TCP Internals

Sebastian Scheibner
Betreuer: Benjamin Hof, Lukas Schwaighofer
Seminar Future Internet SS2013
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: sebastian.scheibner@in.tum.de

ABSTRACT
TCP is the most used transport protocol today. TCP provides a congestion control mechanism that adapts the transfer speed to the available bandwidth. This works remarkably well for long running transfers, however it is not so good for smaller transfers. Short transfers never reach the maximum possible speed. Especially web pages are affected by this. Web pages consist of many small files for html, css, javascript and images.

Google, a company highly dependent on fast web pages, is proposing ways to increase TCP performance. Among those are TCP Fast Open and an increased initial congestion window. Both techniques try to speed up the beginning of a TCP connection, so primarily short transfers profit from these changes. The page load time for an average website decreases by between 10% and 20% with each of the techniques, if both are combined, the page load time decreases by up to 35%.

Both techniques are currently on their way of becoming an Internet Standard and are already implemented in the Linux Kernel.

Keywords
TCP, TCP Fast Open, three-way handshake, initial congestion window, Slow start

1. INTRODUCTION
Over the past few years the bandwidth of consumer internet connections got bigger and bigger, leading to faster internet access. We are now however at a point where more bandwidth doesn't lead to much faster internet experience [4].

The best way to get faster internet access is to decrease the round-trip time (RTT). This is not always easy. A major cause of latency are large buffers in middle boxes and in the end the RTT is limited by the speed of light. The round-trip time between Munich and New York, for example, can never be faster than 65ms simply because no information can travel faster than the speed of light[1]. What can be done is to decrease the number of round-trips needed and so decreasing the impact of a large RTT. This is especially important for short connections.

[1]speed of light in fibre: $66\% \cdot 300 \cdot 10^6 \frac{m}{s} = 200 \cdot 10^6 \frac{m}{s}$, distance between Munich and New York: 6500 km, one-way delay: $6500km/200 \cdot 10^6 \frac{m}{s} = 32.5ms$. The round-trip time to New York and back is 65ms.

The two techniques proposed by Google, TCP Fast Open and increasing the congestion window do exactly that. Both speed up the beginning of a TCP connection by reducing the number of round-trips.

Normally it takes TCP a full round-trip before any payload data is sent. TCP Fast Open enables TCP to send the first payload already in the first connection establishment packet that is sent to the server. That means the complete transfer needs exactly one round-trip less than normally. For small transfers this has a noticeable impact. However sending data in the first packet that is immediately processed by the server actually circumvents the three-way handshake of TCP, which has some security implications. We will look at this later on and also see how they can be dealt with.

TCP can also be improved in another area. At the beginning of a transfer, TCP only sends small amounts of data, and then waits for an acknowledgment. The exact amount of data being is sent is determined by the initial congestion window. The congestion window is increased exponentially until the maximum transfer speed is reached, but for short transfers it never reaches the maximum. That means for short transfers TCP doesn't use the complete available bandwidth as it spends a lot of time waiting for acknowledgments. So the limiting factor here is not bandwidth, but the RTT, that determines how long it takes for an acknowledgment to arrive. The solution to speed up the transfer is again to reduce the number of round-trips. This can be done by increasing the initial congestion window. Sending more data before waiting for an acknowledgment leads to fewer round-trips in total.

The following section gives a brief introduction to TCP with a special focus on the three-way handshake and congestion window, which are important to understand the following sections. Sections 3 and 4 give a detailed view of TCP Fast Open and increasing initial congestion window respectively. The last section 5 shows how much TCP Fast Open and increasing the initial congestion window affect TCP performance, both alone and combined.

2. TCP BASICS

A TCP stream lifetime can be split into three parts: connection establishment (three-way handshake), slow start and congestion avoidance [12].

1. The *three-way handshake* is used to synchronize TCP sequence numbers and ensures that the client didn't send a request with a spoofed IP address.

2. *Slow start* is used to determine the maximum available bandwidth without congesting the network, by exponentially increasing the number of segments that are sent without waiting for an acknowledgment.

3. When that maximum is reached, i.e. when packet loss occurs, a new phase called *congestion avoidance* begins. In congestion avoidance, the number of packets sent is still increased but only linearly and when packet loss occurs, the number of packets is reset to the maximum determined in slow start.

The three-way handshake consists of three packets. First the client sends a connection request packet (SYN flag set) to the server, that contains a random initial sequence number. No application data is contained in the first packet. The server responds with a SYN-ACK packet, that acknowledges the client's sequence number and contains the server's initial sequence number. Then the client sends an ACK packet that acknowledges the server's sequence number and may contain the first real data sent to the server. If the client had sent the first SYN packet with a spoofed IP address, it wouldn't have received the server's SYN-ACK packet because that was sent to the real owner of the IP address. So the client can't send the third packet because it doesn't know the server's sequence number it must acknowledge, so no connection is established.

After the handshake the roles of the server and client are no longer distinguishable. So from now on we only speak of sender and receiver, where client and server are both sender and receiver. Each sender has his own congestion window.

This is when slow start begins. Now the initial congestion window determines how many packets are sent by the sender before waiting for an acknowledgment from the receiver. With each ACK from the receiver, the congestion window of the sender is increased. In total it is approximately doubled for each RTT [3].

Slow start continues until the congestion window has either reached a preconfigured slow start threshold (ssthresh) or when congestion is observed, i.e. when packet loss occurs. After slow start congestion avoidance takes over. Every RTT the congestion window is increased by at most one segment. When packet loss occurs, i.e. the receiver doesn't acknowledge the last packet, but continuously acknowledges the previous packets, the congestion window is decreased to the maximum determined by slow start. If no acknowledgment is received after a certain timeout, the congestion window is reset to the initial congestion window size and a slow start is performed.

There are also other algorithms for congestion avoidance, but as congestion avoidance isn't important for the following sections we will not discuss them here.

3. TCP FAST OPEN

TCP Fast Open (short TFO), as proposed by Radhakrishnan et al. [13], tries to reduce the latency cost of TCP connection establishment.

3.1 Design

The main idea behind TFO is to send data already in the first request packet, not only after the complete three-way handshake. However the three-way handshake is included in TCP for a good reason and it shouldn't just be removed. The TCP handshake protects the server against denial-of-service attacks with spoofed IP addresses. Therefore TFO introduces a so called TFO cookie. The first time a client opens a connection to a server, a normal three-way handshake is performed. During this handshake the client receives a TFO cookie that can be used for future connections.

Figure 1 shows the packets sent during TFO connection establishment. For the first connection, the following steps are performed:

1. The client includes a Fast Open Cookie Request TCP option in the first SYN packet.
2. If the server supports TFO, it generates a cookie based on the client's IP and sends it to the client in the SYN-ACK packet.
3. The client caches the cookie and can use it for all following connections to this server.
4. The connection continues like a normal TCP connection.

Now the client opens a new connection to the server, using the TFO cookie it just received:

1. The client sends the payload data in the SYN packet and includes the TFO cookie in the TCP options.
2. The server validates the cookie and if it's valid, immediately begins processing the payload. If the cookie is invalid, the server discards the payload and a normal three-way handshake is performed before the client sends the payload again.
3. If the cookie was accepted the server may send data to the client before the first ACK from the client is received
4. From now on the connection behaves like a normal TCP connection.

The TFO cookie is computed by the server in a way that the client can't guess it, e.g. by encrypting the client's source IP using AES with a random key. This has the advantage that no data has to be stored per connection on the server. AES encryption is also very fast on modern hardware, so it

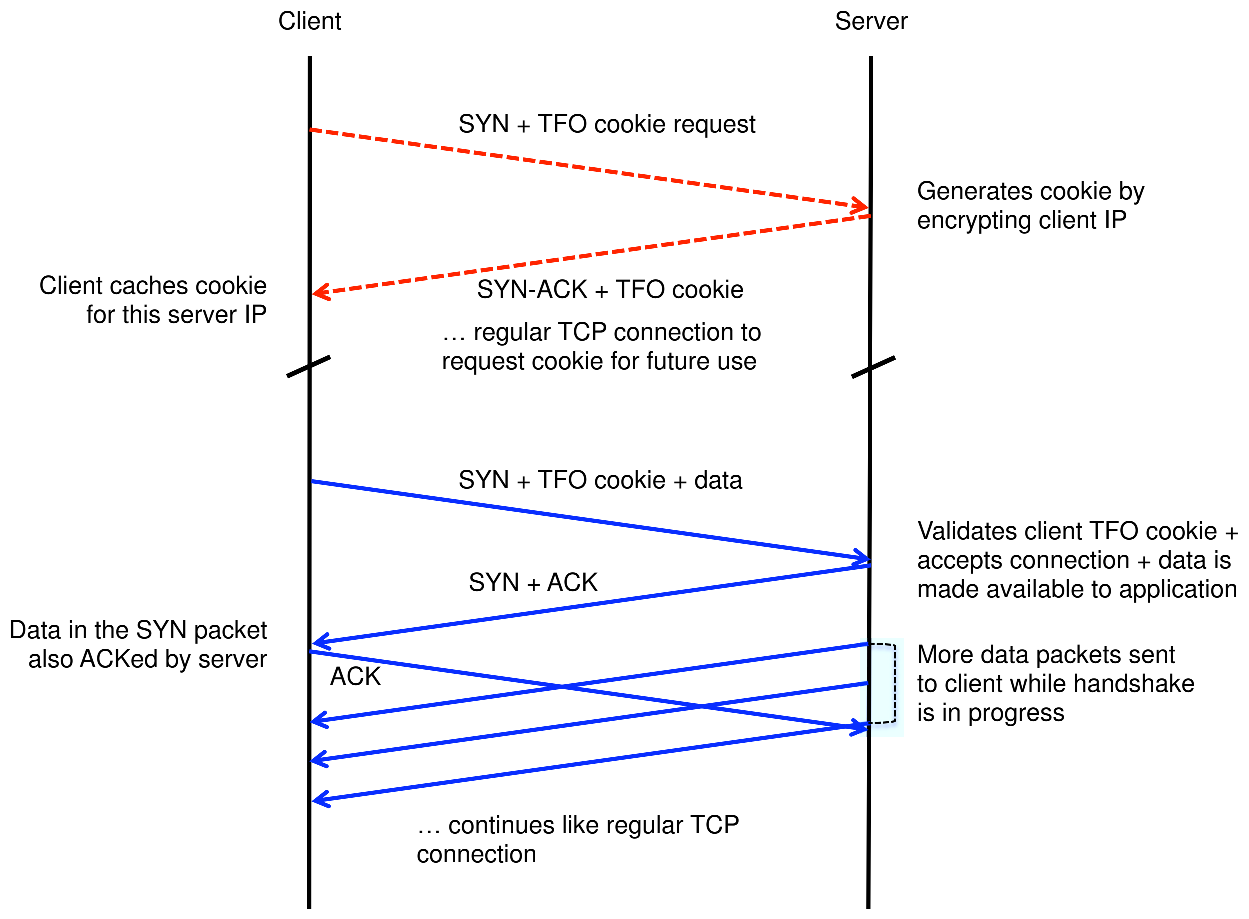
Client
Server
SYN + TFO cookie request
Generates cookie by encrypting client IP
Client caches cookie for this server IP
SYN-ACK + TFO cookie
… regular TCP connection to request cookie for future use
SYN + TFO cookie + data
Validates client TFO cookie + accepts connection + data is made available to application
SYN + ACK
Data in the SYN packet also ACKed by server
ACK
More data packets sent to client while handshake is in progress
… continues like regular TCP connection

Figure 1: TFO connection overview [13]

doesn't expose the server to denial-of-service attacks. The random key is changed periodically, therefore clients must periodically request a new cookie. This prevents clients from harvesting cookies. Instead of changing keys the server could also include timestamps in the cookie generation.

The maximum size of TCP segments is determined using the server's maximum segment size. The server includes its maximum segments size (MSS) in the SYN-ACK packet so the client knows how much data it may send. However for TCP Fast Open, data is already sent in the SYN packet when the client doesn't yet know the server's MSS. Therefore it is recommended that the client caches the MSS, received along with the TFO cookie in the first connection, and uses it for the SYN packet of future connections. Otherwise the client would be restricted to the default MSS of 536 bytes [6].

3.2 Security

TFO also introduces some additional security aspects that need to be examined carefully [13].

3.2.1 Server Resource Exhaustion

Like normal TCP, TFO is vulnerable to SYN floods. However the consequences can be much worse. A traditional SYN flood prevents the server from receiving any more connections by flooding its SYN table. A TFO SYN flood attack is an amplification attack, that causes high load on the server with small load on the client.

If the attacker doesn't have a valid TFO cookie, a handshake must be performed and the server can be protected by using traditional SYN cookies. Once an attacker has a valid TFO cookie he can open many more TCP connections for which the server processes the request before the handshake is complete, meaning the client does not use many resources in order to cause resource exhaustion on the server.

A way to reduce the effectiveness of this attack is to restrict the number of pending TFO connection requests. Pending means the client has sent a SYN packet but no ACK packet yet. If that limit is reached, all following TFO requests will require a normal TCP handshake, which allows normal SYN flood defense techniques to protect the server.

3.2.2 Amplified Reflection Attack

If an attacker acquires a valid TFO cookie from another client, it can send lots of requests with a spoofed IP address to the server. The server sends the response data to the victim client and may congest his network. However in order to acquire a valid SYN cookie, the attacker would have to be in the same network or have remote access to the victim. If the attacker has remote access he won't need to use an amplified reflection attack, he can generate the requests directly on the client.

A way to mitigate that attack on the server is to not send any data before receiving the first ACK from the client. The server still processes the request immediately so it is still faster than without TFO. This is not implemented yet, but could be done if an attack is observed.

3.3 Deployment

This section examines how difficult it is to roll out TFO to the web. For TFO to work, both client and server need to be TFO capable. If one of them isn't, the connection just falls back to a normal TCP handshake, so connection establishment is still possible. This means enabling TFO is backwards compatible, it can be activated on any server or client and normal TCP connections still work.

Since TFO uses a not yet standardized TCP option, servers that don't support TFO respond either with a normal SYN-ACK packet or not at all to a SYN packet with a TFO cookie request. Even if the server supports TFO, the client might still not receive a response, if some middle box discards non standard TCP packets. If the server doesn't respond after a timeout, TFO sends a normal SYN packet without the TFO option. Some NAT gateways use different IPs for new connections, so existing client TFO cookies are no longer valid on the server, then a normal three-way handshake is performed and the data in the SYN packet is rejected.

In order to support TFO, the TCP stack in the operating system needs deep modifications. This includes creating and validating TFO cookies on the server and requesting, as well as caching and sending the TFO cookie on the client side. However, on the application level only small changes are needed to use TFO on a TFO capable operating system. The application states that it wants to use TFO by setting a TCP socket flag. Furthermore instead of the usually used connect() and send() function calls, the application must use the already existing function sendto() that combines the connect() and send() calls. This is necessary so the operating system has the data that will be sent in the first SYN packet in case TFO can be used.

The Linux kernel already supports TFO since version 3.7. However it must be first enabled system-wide before applications can use it[2]. Google is currently working on an Internet Draft to make TFO an Internet Standard [6].

4. INCREASE INITIAL CONGESTION WINDOW

The initial congestion window specifies the number of packets that are sent before waiting for an ACK packet from the receiver. The now obsolete RFC 2581 specifies the initial congestion window to be at most 2 segments, it was replaced by RFC 5681 in 2009 which specifies a maximum of 2 to 4 segments, depending on the sender maximum segment size [3, 2, 1].

[2]`sysctl -w net.ipv4.tcp_fastopen=3`

4.1 Design

The initial congestion window is largely responsible for the throughput at the beginning of a new TCP connection. Because bandwidth has increased considerably over the past years the first segments of the congestion window are sent quickly. Then however the sender must wait for a full round-trip time before sending the next segment. As previously mentioned, the round-trip time (RTT) has only marginally decreased over the past years. This means that during slow start the TCP connection spends a lot of time waiting while only gradually sending more packets. So, in order to speed up short transfers, the RTT must be decreased. Decreasing the RTT is rather hard, as previously mentioned, this is caused by large buffers in middle boxes. Instead we can try to reduce the number of round-trips it takes a connection to finish. This way we also speed up the connection.

One proposal to decrease the number of round-trips is to increase the initial congestion window, sending more data before waiting for an acknowledgment. As mentioned in section 2, the congestion window is usually doubled for each round-trip, so a larger initial congestion window results in fewer round-trips and the transfer finishes sooner [8]. For large transfers we don't see much improvement, the slow start is only a small fraction of the transfer time, most of the life time is spent in congestion avoidance. However, small transfers only rarely reach congestion avoidance and spend most of their life time in slow start. Those transfers finish much faster.

Increasing start up performance also reduces the need for browsers to open multiple parallel connections. This is done to decrease page load times, but circumvents slow start by opening up to 6 connections per server [15].

4.2 Deployment

Implementing an increased initial congestion window is fairly straight-forward. Only a constant has to be changed on the sender side. All receivers that have a large enough receive window of at least 10 segments get the benefit of the increased initial congestion window. This is the case for Linux[3], Windows and OS X.

However you should be aware that this violates the currently valid RFC 5681, which specifies the initial congestion window MUST NOT be more than 4 segments. The IETF is working on an Internet Draft that will allow initial congestion windows with up to 10 segments, but until this becomes a standard, servers with larger initial congestion windows are not standard compliant [7].

5. EVALUATION

Dukkipati et al. tested the performance improvement of an increased initial congestion window, by performing various experiments with a congestion window of 10 segments compared to a congestion window of 3 segments [8]. For high RTT networks they observed approximately 10% latency improvements of HTTP responses. TCP Fast Open was tested by Radhakrishnan et al. [13]. They used an early implementation in the Linux kernel to determine the latency improvement. They observed an improvement of 10% for the

[3]since kernel version 2.6.39

latency of a single HTTP request and 4% to 40% for overall page load time.

In order to reproduce these results I set up a mininet VM [11] that allows simulating arbitrarily complex network setups using a simple command-line interface or a python API. Mininet allows to simulate different network properties, like setting the latency and bandwidth on links. For my experiments I used a simple setup with two hosts connected through a switch. In order to test TCP Fast Open I had to update the Linux kernel to 3.8.3 because the mininet VM comes with a kernel 3.5 that doesn't support TFO.

The first experiment tests the duration of a file transfer using different initial congestion windows. The value for the initial congestion window is increased from 1 to 20. In figure 2 you can see the duration of the transfer of three different files as a function of the initial congestion window (ICWND) in a high latency network (RTT 200ms). For an ICWND of 4, which is the current standard, transferring a 60KiB file takes 1.3 seconds. With a ICWND of 10 it only takes 1.1 seconds, this is a improvement of 15%. In a lower latency network the improvement a bit less. But, as figure 3 shows, I still observed 13% improvement for the 60KiB file with 40ms RTT.

The diagrams also show that the improvement is more noticeable for short transfers. The 1MiB transfer in this experiment profits only very little from the increased initial congestion window.

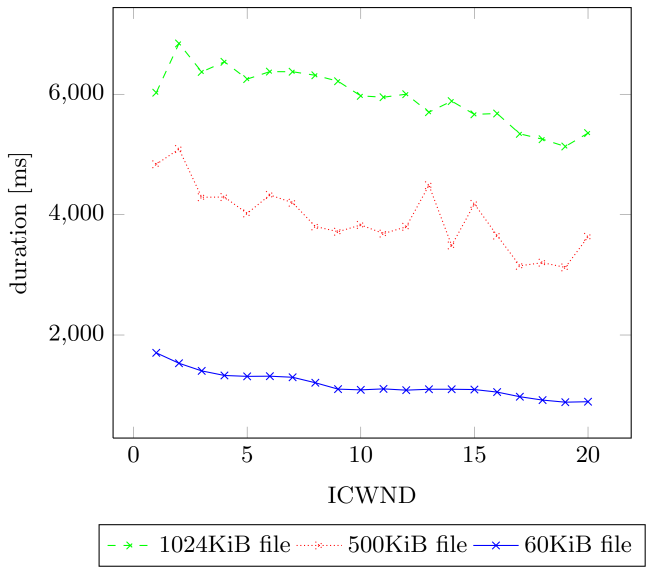

Figure 2: 10Mbit/s RTT: 200ms

The next experiment simulates an average website request. We want to determine how much a increased initial congestion window (ICWND) and TCP Fast Open improve the load time of this test website. A client requests one main file with about 30KiB. Then several other transfers are started simultaneously for files like css, javascript and images with sizes between 10 and 60KiB.

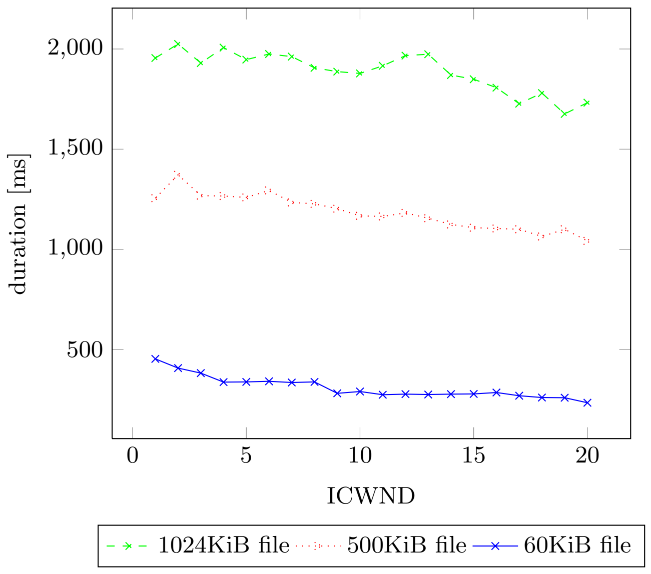

Figure 3: 10Mbit/s RTT: 40ms

This scenario is performed for different RTTs between 10ms and 200ms and with all combinations of the previously discussed techniques, TFO disabled and enabled and with an initial congestion window of 4 and 10.

To simulate that the client connects to the server for the first time, the server's TFO cookie key must be reset for each test[4]. So even with TFO enabled, the first connection uses a three way handshake and only the following connections send their data in the SYN packet.

In figure 4 we can see the absolute duration of the complete download as a function of the RTT. As expected, the transfer takes the longest, without TFO and with a normal initial congestion window. That's the line at the top, the line at the bottom is with TFO and an increased ICWND. Those transfers are the fastest. If TFO and increased ICWND are used individually they both take about the same time somewhere between the two outer lines.

Figure 5 shows the improvement of TFO and an increased ICWND relative to the current standard. The first thing we can see is that when we have a larger RTT the improvement is larger. For RTTs above 50ms the improvement of TFO and increased ICWND individually is between 5% and 20%. Both techniques improve the load time approximately the same. For other websites this doesn't have to be the case, if there are files of several hundred kilo bytes the increased ICWND should have a larger effect than TFO.

The most interesting thing is the column where both techniques are used. The combination of both techniques yields an improvement between 15% and 35%, that's almost the sum of the individual improvements.

[4]`sysctl -w net.ipv4.tcp_fastopen_key=RAN-DOM-VAL-UE`

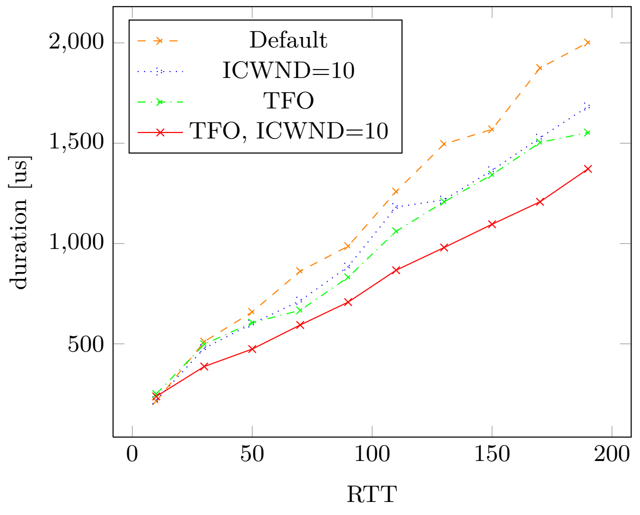

Figure 4: Compare website access time (10Mbit/s) with/without TFO, increased ICWND

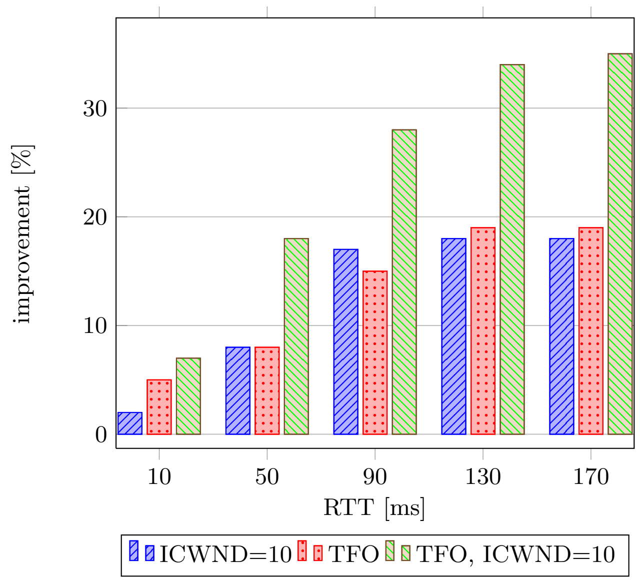

Figure 5: Relative compare website access time (10Mbit/s) with/without TFO, increased ICWND

6. RELATED WORK

Roan Kattouw and Max Meyers [10] reproduced the TFO performance analysis done by Radhakrishnan et al. [13] for loading web pages by replaying page loads for four popular web sites. They also found that TFO improves the download speed of popular websites by at least 10%.

In addition to the TCP enhancements mentioned in this text, Google is also working on a complete TCP replacement called QUIC [14]. QUIC is supposed to be faster than TCP, by using multiple streams in one connection. It also offers encryption, congestion control and forward error correction. This technology is not yet officially announced and only exists as source code in the chromium repository.

SPDY [9] is another new protocol developed by Google, that works one layer above TCP and is supposed to replace HTTP. SPDY offers default encryption, header compression and multiplexing several streams in one connection. Many browsers already support SPDY and also various web servers. The new HTTP/2.0 standard [5], that is currently developed is mainly based on SPDY.

7. CONCLUSION

As shown in section 5, TCP Fast Open and an increased initial congestion window bring large performance benefits for short transfers. Especially the increased initial congestion window is a small change that has a significant impact on transfer performance. The deployment is easy and it comes at little cost in today's networks. A proposal to increase the initial congestion window by default is in the process of becoming an Internet Standard. TCP Fast Open considerably improves TCP start-up time. It requires more changes in the networking layer than increasing the initial congestion window, but that is doable. The TFO cookie provides acceptable protection against possible denial of service attacks on the server. TFO is already implemented in the Linux kernel, showing that TFO brings at least 10% performance improvements for short transfers. TFO is on its way of becoming an Internet Standard, the first step to become widely adopted and enabled by default. Combining both techniques results in an even larger performance improvement. The whole web would become a bit faster if all hosts implement these changes.

8. REFERENCES

[1] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390, October 2002.

[2] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[4] M. Belshe. More Bandwidth Doesn't Matter (Much), May 2010. https://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/, 17.03.13.

[5] M. Belshe, M. Thomson, and A. Melnikov. SPDY protocol draft-ietf-httpbis-http2-00, November 2012. https://tools.ietf.org/html/draft-ietf-httpbis-http2-00.

[6] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. IETF Internet Draft – work in progress 03, Google, Inc., February 2013.

[7] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's Initial Window. IETF Internet Draft – work in progress 08, Google, Inc., February 2013.

[8] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *Computer Communication Review*, 40(3):26–33, 2010.

[9] Google. SPDY. http://www.chromium.org/spdy.

[10] R. Kattouw and M. Meyers. CS244 '13: TCP fast open, March 2013.

https://reproducingnetworkresearch.wordpress.com/2013/03/12/cs244-13-tcp-fast-open/.

[11] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[12] U. of Southern California. Transmission Control Protocol. RFC 793 (Internet Standard), September 1981.

[13] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP fast open. In K. Cho and M. Crovella, editors, *CoNEXT*, page 21. ACM, 2011.

[14] M. Riepe. Google arbeitet an "QUIC", February 2013. http://www.heise.de/ix/meldung/Google-arbeitet-an-QUIC-1809419.html (German).

[15] S. Souders. Roundup on parallel connections, March 2008. http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/.