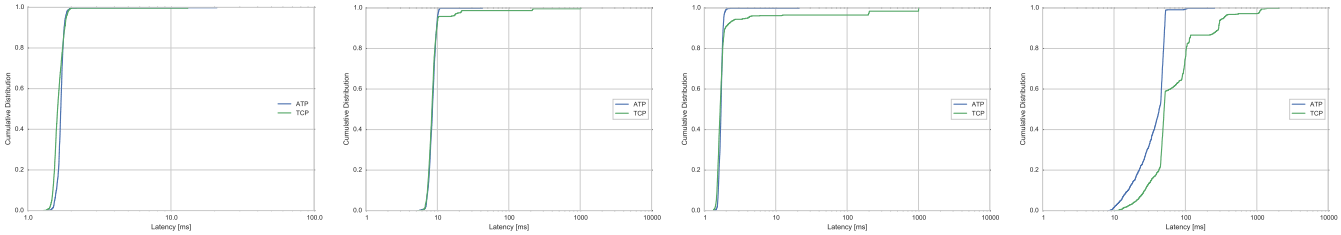


Fig. 3. Depiction of the testbed we used for our experiments. One 100 Mbps link is added between all tested connection for congestion control effects.

A. Comparison of ATP and TCP

The main goal of our design of ATP is to reduce the latency of small flows, in the presence of larger flows. In this subsection, we compare several small ATP connections to TCP on different background traffic scenarios. We will see that, especially in a slightly congested network, ATP often can omit retransmissions and thus additional latency.

For each background traffic scenario, 1000 connections were subsequently completed, each transferring 8.5 kB data from 192.168.1.11 to 192.168.1.21. Background traffic was sent permanently from 192.168.1.12 to 192.168.1.22. For each connection the time for its completion was measured. Note that for TCP connections, the time for the three-way handshake



(a) No other traffic in the network.

(b) Background traffic of 192.168.1.12 permanently sending a large stream of TCP data to 192.168.1.22.

(c) Link loss of 1 % on the Linux machine acting as a switch. The loss was introduced only in the direction in which the data was sent, and not in the acknowledgment direction.

(d) Background traffic of 192.168.1.12 permanently sending a large stream of UDP data to 192.168.1.22; Of all the UDP packets sent, 1.5 % were dropped in the network.

Fig. 4. Comparison of the cumulative distribution functions of the connection completion times of ATP and TCP in various settings of background traffic. ATP is depicted in blue, TCP in green. In these experiments, the sender sent a data stream of 8.5 kB 1000 times from 192.168.1.11 to 192.168.1.21 in the network depicted in Figure 3. While TCP slightly outperforms ATP in cases of no background traffic, the latency tail of the completion times grows considerably with heavy background traffic and data loss, letting ATP complete considerably faster than TCP. E.g., already in Subfigure 4c, ATP finishes a bit after 10ms, while TCP takes about 1000ms to complete for the last flows.

was subtracted from the measured time, in order to have a direct comparison to ATP, which does not perform such a handshake.

No Background Traffic. In the first experiment, there is no background traffic at all. The results shown in Subfigure 4a demonstrate that TCP and ATP behave very alike. The median and the mean of the flow duration is slightly lower for TCP than for ATP, which is expected since ATP transmits 2 additional `FEC` packets for each connection.

TCP Background Traffic. In the second experiment, there is background traffic originating from a large TCP stream. The results are displayed in Subfigure 4b. Since TCP implements a congestion control algorithm, the network will not be excessively congested. Nevertheless, due to the small buffer size set in the desktop machine (acting as a switch), some connections experience packet loss. ATP can reconstruct the stream on the receiver side and – apart from one single connection in the whole sample – does not need to retransmit data at all. This leads to a significantly lower 99th percentile of the flow duration. TCP's 99th percentile is at 210 ms while ATP's is at 11 ms. However, TCP's 95th percentile is only 2 % higher.

Lossy Link In Subfigure 4c the desktop machine introduces a loss rate of 1 %. In such a case ATP will usually be able to reconstruct the lost packets immediately. The 99th percentile is at 2 ms, which is the same as in the case of no loss at all. However, the 99th percentile of TCP is at over *one second*! The reason for this is mainly due to the problem TCP has when the last data packet is lost. The sender waits until a timeout occurs. ATP's last packet sent is a `FEC` packet, and the second last is the stream's last data packet. As long as both of these last packets are not lost in the network, the worst that can happen is a receiver timeout, which is significantly faster than a sender timeout.

UDP Background Traffic The massive injection of UDP packets into the network with a speed slightly above the maximum link capacity congests the network heavily. It results in a network that experiences 1.5 % packet loss on average. Additionally, all buffers are full, which makes retransmissions more expensive. In contrast to experiments with TCP background traffic, where the 95th percentile did not differ extremely between TCP and ATP, the 95th percentile in this experiment differs vastly: While with ATP 95 % of all flows finish within 50 ms, TCP needs 340 ms. The 99th percentile is at 53 ms for ATP and at 1134 ms for TCP. These results are shown in Subfigure 4d.

Tail Latency Subfigures 4a to 4d show the cumulative distribution of the latencies in different environments. Since the tails of the distributions are not clearly visible, Figure 5 shows the last percentiles. We see again, that in case there is congestion, or if packet loss occurs in the network, ATP can reduce the tail latency successfully.

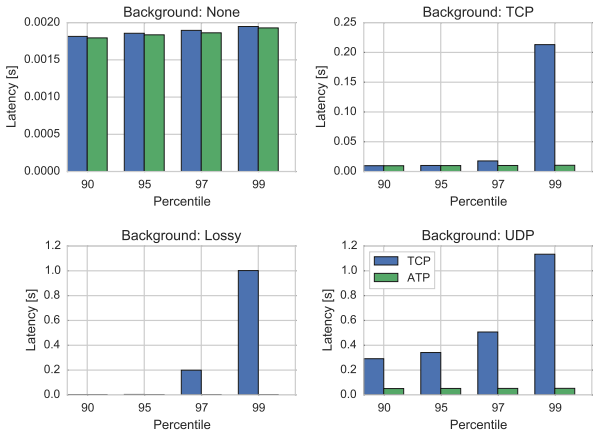


Fig. 5. Tail latencies of ATP and TCP compared in different background traffic scenarios. The different scenarios are described in Subsection III-A.

B. Fairness of Concurrent Connections

An important feature of a transport layer protocol is that it does not starve other connections. If two connections are started in parallel, both connections should use approximately the same bandwidth. By its similarity in design to TCP, ATP is fair to both ATP and TCP – which we also evaluated experimentally in our network testbed.

Due to to space constraints in this short paper, we just briefly review one experiment for TCP: In Figure 6, a TCP and an

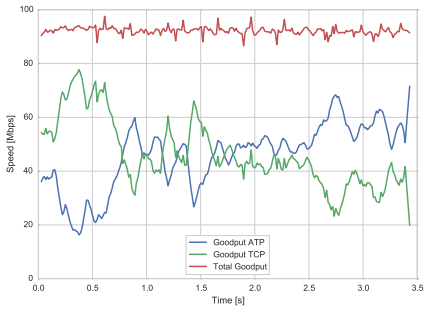


Fig. 6. An ATP and a TCP connection transfer 20 MB of data each. ATP from 192.168.1.11 to 192.168.1.21, TCP from 192.168.1.12 to 192.168.1.22.

ATP stream are started at the same time and both must transmit 20 MB of data. Looking at the whole connection duration the link is shared in a fair way: the TCP stream finishes after 3.43s, while it takes ATP 1.18 % longer to complete.

IV. Related Work

Adding forward error correction (FEC) at the link layer is a well studied concept in wireless networks, see, e.g., [5]. More closely related to our work is applying FEC on a packet level basis. The authors of [6], following an idea of [7], evaluated the effect of applying FEC on the IP and TCP interface by running simulations. If their new *intermediate layer* detects the loss of the next segment of the TCP stream, it waits some time for the arrival of an error correction packet and will rebuild the missing IP packet. Used between two layers, the transport layer protocol can not directly benefit from the additional information provided by the applied FEC[1]. Additionally, their solution adds latency, since the new intermediate layer waits for potential correction packets. This and other [8] early proposals focused on lossy links, rather than on data centers.

Recently, multiple projects have attempted to reduce the latency of flows within a data center. The HULL architecture [9] trades 10 % of the links' bandwidth for reduced latency. If the traffic load on a link arrives at 90 % of the total capacity, a NetFPGA sets the Explicit Congestion Notification (ECN) flag and TCP will slow down. This avoids the filling of the switch's queues and will thus reduce latency. Fastpass [10] lets each sender delegate control to a centralized arbiter which decides when a packet has to be sent and which paths it should take. This results in a single point of failure, or at least a non trivial handover from the primary to the secondary controller. Both of these suggestions need adaption of the intermediate network infrastructure itself, while our approach has the advantage of only requiring software adjustments at the endpoints.

Lastly, for a recent discussion of the impact of the latency tail, also beyond a data center setting, we refer to [11].

V. Conclusion and Outlook

We designed and implemented a transport layer protocol called ATP, which provides the same functionalities as TCP: A reliable transfer of a data stream within an IP network without congesting the network, and with fairness to other ATP, as well as TCP streams. The design goal was to reduce the latency tail of small flows within data centers.

The protocol uses a simple systematic block coding. After sending a variable number of data packets, a FEC packet is sent, containing the XOR of the previously sent data packets. This allows the immediate reconstruction of the stream, if the network dropped a packet and thus reduces the need for retransmissions. ATP uses a sliding window mechanism to control its send rate. Whenever a packet loss occurs, the send rate is decreased and the same time the FEC rate is increased.

In a small testbed, we evaluated the performance of ATP and verified that it behaves fairly to other connections. In a series of multiple small data transfers, ATP often outperforms TCP, since it is able to avoid retransmissions. In an environment which has TCP background traffic, ATP's 99th percentile of the flow completion time is at 11 ms, while TCP's is at 210 ms. The additional FEC information induces a small overhead on the network's load, but the latency decrease can improve the user experience. ATP needs no central controller, or changes to the intermediate hardware, and only relies on software adjustments on the end-hosts.

A next step of using packet based FEC information to shrink the latency of small flows would be to include ATP's functionality as a TCP extension. This would guarantee backward compatibility with hosts that do not support ATP. It is furthermore possible to augment ATP's implementation by a priority-aware congestion control algorithm.

[1]This can be addressed by using Explicit Congestion Notification (ECN) messages, as suggested by the authors.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. Klaus-Tycho Foerster was partially supported by Microsoft Research.

References

[1] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Internet measurement conference*. ACM, 2009.

[2] A. Feldmann, J. Rexford, and R. Cáceres, "Efficient policies for carrying web traffic over flow-switched networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, pp. 673–685, Dec. 1998.

[3] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50–61, 2011.

[4] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *IEEE Optical Interconnects Conf.* IEEE, 2013.

[5] B. Liu, D. Goeckel, and D. Towsley, "Tcp-cognizant adaptive forward error correction in wireless networks," in *GLOBECOM*. IEEE, 2002.

[6] H. Lundqvist and G. Karlsson, "TCP with end-to-end FEC," in *Communications, 2004 International Zurich Seminar on*. IEEE, 2004.

[7] C. Huitema, "The case for packet level fec," in *TC6 WG6.1/6.4 Fifth International Workshop on Protocols for High-Speed Networks V*, 1997.

[8] O. Tickoo, V. Subramanian, S. Kalyanaraman, and K. K. Ramakrishnan, "LT-TCP: end-to-end framework to improve TCP performance over networks with lossy channels," in *IWQoS*, 2005.

[9] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *NSDI*, 2012.

[10] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.

[11] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.