Internet Draft
draft-ietf-tcpm-fastopen-02.txt
Intended status: Experimental
Expiration date: April, 2013

Y. Cheng
J. Chu
S. Radhakrishnan
A. Jain
Google, Inc.
Octobor 22, 2012

TCP Fast Open

Status of this Memo

Copyright Notice

Abstract

TCP Fast Open (TFO) allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiving end during the initial connection handshake, thus saving up to one full round trip time (RTT) compared to standard TCP which requires a three-way handshake (3WHS) to complete before data can be exchanged.

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. TFO refers to TCP Fast Open. Client refers to the TCP's active open side and server refers to the TCP's passive open side.

1. **Introduction**

TCP Fast Open (TFO) enables data to be exchanged safely during TCP's connection handshake.

This document describes a design that enables applications to save a round trip while avoiding severe security ramifications. At the core of TFO is a security cookie used by the server side to authenticate a client initiating a TFO connection. This document covers the details of exchanging data during TCP's initial handshake, the protocol for TFO cookies, and potential new security vulnerabilities and their mitigation. It also includes discussions of deployment issues and related proposals. TFO requires extensions to the socket API but this document does not cover that.

TFO is motivated by the performance needs of today's Web applications. Network latency is determined by the round-trip time (RTT) and the number of round trips required to transfer application data. RTT consists of propagation delay and queuing delay. Network bandwidth has grown substantially over the past two decades, reducing queuing delay, while propagation delay is largely constrained by the speed of light and has remained unchanged. Therefore reducing the number of round trips has become the most effective way to improve the latency of Web applications [CDCM11].

Standard TCP only permits data exchange after 3WHS [RFC793], which adds one RTT to the network latency. For short transfers (e.g., web objects) this additional RTT is a significant portion of the network latency [THK98]. One widely deployed solution is HTTP persistent connections. However, this solution is limited since hosts and middle boxes terminate idle TCP connections due to resource constraints. For example, the Chrome browser keeps TCP connections idle up to 5

minutes but 35% of Chrome HTTP requests are made on new TCP connections. We discuss HTTP persistent connections further in section 7.1.

2. Data In SYN

[RFC793] (section 3.4) already allows data in SYN packets but forbids the receiver to deliver the data to the application until 3WHS is completed. This is because TCP's initial handshake serves to capture 1) Old or duplicate SYNs and 2)SYNs with spoofed IP addresses.

TFO allows data to be delivered to the application before 3WHS is completed, thus opening itself to a possible data integrity problem caused by the problematic SYN packets above. This could cause a problem in the following two examples: a) the receiver host receives both duplicate and original SYNs before and after the host reboots, and b) the duplicate is received after the connection created by the original SYN has been closed. The receiver will not be protected by the 2MSL TIMEWAIT state if the close is initiated by the sender. In both cases, the data is replayed.

2.1. TCP Semantics and Duplicate SYNs

The proposed T/TCP protocol employs a new TCP "TAO" option and connection count to guard against old or duplicate SYNs [RFC1644]. The solution is complex, involving state tracking on a per remote peer basis, and is vulnerable to IP spoofing attacks. Moreover, it has been shown that despite its complexity, T/TCP is still not entirely protected. Old or duplicate SYNs may still be accepted by a T/TCP server [PHRACK98].

Rather than trying to capture all dubious SYN packets to make TFO 100% compatible with TCP semantics, we made a design decision early on to accept old SYN packets with data, i.e., to restrict TFO to use with a class of applications that are tolerant of duplicate SYN packets with data. We believe this is the right design trade-off balancing complexity with usefulness. Applications that require transactional semantics already deploy specific mechanisms to tolerate similar data replay issues in TCP today. For example, a browser reload event may replay any HTTP request even without data in SYN. For transactional HTTP requests applications typically include unique identifiers in the HTTP headers. Thus, allowing data in SYN poses little risk to existing HTTP applications.

However, we note that some applications may rely on TCP 3-way handshake semantics. For this reason, TFO MUST be used explicitly by applications on a per service port basis.

2.2. SYNs with spoofed IP addresses

Standard TCP suffers from the SYN flood attack [RFC4987] because bogus SYN packets, i.e., SYN packets with spoofed source IP addresses can easily fill up a listener's small queue, causing a service port to be blocked completely until timeouts. Secondary damage comes from these SYN requests taking up memory space. Though this is less of an issue today as servers typically have plenty of memory.

TFO goes one step further to allow server side TCP to process and send up data to the application layer before 3WHS is completed. This opens up more serious new vulnerabilities. Applications serving ports that have TFO enabled may waste lots of CPU and memory resources processing the requests and producing the responses. If the response is much larger than the request, the attacker can mount an amplified reflection attack against victims of choice beyond the TFO server itself.

Numerous mitigation techniques against the regular SYN flood attack exist and have been well documented [RFC4987]. Unfortunately none are applicable to TFO. We propose a server supplied cookie to mitigate most of the security issues introduced by TFO. We defer further discussion of SYN flood attacks to the "Security Considerations" section.

3. Protocol Overview

The key component of TFO is the Fast Open Cookie (cookie), a message authentication code (MAC) tag generated by the server. The client requests a cookie in one regular TCP connection, then uses it for future TCP connections to exchange data during 3WHS:
Requesting a Fast Open Cookie:

1. The client sends a SYN with a Fast Open Cookie Request option.

2. The server generates a cookie and sends it through the Fast Open Cookie option of a SYN-ACK packet.

3. The client caches the cookie for future TCP Fast Open connections (see below).

Performing TCP Fast Open:

1. The client sends a SYN with Fast Open Cookie option and data.

2. The server validates the cookie:
   a. If the cookie is valid, the server sends a SYN-ACK acknowledging both the SYN and the data. The server then delivers the data to the application.

      b. Otherwise, the server drops the data and sends a SYN-ACK
         acknowledging only the SYN sequence number.

   3. If the server accepts the data in the SYN packet, it may send the
      response data before the handshake finishes. The max amount is
      governed by the TCP's congestion control [RFC5681].

   4. The client sends an ACK acknowledging the SYN and the server data.
      If the client's data is not acknowledged, the client retransmits
      the data in the ACK packet.

   5. The rest of the connection proceeds like a normal TCP connection.
   The client can repeat many Fast Open operations once it acquires a
   cookie (until the cookie is expired by the server). Thus TFO is
   useful for applications that have temporal locality on client and
   server connections.

   Requesting Fast Open Cookie in connection 1:

```
      TCP A (Client)                                      TCP B(Server)
      ______________                                      _____________
      CLOSED                                                     LISTEN

   #1 SYN-SENT       ----- <SYN,CookieOpt=NIL>  ---------->  SYN-RCVD

   #2 ESTABLISHED    <---- <SYN,ACK,CookieOpt=C> ----------  SYN-RCVD
       (caches cookie C)
```

   Performing TCP Fast Open in connection 2:

```
      TCP A (Client)                                      TCP B(Server)
      ______________                                      _____________
      CLOSED                                                     LISTEN

   #1 SYN-SENT       ----- <SYN=x,CookieOpt=C,DATA_A> ---->  SYN-RCVD

   #2 ESTABLISHED    <---- <SYN=y,ACK=x+len(DATA_A)+1> ----  SYN-RCVD

   #3 ESTABLISHED    <---- <ACK=x+len(DATA_A)+1,DATA_B>----  SYN-RCVD

   #4 ESTABLISHED    ----- <ACK=y+1>--------------------> ESTABLISHED

   #5 ESTABLISHED    --- <ACK=y+len(DATA_B)+1>----------> ESTABLISHED
```

4. Protocol Details

4.1. Fast Open Cookie

The Fast Open Cookie is designed to mitigate new security vulnerabilities in order to enable data exchange during handshake. The cookie is a message authentication code tag generated by the server and is opaque to the client; the client simply caches the cookie and passes it back on subsequent SYN packets to open new connections. The server can expire the cookie at any time to enhance security.

4.1.1. TCP Options

Fast Open Cookie Option

The server uses this option to grant a cookie to the client in the SYN-ACK packet; the client uses it to pass the cookie back to the server in the SYN packet.

```
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |      Kind     |    Length     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                            Cookie                             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Kind            1 byte: constant TBD (assigned by IANA)
Length          1 byte: range 6 to 18 (bytes); limited by
                        remaining space in the options field.
                        The number MUST be even.
Cookie          4 to 16 bytes (Length - 2)
```

Options with invalid Length values or without SYN flag set MUST be ignored. The minimum Cookie size is 4 bytes. Although the diagram shows a cookie aligned on 32-bit boundaries, alignment is not required.

Fast Open Cookie Request Option

The client uses this option in the SYN packet to request a cookie from a TFO-enabled server

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Kind     |    Length     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Kind	1 byte: same as the Fast Open Cookie option
Length	1 byte: constant 2. This distinguishes the option from the Fast Open cookie option.

Options with invalid Length values, without SYN flag set, or with ACK flag set MUST be ignored.

4.1.2. Server Cookie Handling

The server is in charge of cookie generation and authentication. The cookie SHOULD be a message authentication code tag with the following properties:

1. The cookie authenticates the client's (source) IP address of the SYN packet. The IP address can be an IPv4 or IPv6 address.

2. The cookie can only be generated by the server and can not be fabricated by any other parties including the client.

3. The generation and verification are fast relative to the rest of SYN and SYN-ACK processing.

4. A server may encode other information in the cookie, and accept more than one valid cookie per client at any given time. But this is all server implementation dependent and transparent to the client.

5. The cookie expires after a certain amount of time. The reason for cookie expiration is detailed in the "Security Consideration" section. This can be done by either periodically changing the server key used to generate cookies or including a timestamp when generating the cookie.

   To gradually invalidate cookies over time, the server can implement key rotation to generate and verify cookies using multiple keys. This approach is useful for large-scale servers to retain Fast Open rolling key updates. We do not specify a particular mechanism because the implementation is often server specific.

The server supports the cookie generation and verification operations:

- GetCookie(IP_Address): returns a (new) cookie

- IsCookieValid(IP_Address, Cookie): checks if the cookie is valid, i.e., it has not expired and it authenticates the client IP address.

Example Implementation: a simple implementation is to use AES_128 to encrypt the IPv4 (with padding) or IPv6 address and truncate to 64 bits. The server can periodically update the key to expire the cookies. AES encryption on recent processors is fast and takes only a few hundred nanoseconds [RCCJB11].

If only one valid cookie is allowed per-client and the server can regenerate the cookie independently, the best validation process is to simply regenerate a valid cookie and compare it against the incoming cookie. In that case if the incoming cookie fails the check, a valid cookie is readily available to be sent to the client.

The server MAY return a cookie request option, e.g., a null cookie, to signal the support of Fast Open without generating cookies, for probing or debugging purposes.

4.1.3. Client Cookie Handling

The client MUST cache cookies from servers for later Fast Open connections. For a multi-homed client, the cookies are both client and server IP dependent. Beside the cookie, we RECOMMEND that the client caches the MSS and RTT to the server to enhance performance.

The MSS advertised by the server is stored in the cache to determine the maximum amount of data that can be supported in the SYN packet. This information is needed because data is sent before the server announces its MSS in the SYN-ACK packet. Without this information, the data size in the SYN packet is limited to the default MSS of 536 bytes [RFC1122]. The client SHOULD update the cache MSS value whenever it discovers new MSS value, e.g., through path MTU discovery.

Caching RTT allows seeding a more accurate SYN timeout than the default value [RFC6298]. This lowers the performance penalty if the network or the server drops the SYN packets with data or the cookie options (See "Reliability and Deployment Issues" section below).

The cache replacement algorithm is not specified and is left for the implementations.

Note that before TFO sees wide deployment, clients are advised to also cache negative responses from servers in order to reduce the amount of futile TFO attempts. Since TFO is enabled on a per-service port basis but cookies are independent of service ports, clients'

cache should include remote port numbers too.

4.2. Fast Open Protocol

One predominant requirement of TFO is to be fully compatible with existing TCP implementations, both on the client and the server sides.

The server keeps two variables per listening port:

FastOpenEnabled: default is off. It MUST be turned on explicitly by the application. When this flag is off, the server does not perform any TFO related operations and MUST ignore all cookie options.

PendingFastOpenRequests: tracks number of TFO connections in SYN-RCVD state. If this variable goes over a preset system limit, the server SHOULD disable TFO for all new connection requests until PendingFastOpenRequests drops below the system limit. This variable is used for defending some vulnerabilities discussed in the "Security Considerations" section.

The server keeps a FastOpened flag per TCB to mark if a connection has successfully performed a TFO.

4.2.1. Fast Open Cookie Request

Any client attempting TFO MUST first request a cookie from the server with the following steps:

1. The client sends a SYN packet with a Fast Open Cookie Request option.

2. The server SHOULD respond with a SYN-ACK based on the procedures in the "Server Cookie Handling" section. This SYN-ACK SHOULD contain a Fast Open Cookie option if the server currently supports TFO for this listener port.

3. If the SYN-ACK contains a Fast Open Cookie option, the client replaces the cookie and other information as described in the "Client Cookie Handling" section. Otherwise, if the SYN-ACK is first seen, i.e.,not a (spurious) retransmission, the client MAY remove the server information from the cookie cache. If the SYN-ACK is a spurious retransmission without valid Fast Open Cookie Option, the client does nothing to the cookie cache for the reasons below.

The network or servers may drop the SYN or SYN-ACK packets with the new cookie options which causes SYN or SYN-ACK timeouts. We RECOMMEND

both the client and the server retransmit SYN and SYN-ACK without the cookie options on timeouts. This ensures the connections of cookie requests will go through and lowers the latency penalties (of dropped SYN/SYN-ACK packets). The obvious downside for maximum compatibility is that any regular SYN drop will fail the cookie (although one can argue the delay in the data transmission till after 3WHS is justified if the SYN drop is due to network congestion). Next section describes a heuristic to detect such drops when the client receives the SYN-ACK.

We also RECOMMEND the client to record servers that failed to respond to cookie requests and only attempt another cookie request after certain period. An alternate proposal is to request cookie in FIN instead since FIN-drop by incompatible middle-box does not affect latency. However such paths are likely to drop SYN packet with data later, and many applications close the connections with RST instead, so the actual benefit of this approach is not clear.

4.2.2. TCP Fast Open

Once the client obtains the cookie from the target server, the client can perform subsequent TFO connections until the cookie is expired by the server. The nature of TCP sequencing makes the TFO specific changes relatively small in addition to [RFC793].

Client: Sending SYN

To open a TFO connection, the client MUST have obtained the cookie from the server:

1. Send a SYN packet.

   a. If the SYN packet does not have enough option space for the Fast Open Cookie option, abort TFO and fall back to regular 3WHS.

   b. Otherwise, include the Fast Open Cookie option with the cookie of the server. Include any data up to the cached server MSS or default 536 bytes.

2. Advance to SYN-SENT state and update SND.NXT to include the data accordingly.

3. If RTT is available from the cache, seed SYN timer according to [RFC6298].

To deal with network or servers dropping SYN packets with payload or unknown options, when the SYN timer fires, the client SHOULD retransmit a SYN packet without data and Fast Open Cookie options.

Server: Receiving SYN and responding with SYN-ACK

Upon receiving the SYN packet with Fast Open Cookie option:

1. Initialize and reset a local FastOpened flag. If FastOpenEnabled is false, go to step 5.

2. If PendingFastOpenRequests is over the system limit, go to step 5.

3. If IsCookieValid() in section 4.1.2 returns false, go to step 5.

4. Buffer the data and notify the application. Set FastOpened flag and increment PendingFastOpenRequests.

5. Send the SYN-ACK packet. The packet MAY include a Fast Open Option. If FastOpened flag is set, the packet acknowledges the SYN and data sequence. Otherwise it acknowledges only the SYN sequence. The server MAY include data in the SYN-ACK packet if the response data is readily available. Some application may favor delaying the SYN-ACK, allowing the application to process the request in order to produce a response, but this is left to the implementation.

6. Advance to the SYN-RCVD state. If the FastOpened flag is set, the server MUST follow the congestion control [RFC5681], in particular the initial congestion window [RFC3390], to send more data packets.

If the SYN-ACK timer fires, the server SHOULD retransmit a SYN-ACK segment with neither data nor Fast Open Cookie options for compatibility reasons.

Client: Receiving SYN-ACK

The client SHOULD perform the following steps upon receiving the SYN-ACK:

1. Update the cookie cache if the SYN-ACK has a Fast Open Cookie Option or MSS option or both.

2. Send an ACK packet. Set acknowledgment number to RCV.NXT and include the data after SND.UNA if data is available.

3. Advance to the ESTABLISHED state.

Note there is no latency penalty if the server does not acknowledge the data in the original SYN packet. The client SHOULD retransmit any unacknowledged data in the first ACK packet in step 2. The data exchange will start after the handshake like a regular TCP

connection.

If the client has timed out and retransmitted only regular SYN packets, it can heuristically detect paths that intentionally drop SYN with Fast Open option or data. If the SYN-ACK acknowledges only the initial sequence and does not carry a Fast Open cookie option, presumably it is triggered by a retransmitted (regular) SYN and the original SYN or the corresponding SYN-ACK was lost.

Server: Receiving ACK

Upon receiving an ACK acknowledging the SYN sequence, the server decrements PendingFastOpenRequests and advances to the ESTABLISHED state. No special handling is required further.

5. Reliability and Deployment Issues

Network or Hosts Dropping SYN packets with data or unknown options

A study [MAF04] found that some middle-boxes and end-hosts may drop packets with unknown TCP options incorrectly. Studies [LANGLEY06, HNRGHT11] both found that 6% of the probed paths on the Internet drop SYN packets with data or with unknown TCP options. The TFO protocol deals with this problem by retransmitting SYN without data or cookie options and we recommend tracking these servers in the client.

Server Farms

A common server-farm setup is to have many physical hosts behind a load-balancer sharing the same server IP. The load-balancer forwards new TCP connections to different physical hosts based on certain load-balancing algorithms. For TFO to work, the physical hosts need to share the same key and update the key at about the same time.

Network Address Translation (NAT)

The hosts behind NAT sharing same IP address will get the same cookie to the same server. This will not prevent TFO from working. But on some carrier-grade NAT configurations where every new TCP connection from the same physical host uses a different public IP address, TFO does not provide latency benefit. However, there is no performance penalty either as described in Section "Client: Receiving SYN-ACK".

6. Security Considerations

The Fast Open cookie stops an attacker from trivially flooding spoofed SYN packets with data to burn server resources or to mount an

amplified reflection attack on random hosts. The server can defend against spoofed SYN floods with invalid cookies using existing techniques [RFC4987]. We note that generating bogus cookies is usually cheaper than validating them. But the additional cost of validating the cookies, inherent to any authentication scheme, may not be substantial compared to processing a regular SYN packet.

However, the attacker may still obtain cookies from some compromised hosts, then flood spoofed SYN with data and "valid" cookies (from these hosts or other vantage points). With DHCP, it's possible to obtain cookies of past IP addresses without compromising any host. Below we identify new vulnerabilities of TFO and describe the countermeasures.

6.1. Server Resource Exhaustion Attack by SYN Flood with Valid Cookies

Like regular TCP handshakes, TFO is vulnerable to such an attack. But the potential damage can be much more severe. Besides causing temporary disruption to service ports under attack, it may exhaust server CPU and memory resources.

For this reason it is crucial for the TFO server to limit the maximum number of total pending TFO connection requests, i.e., PendingFastOpenRequests. When the limit is exceeded, the server temporarily disables TFO entirely as described in "Server Cookie Handling". Then subsequent TFO requests will be downgraded to regular connection requests, i.e., with the data dropped and only SYN acknowledged. This allows regular SYN flood defense techniques [RFC4987] like SYN-cookies to kick in and prevent further service disruption.

There are other subtle but important differences in the vulnerability between TFO and regular TCP handshake. Before the SYN flood attack broke out in the late '90s, typical listener's max qlen was small, enough to sustain the highest expected new connection rate and the average RTT for the SYN-ACK packets to be acknowledged in time. E.g., if a server is designed to handle at most 100 connection requests per second, and the average RTT is 100ms, a max qlen on the order of 10 will be sufficient.

This small max qlen made it very easy for any attacker, even equipped with just a dailup modem to the Internet, to cause major disruptions to a web site by simply throwing a handful of "SYN bombs" at its victim of choice. But for this attack scheme to work, the attacker must pick a non-responsive source IP address to spoof with. Otherwise the SYN-ACK packet will trigger TCP RST from the host whose IP address has been spoofed, causing corresponding connection to be removed from the server's listener queue hence defeating the attack.

In other words, the main damage of SYN bombs against the standard TCP stack is not directly from the bombs themselves costing TCP processing overhead or host memory, but rather from the spoofed SYN packets filling up the often small listener's queue.

On the other hand, TFO SYN bombs can cause damage directly if admitted without limit into the stack. The RST packets from the spoofed host will fuel rather than defeat the SYN bombs as compared to the non-TFO case, because the attacker can flood more SYNs with data to cost more data processing resources. For this reason, a TFO server needs to monitor the connections in SYN-RCVD being reset in addition to imposing a reasonable max qlen. Implementations may combine the two, e.g., by continuing to account for those connection requests that have just been reset against the listener's PendingFastOpenRequests until a timeout period has passed.

Limiting the maximum number of pending TFO connection requests does make it easy for an attacker to overflow the queue, causing TFO to be disabled. We argue that causing TFO to be disabled is unlikely to be of interest to attackers because the service will remain intact without TFO hence there is hardly any real damage.

6.2. Amplified Reflection Attack to Random Host

Limiting PendingFastOpenRequests with a system limit can be done without Fast Open Cookies and would protect the server from resource exhaustion. It would also limit how much damage an attacker can cause through an amplified reflection attack from that server. However, it would still be vulnerable to an amplified reflection attack from a large number of servers. An attacker can easily cause damage by tricking many servers to respond with data packets at once to any spoofed victim IP address of choice.

With the use of Fast Open Cookies, the attacker would first have to steal a valid cookie from its target victim. This likely requires the attacker to compromise the victim host or network first.

The attacker here has little interest in mounting an attack on the victim host that has already been compromised. But she may be motivated to disrupt the victim's network. Since a stolen cookie is only valid for a single server, she has to steal valid cookies from a large number of servers and use them before they expire to cause sufficient damage without triggering the defense in the previous section.

One can argue that if the attacker has compromised the target network or hosts, she could perform a similar but simpler attack by injecting bits directly. The degree of damage will be identical, but TFO-

specific attack allows the attacker to remain anonymous and disguises the attack as from other servers.

The best defense is for the server not to respond with data until handshake finishes. In this case the risk of amplification reflection attack is completely eliminated. But the potential latency saving from TFO may diminish if the server application produces responses earlier before the handshake completes.

6.3 Attacks from behind sharing public IPs (NATs)

An attacker behind NAT can easily obtain valid cookies to launch the above attack to hurt other clients that share the path. [BOB12] suggested that the server can extend cookie generation to include the TCP timestamp---GetCookie(IP_Address, Timestamp)---and implement it by encrypting the concatenation of the two values to generate the cookie. The client stores both the cookie and its corresponding timestamp, and echoes both in the SYN. The server then implements IsCookieValid(IP_Address, Timestamp, Cookie) by encrypting the IP and timestamp data and comparing it with the cookie value.

This enables the server to issue different cookies to clients that share the same IP address, hence can selectively discard those misused cookies from the attacker. However the attacker can simply repeat the attack with new cookies. The server would eventually need to throttle all requests from the IP address just like the current approach. Moreover this approach requires modifying [RFC 1323] to send non-zero Timestamp Echo Reply in SYN, potentially cause firewall issues. Therefore we believe the benefit may not outweigh the drawbacks.

7. Web Performance

7.1. HTTP persistent connection

TCP connection setup overhead has long been identified as a performance bottleneck for web applications [THK98]. HTTP persistent connection was proposed to mitigate this issue and has been widely deployed. However, [RCCJR11][AERG11] show that the average number of transactions per connection is between 2 and 4, based on large-scale measurements from both servers and clients. In these studies, the servers and clients both kept the idle connections up to several minutes, well into the human think time.

Can the utilization rate increase by keeping connections even longer? Unfortunately, this is problematic due to middle-boxes and rapidly growing mobile end hosts. One major issue is NAT. Studies

[HNESSK10][MQXMZ11] show that the majority of home routers and ISPs fail to meet the the 124 minutes idle timeout mandated in [RFC5382]. In [MQXMZ11], 35% of mobile ISPs timeout idle connections within 30 minutes. NAT boxes do not possess a reliable mechanism to notify end hosts when idle connections are removed from local tables, either due to resource constraints such as mapping table size, memory, or lookup overhead, or due to the limited port number and IP address space. Moreover, unmapped packets received by NAT boxes are often dropped silently. (TCP RST is not required by RFC5382.) The end host attempting to use these broken connections are often forced to wait for a lengthy TCP timeout. Thus the browser risks large performance penalty when keeping idle connections open. To circumvent this problem, some applications send frequent TCP keep-alive probes. However, this technique drains power on mobile devices [MQXMZ11]. In fact, power has become a prominent issue in modern LTE devices that mobile browsers close the HTTP connections within seconds or even immediately [SOUDERS11].

Idle connections also consume more memory resources. Due to the complexity of today's web applications, the application layer often needs orders of magnitude more memory than the TCP connection footprint. As a result, servers need to implement advanced resource management in order to support a large number of idle connections.

7.2 Case Study: Chrome Browser

[RCCJR11] studied Chrome browser performance based on 28 days of global statistics. Chrome browser keeps idle HTTP persistent connections up to 5 to 10 minutes. However the average number of the transactions per connection is only 3.3. Due to the low utilization, TCP 3WHS accounts up to 25% of the HTTP transaction network latency. The authors tested a Linux TFO implementation with TFO enabled Chrome browser on popular web sites in emulated environments such as residential broadband and mobile networks. They showed that TFO improves page load time by 10% to 40%. More detailed on the design tradeoffs and measurement can be found at [RCCJB11].

8. TFO's Applicability

TFO aims at latency conscious applications that are sensitive to TCP's initial connection setup delay. These application protocols often employ short-lived TCP connections, or employ long-lived connections but are more sensitive to the connection setup delay due to, e.g., a more strict connection fail-over requirement.

Only transaction-type applications where RTT constitutes a significant portion of the total end-to-end latency will likely benefit from TFO. Moreover, the client request must fit in the SYN

packet. Otherwise there may not be any saving in the total number of round trips required to complete a transaction.

To the extent possible applications protocols SHOULD employ long-lived connections to best take advantage of TCP's built-in congestion control algorithm, and to reduce the impact from TCP's connection setup overhead. E.g., for the web applications, P-HTTP will likely help and is much easier to deploy hence should be attempted first. TFO will likely provide further latency reduction on top of P-HTTP. But the additional benefit will depend on how much persistency one can get from HTTP in a given operating environment.

One alternative to short-lived TCP connection might be UDP, which is connectionless hence doesn't inflict any connection setup delay, and is best suited for application protocols that are transactional. Practical deployment issues such as middle-box and/or firewall traversal may severely limit the use of UDP based application protocols though.

Note that when the application employs too many short-lived connections, it may negatively impact network stability, as these connections often exit before TCP's congestion control algorithm kicks in. Implementations supporting large number of short-lived connections should employ temporal sharing of TCB data as described in [RFC2140].

More discussion on TCP Fast Open and its projected performance benefit can be found in [RCCJB11].

9. Related Work

9.1. T/TCP

TCP Extensions for Transactions [RFC1644] attempted to bypass the three-way handshake, among other things, hence shared the same goal but also the same set of issues as TFO. It focused most of its effort battling old or duplicate SYNs, but paid no attention to security vulnerabilities it introduced when bypassing 3WHS. Its TAO option and connection count, besides adding complexity, require the server to keep state per remote host, while still leaving it wide open for attacks. It is trivial for an attacker to fake a CC value that will pass the TAO test. Unfortunately, in the end its scheme is still not 100% bullet proof as pointed out by [PHRACK98].

As stated earlier, we take a practical approach to focus TFO on the security aspect, while allowing old, duplicate SYN packets with data after recognizing that 100% TCP semantics is likely infeasible. We

believe this approach strikes the right tradeoff, and makes TFO much simpler and more appealing to TCP implementers and users.

9.2. Common Defenses Against SYN Flood Attacks

TFO is still vulnerable to SYN flood attacks just like normal TCP handshakes, but the damage may be much worse, thus deserves a careful thought.

There have been plenty of studies on how to mitigate attacks from regular SYN flood, i.e., SYN without data [RFC4987]. But from the stateless SYN-cookies to the stateful SYN Cache, none can preserve data sent with SYN safely while still providing an effective defense.

The best defense may be to simply disable TFO when a host is suspected to be under a SYN flood attack, e.g., the SYN backlog is filled. Once TFO is disabled, normal SYN flood defenses can be applied. The "Security Consideration" section contains a thorough discussion on this topic.

9.3. TCP Cookie Transaction (TCPCT)

TCPCT [RFC6013] eliminates server state during initial handshake and defends spoofing DoS attacks. Like TFO, TCPCT allows SYN and SYN-ACK packets to carry data. However, TCPCT and TFO are designed for different goals and they are not compatible.

The TCPCT server does not keep any connection state during the handshake, therefore the server application needs to consume the data in SYN and (immediately) produce the data in SYN-ACK before sending SYN-ACK. Otherwise the application's response has to wait until handshake completes. In contrary, TFO allows server to respond data during handshake. Therefore for many request-response style applications, TCPCT may not achieve same latency benefit as TFO.

Rapid-Restart [SIMPSON11] is based on TCPCT and shares similar goal as TFO. In Rapid-Restart, both the server and the client retain the TCP control blocks after a connection is terminated in order to allow/resume data exchange in next connection handshake. In contrary, TFO does not require keeping both TCB on both sides and is more scalable.

10. IANA Considerations

The Fast Open Cookie Option and Fast Open Cookie Request Option define no new namespace. The options require IANA allocate one value from the TCP option Kind namespace. Early implementation before the

allocation SHOULD follow [EXPOPT] and use experimental option 254 and magic number 0xF989 (16 bits), and migrate to the new option after the allocation according.

11. Acknowledgement

We thank Rick Jones, Bob Briscoe, Adam Langley, Matt Mathis, Neal Cardwell, Roberto Peon, and Tom Herbert for their feedbacks. We especially thank Barath Raghavan for his contribution on the security design of Fast Open.

12. References

12.1. Normative References

[RFC793] Postel, J. "Transmission Control Protocol", RFC 793, September 1981.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC5382] S. Guha, Ed., Biswas, K., Ford B., Sivakumar S., Srisuresh, P., "NAT Behavioral Requirements for TCP", RFC 5382

[RFC5681] Allman, M., Paxson, V. and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC6298] Paxson, V., Allman, M., Chu, J. and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

12.2. Informative References

[AERG11] M. Al-Fares, K. Elmeleegy, B. Reed, and I. Gashinsky, "Overclocking the Yahoo! CDN for Faster Web Page Loads". In Proceedings of Internet Measurement Conference, November 2011.

[CDCM11] Chu, J., Dukkipati, N., Cheng, Y. and M. Mathis, "Increasing TCP's Initial Window", Internet-Draft draft-ietf-tcpm-initcwnd-02.txt (work in progress), October 2011.

[EXPOPT] Touch, Joe, "Shared Use of Experimental TCP Options", Internet-Draft draft-ietf-tcpm-experimental-options (work in progress), October 2012.

[HNESSK10] S. Haetoenen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, M. Kojo., "An Experimental Study of Home Gateway Characteristics". In Proceedings of Internet Measurement Conference. Octobor 2010

[HNRGHT11] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, "Is it Still Possible to Extend TCP?". In Proceedings of Internet Measurement Conference. November 2011.

[LANGLEY06] Langley, A, "Probing the viability of TCP extensions", URL http://www.imperialviolet.org/binary/ecntest.pdf

[MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring

Interactions Between Transport Protocols and Middleboxes", In Proceedings of Internet Measurement Conference, October 2004.

[MQXMZ11] Z. Mao, Z. Qian, Q. Xu, Z. Mao, M. Zhang. "An Untold Story of Middleboxes in Cellular Networks", In Proceedings of SIGCOMM. August 2011.

[PHRACK98] "T/TCP vulnerabilities", Phrack Magazine, Volume 8, Issue 53 artical 6. July 8, 1998. URL http://www.phrack.com/issues.html?issue=53&id=6

[QWGMSS11] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck. "Profiling Resource Usage for Mobile Applications: A Cross-layer Approach", In Proceedings of International Conference on Mobile Systems. April 2011.

[RCCJB11] Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A. and B. Raghavan, "TCP Fast Open". In Proceedings of 7th ACM CoNEXT Conference, December 2011.

[RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC2140, April 1997.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC6013, January 2011.

[SIMPSON11] Simpson, W., "Tcp cookie transactions (tcpct) rapid restart", Internet draft draft-simpson-tcpct-rr-02.txt (work in progress), July 2011.

[SOUDERS11] S. Souders. "Making A Mobile Connection". http://www.stevesouders.com/blog/2011/09/21/making-a-mobile-connection/

[THK98] Touch, J., Heidemann, J., Obraczka, K., "Analysis of HTTP Performance", USC/ISI Research Report 98-463. December 1998.

[BOB12] Briscoe, B., "Some ideas building on draft-ietf-tcpm-fastopen-01", tcpm list, http://www.ietf.org/mail-archive/web/tcpm/current/

msg07192.html

Author's Addresses

Yuchung Cheng
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: ycheng@google.com

Jerry Chu
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: hkchu@google.com

Sivasankar Radhakrishnan
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Dr
La Jolla, CA 92093-0404
EMail: sivasankar@cs.ucsd.edu

Arvind Jain
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: arvind@google.com