Internet Engineering Task Force (IETF)
Request for Comments: 7413
Category: Experimental
ISSN: 2070-1721
December 2014

Y. Cheng
J. Chu
S. Radhakrishnan
A. Jain
Google

TCP Fast Open

Abstract

This document describes an experimental TCP mechanism called TCP Fast Open (TFO). TFO allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiving end during the initial connection handshake, and saves up to one full round-trip time (RTT) compared to the standard TCP, which requires a three-way handshake (3WHS) to complete before data can be exchanged. However, TFO deviates from the standard TCP semantics, since the data in the SYN could be replayed to an application in some rare circumstances. Applications should not use TFO unless they can tolerate this issue, as detailed in the Applicability section.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
   1.1. Terminology 4
2. Data in SYN 4
   2.1. Relaxing TCP Semantics on Duplicated SYNs 4
   2.2. SYNs with Spoofed IP Addresses 5
3. Protocol Overview 5
4. Protocol Details 7
   4.1. Fast Open Cookie 7
        4.1.1. Fast Open Option 8
        4.1.2. Server Cookie Handling 8
        4.1.3. Client Cookie Handling 9
               4.1.3.1. Client Caching Negative Responses 10
   4.2. Fast Open Protocol 11
        4.2.1. Fast Open Cookie Request 11
        4.2.2. TCP Fast Open 12
5. Security Considerations 14
   5.1. Resource Exhaustion Attack by SYN Flood with Valid
        Cookies 14
        5.1.1. Attacks from behind Shared Public IPs (NATs) 15
   5.2. Amplified Reflection Attack to Random Host 16
6. TFO Applicability 17
   6.1. Duplicate Data in SYNs 17
   6.2. Potential Performance Improvement 17
   6.3. Example: Web Clients and Servers 18
        6.3.1. HTTP Request Replay 18
        6.3.2. HTTP over TLS (HTTPS) 18
        6.3.3. Comparison with HTTP Persistent Connections 18
        6.3.4. Load Balancers and Server Farms 19

7. Open Areas for Experimentation 19
   7.1. Performance Impact Due to Middleboxes and NAT 19
   7.2. Impact on Congestion Control 20
   7.3. Cookie-less Fast Open 20
8. Related Work 20
   8.1. T/TCP 20
   8.2. Common Defenses against SYN Flood Attacks 21
   8.3. Speculative Connections by the Applications 21
   8.4. Fast Open Cookie-in-FIN 21
   8.5. TCP Cookie Transaction (TCPCT) 21
9. IANA Considerations 22
10. References 22
   10.1. Normative References 22
   10.2. Informative References 23
Appendix A. Example Socket API Changes to Support TFO 25
  A.1. Active Open 25
  A.2. Passive Open 25
Acknowledgments 26
Authors' Addresses 26

1. Introduction

TCP Fast Open (TFO) is an experimental update to TCP that 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, potential new security vulnerabilities and their mitigation, and the new socket API.

TFO is motivated by the performance needs of today's Web applications. Current TCP only permits data exchange after the three-way handshake (3WHS) [RFC793], which adds one RTT to network latency. For short Web transfers this additional RTT is a significant portion of overall network latency, even when HTTP persistent connection is widely used. For example, the Chrome browser [Chrome] keeps TCP connections idle for up to 5 minutes, but 35% of HTTP requests are made on new TCP connections [RCCJR11]. For such Web and Web-like applications, placing data in the SYN can yield significant latency improvements. Next we describe how we resolve the challenges that arise upon doing so.

1.1. 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 TCP's active open side, and "server" refers to TCP's passive open side.

2. Data in SYN

Standard TCP already allows data to be carried in SYN packets ([RFC793], Section 3.4) but forbids the receiver from delivering it to the application until the 3WHS is completed. This is because TCP's initial handshake serves to capture old or duplicate SYNs.

To enable applications to exchange data in a TCP handshake, TFO removes the constraint and allows data in SYN packets to be delivered to the application. This change to TCP semantic raises two issues (discussed in the following subsections) that make TFO unsuitable for certain applications.

Therefore, TCP implementations MUST NOT use TFO by default, but only use TFO if requested explicitly by the application on a per-service-port basis. Applications need to evaluate TFO applicability as described in Section 6 before using TFO.

2.1. Relaxing TCP Semantics on Duplicated SYNs

TFO allows data to be delivered to the application before the 3WHS is completed, thus opening itself to a data integrity issue in either of the two cases below:

a) the receiver host receives data in a duplicate SYN after it has forgotten it received the original SYN (e.g., due to a reboot);

b) the duplicate is received after the connection created by the original SYN has been closed and the close was initiated by the sender (so the receiver will not be protected by the TIME-WAIT 2 MSL state).

The now obsoleted T/TCP [RFC1644] (obsoleted by [RFC6247]) attempted to address these issues. It was not successful and not deployed due to various vulnerabilities as described in Section 8, "Related Work". 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 use to a class of applications (Section 6) that are tolerant of duplicate SYN packets with data. We believe this is the right design trade-off: balancing complexity with usefulness.

2.2. SYNs with Spoofed IP Addresses

Standard TCP suffers from the SYN flood attack [RFC4987] because SYN packets with spoofed source IP addresses can easily fill up a listener's small queue, causing a service port to be blocked completely.

TFO goes one step further to allow server-side TCP to send up data to the application layer before the 3WHS is completed. This opens up 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 further mount an amplified reflection attack against victims of choice beyond the TFO server itself.

Numerous mitigation techniques against regular SYN flood attacks exist and have been well documented [RFC4987]. Unfortunately, none are applicable to TFO. We propose a server-supplied cookie to mitigate these new vulnerabilities in Section 3 and evaluate the effectiveness of the defense in Section 7.

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 the 3WHS:

Requesting a Fast Open Cookie:

1. The client sends a SYN with a Fast Open option with an empty cookie field to request a cookie.

2. The server generates a cookie and sends it through the Fast Open 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 data and the cookie in the Fast Open option.

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 maximum amount is governed by 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 a handshake. The cookie is a MAC 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. Fast Open Option

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

```
Kind            1 byte: value = 34
Length          1 byte: range 6 to 18 (bytes); limited by
                        remaining space in the options field.
                        The number MUST be even.
Cookie          0, or 4 to 16 bytes (Length - 2)
```

The Fast Open option is used to request or to send a Fast Open Cookie. When a cookie is not present or is empty, the option is used by the client to request a cookie from the server. When the cookie is present, the option is used to pass the cookie from the server to the client or from the client back to the server (to perform a Fast Open).

The minimum cookie size is 4 bytes. Although the diagram shows a cookie aligned on 32-bit boundaries, alignment is not required. Options with invalid Length values or without the SYN 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 MAC tag with the following properties. We use "SHOULD" because, in some cases, the cookie may be trivially generated as discussed in Section 7.3.

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

2. The cookie can only be generated by the server and cannot 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 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 Considerations" section (Section 5). 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 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 the cookie 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 [RCCJR11].

If only one valid cookie is allowed per IP, 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.

4.1.3. Client Cookie Handling

The client MUST cache cookies from servers for later Fast Open connections. For a multihomed client, the cookies are dependent on the client and server IP addresses. Hence, the client should cache at most one (most recently received) cookie per client and server IP address pair.

When caching cookies, we recommend that the client also cache the Maximum Segment Size (MSS) advertised by the server. The client can cache the MSS advertised by the server in order to determine the maximum amount of data that the client can fit in the SYN packet in subsequent TFO connections. Caching the server MSS is useful because, with Fast Open, a client sends data in the SYN packet before the server announces its MSS in the SYN-ACK packet. If the client sends more data in the SYN packet than the server will accept, this will likely require the client to retransmit some or all of the data. Hence, caching the server MSS can enhance performance.

Without a cached server MSS, the amount of data in the SYN packet is limited to the default MSS of 536 bytes for IPv4 [RFC1122] and 1220 bytes for IPv6 [RFC2460]. Even if the client complies with this limit when sending the SYN, it is known that an IPv4 receiver advertising an MSS less than 536 bytes can receive a segment larger than it is expecting.

If the cached MSS is larger than the typical size (1460 bytes for IPv4 or 1440 bytes for IPv6), then the excess data in the SYN packet may cause problems that offset the performance benefit of Fast Open. For example, the unusually large SYN may trigger IP fragmentation and may confuse firewalls or middleboxes, causing SYN retransmission and other side effects. Therefore, the client MAY limit the cached MSS to 1460 bytes for IPv4 or 1440 for IPv6.

4.1.3.1. Client Caching Negative Responses

The client MUST cache negative responses from the server in order to avoid potential connection failures. Negative responses include the server not acknowledging the data in the SYN, ICMP error messages, and (most importantly) no response (SYN-ACK) from the server at all, i.e., connection timeout. The last case is likely due to incompatible middleboxes or firewall blocking the connection completely after processing the SYN packet with data. If the client does not react to these negative responses and continues to retry Fast Open, the client may never be able to connect to the specific server.

For any negative responses, the client SHOULD disable Fast Open on the specific path (the source and destination IP addresses and ports) at least temporarily. Since TFO is enabled on a per-service-port basis, but cookies are independent of service ports, the client's 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, on both the client and server sides.

The server keeps two variables per listening socket (IP address and 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 the number of TFO connections in SYN-RCVD state. If this variable goes over a preset system limit, the server MUST 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 (Section 5).

The server keeps a FastOpened flag per connection 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 option with a Length field of 0 (empty cookie field).

2. The server responds with a SYN-ACK based on the procedures in the "Server Cookie Handling" section (Section 4.1.2). This SYN-ACK may contain a Fast Open option if the server currently supports TFO for this listener port.

3. If the SYN-ACK has a Fast Open option with a cookie, the client replaces the cookie and other information as described in the "Client Cookie Handling" section (Section 4.1.3). Otherwise, if the SYN-ACK is first seen and not a (spurious) retransmission, the client MAY remove the server information from the cookie cache. If the SYN-ACK is a spurious retransmission, 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 will cause SYN or SYN-ACK timeouts. We RECOMMEND both the client and the server to retransmit SYN and SYN-ACK packets without the cookie options on timeouts. This ensures the connections of cookie requests will go through and lowers the latency penalty (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 until after the 3WHS is justified if the SYN drop is due to network congestion). The next section describes a heuristic to detect such drops when the client receives the SYN-ACK.

We also RECOMMEND the client to record the set of servers that failed to respond to cookie requests and only attempt another cookie request after a certain period.

4.2.2. TCP Fast Open

Once the client obtains the cookie from the target server, it can perform subsequent TFO connections until the cookie is expired by the server.

Client: Sending SYN

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

1. Send a SYN packet.

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

   b. Otherwise, include the Fast Open 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.

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 options.

Server: Receiving SYN and responding with SYN-ACK

Upon receiving the SYN packet with Fast Open 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 the FastOpened flag and increment PendingFastOpenRequests.

5. Send the SYN-ACK packet. The packet MAY include a Fast Open option. If the 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 applications may favor delaying the SYN-ACK, allowing the application to process the request in order to produce a response, but this is left up to the implementation.

6. Advance to the SYN-RCVD state. If the FastOpened flag is set, the server MUST follow [RFC5681] (based on [RFC3390]) to set the initial congestion window for sending more data packets.

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

A special case is simultaneous open where the SYN receiver is a client in SYN-SENT state. The protocol remains the same because [RFC793] already supports both data in the SYN and simultaneous open. But the client's socket may have data available to read before it's connected. This document does not cover the corresponding API change.

Client: Receiving SYN-ACK

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

1. If the SYN-ACK has a Fast Open option, an MSS option, or both, update the corresponding cookie and MSS information in the cookie cache.

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 SYNs with the 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. 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 although generating bogus cookies is cost free, the cost of validating the cookies, inherent to any authentication scheme, may be substantial compared to processing a regular SYN packet. We describe these new vulnerabilities of TFO and the countermeasures in detail below.

5.1. Resource Exhaustion Attack by SYN Flood with Valid Cookies

An attacker may still obtain cookies from some compromised hosts, then flood spoofed SYN packets with data and "valid" cookies (from these hosts or other vantage points). 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. Such an attack will show up on application server logs as an application-level DoS from botnets, triggering other defenses and alerts.

To protect the server, it is important to limit the maximum number of total pending TFO connection requests, i.e., PendingFastOpenRequests (Section 4.2). When the limit is exceeded, the server temporarily disables TFO entirely as described in "Server Cookie Handling" (Section 4.1.2). Then, subsequent TFO requests will be downgraded to regular connection requests, i.e., with the data dropped and only SYNs acknowledged. This allows regular SYN flood defense techniques [RFC4987] like SYN cookies to kick in and prevent further service disruption.

The main impact of SYN floods against the standard TCP stack is not directly from the floods 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 floods can cause damage directly if admitted without limit into the stack. The reset (RST) packets from the spoofed host will fuel rather than defeat the SYN floods as compared to the non-TFO case, because the attacker can flood more SYNs with data and incur more cost in terms of 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 queue length. 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.

5.1.1. Attacks from behind Shared Public IPs (NATs)

An attacker behind a NAT can easily obtain valid cookies to launch the above attack to hurt other clients that share the path. [BRISCOE12] 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 it 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, it 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 [RFC1323] (obsoleted by [RFC7323]) to send a non-zero Timestamp Echo Reply in the SYN, potentially causing firewall issues. Therefore, we believe the benefit does not outweigh the drawbacks.

5.2. 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. But, in some cases, it may be relatively easy.

The attacker here has little interest in mounting an attack on the victim host that has already been compromised. But it may be motivated to disrupt the victim's network. Since a stolen cookie is only valid for a single server, it 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.

One can argue that if the attacker has compromised the target network or hosts, it could perform a similar but simpler attack by injecting bits directly. The degree of damage will be identical, but a TFO-specific attack allows the attacker to remain anonymous and disguises the attack as from other servers.

For example, with DHCP, an attacker can obtain cookies when he (or the host he has compromised) owns a particular IP address by performing regular Fast Open to servers supporting TFO and he can collect valid cookies. Then, the attacker actively or passively releases his IP address. When the IP address is reassigned to another host (victim) via DHCP, the attacker then floods spoofed Fast Open requests with valid cookies to the servers. Since the cookies are valid, these servers accept the requests and respond with a SYN-ACK plus data packets to the victim instead of the attacker. Thus, the attacker is able to launch amplified reflection attacks to other hosts that share IP addresses.

The best defense is for the server not to respond with data until the handshake finishes. In this case, the risk of an 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. TFO Applicability

This section is to help applications considering TFO to evaluate TFO's benefits and drawbacks using the Web client and server applications as an example throughout. Applications here refer specifically to the process that writes data into the socket -- for example, a JavaScript process that sends data to the server. A proposed socket API change is provided in the Appendix.

6.1. Duplicate Data in SYNs

It is possible that using TFO results in the first data written to a socket to be delivered more than once to the application on the remote host (Section 2.1). This replay potential only applies to data in the SYN but not subsequent data exchanges.

Empirically, [JIDKT07] showed the packet duplication on a Tier-1 network is rare. Since the replay only happens specifically when the SYN data packet is duplicated and also the duplicate arrives after the receiver has cleared the original SYN's connection state, the replay is thought to be uncommon in practice. Nevertheless, a client that cannot handle receiving the same SYN data more than once MUST NOT enable TFO to send data in a SYN. Similarly, a server that cannot accept receiving the same SYN data more than once MUST NOT enable TFO to receive data in a SYN. Further investigation is needed to judge the probability of receiving duplicated SYN or SYN-ACK packets with data in networks that are not Tier 1.

6.2. Potential Performance Improvement

TFO is designed for latency-conscious applications that are sensitive to TCP's initial connection setup delay. To benefit from TFO, the first application data unit (e.g., an HTTP request) needs to be no more than TCP's maximum segment size (minus options used in the SYN). Otherwise, the remote server can only process the client's application data unit once the rest of it is delivered after the initial handshake, diminishing TFO's benefit.

To the extent possible, applications SHOULD reuse the connection to take advantage of TCP's built-in congestion control and reduce connection setup overhead. An application that employs too many short-lived connections will negatively impact network stability, as these connections often exit before TCP's congestion control algorithm takes effect.

6.3. Example: Web Clients and Servers

6.3.1. HTTP Request Replay

While TFO is motivated by Web applications, the browser should not use TFO to send requests in SYNs if those requests cannot tolerate replays. One example is POST requests without application-layer transaction protection (e.g., a unique identifier in the request header).

On the other hand, TFO is particularly useful for GET requests. GET request replay could happen across striped TCP connections: after a server receives an HTTP request but before the ACKs of the requests reach the browser, the browser may time out and retry the same request on another (possibly new) TCP connection. This differs from a TFO replay only in that the replay is initiated by the browser, not by the TCP stack.

6.3.2. HTTP over TLS (HTTPS)

For Transport Layer Security (TLS) over TCP, it is safe and useful to include a TLS client_hello in the SYN packet to save one RTT in the TLS handshake. There is no concern about violating idempotency. In particular, it can be used alone with the speculative connection above.

6.3.3. Comparison with HTTP Persistent Connections

Is TFO useful given the wide deployment of HTTP persistent connections? The short answer is yes. Studies ([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 idle connections up to several minutes, well into "human think" time.

Keeping connections open and idle even longer risks a greater performance penalty. [HNESSK10] and [MQXMZ11] show that the majority of home routers and ISPs fail to meet the 124-minute idle timeout mandated in [RFC5382]. In [MQXMZ11], 35% of mobile ISPs silently time out idle connections within 30 minutes. End hosts, unaware of silent middlebox timeouts, suffer multi-minute TCP timeouts upon using those long-idle connections.

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 such a prominent issue in modern Long Term Evolution (LTE) devices that mobile browsers close HTTP connections within seconds or even immediately [SOUDERS11].

[RCCJR11] studied the performance of the Chrome browser [Chrome] based on 28 days of global statistics. The Chrome browser keeps idle HTTP persistent connections for 5 to 10 minutes. However, the average number of the transactions per connection is only 3.3, and the TCP 3WHS accounts for up to 25% of the HTTP transaction network latency. The authors estimated that TFO improves page load time by 10% to 40% on selected popular Web sites.

6.3.4. Load Balancers and Server Farms

Servers behind load balancers that accept connection requests to the same server IP address should use the same key such that they generate identical Fast Open cookies for a particular client IP address. Otherwise, a client may get different cookies across connections; its Fast Open attempts would fall back to the regular 3WHS.

7. Open Areas for Experimentation

We now outline some areas that need experimentation in the Internet and under different network scenarios. These experiments should help evaluate Fast Open benefits and risks and its related protocols.

7.1. Performance Impact Due to Middleboxes and NAT

[MAF04] found that some middleboxes and end hosts may drop packets with unknown TCP options. Studies ([LANGLEY06] [HNRGHT11]) have 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 falling back to the regular TCP handshake and retransmitting the SYN without data or cookie options after the initial SYN timeout. Moreover, the implementation is recommended to negatively cache such incidents to avoid recurring timeouts. Further study is required to evaluate the performance impact of these drop behaviors.

Another interesting study is the loss of TFO performance benefit behind certain Carrier-Grade NAT (CGN). Typically, hosts behind a NAT sharing the same IP address will get the same cookie for the same server. This will not prevent TFO from working. But, on some CGN configurations where every new TCP connection from the same physical host uses a different public IP address, TFO does not provide latency benefits. However, there is no performance penalty either, as described in the "Client: Receiving SYN-ACK" text in Section 4.2.2.

7.2. Impact on Congestion Control

Although TFO does not directly change TCP's congestion control, there are subtle cases where it could do so. When a SYN-ACK times out, regular TCP reduces the initial congestion window before sending any data [RFC5681]. However, in TFO, the server may have already sent up to an initial window of data.

If the server serves mostly short connections, then the losses of SYN-ACKs are not as effective as regular TCP on reducing the congestion window. This could result in an unstable network condition. The connections that experience losses may attempt again and add more load under congestion. A potential solution is to temporarily disable Fast Open if the server observes many SYN-ACK or data losses during the handshake across connections. Further experimentation regarding the congestion control impact will be useful.

7.3. Cookie-less Fast Open

The cookie mechanism mitigates resource exhaustion and amplification attacks. However, cookies are not necessary if the server has application-level protection or is immune to these attacks. For example, a Web server that only replies with a simple HTTP redirect response that fits in the SYN-ACK packet may not care about resource exhaustion.

For such applications the server may choose to generate a trivial or even a zero-length cookie to improve performance by avoiding the cookie generation and verification. If the server believes it's under a DoS attack through other defense mechanisms, it can switch to regular Fast Open for listener sockets.

8. Related Work

8.1. T/TCP

TCP Extensions for Transactions [RFC1644] attempted to bypass the 3WHS, among other things; hence, it 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 the 3WHS [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 trade-off and makes TFO much simpler and more appealing to TCP implementers and users.

8.2. Common Defenses against SYN Flood Attacks

[RFC4987] studies the mitigation of attacks from regular SYN floods, i.e., SYNs without data. But from the stateless SYN cookies to the stateful SYN Cache, none can preserve data sent with SYNs safely while still providing an effective defense.

The best defense may be simply to 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 Considerations" section (Section 5) contains a thorough discussion on this topic.

8.3. Speculative Connections by the Applications

Some Web browsers maintain a history of the domains for frequently visited Web pages. The browsers then speculatively pre-open TCP connections to these domains before the user initiates any requests for them [BELSHE11]. While this technique also saves the handshake latency, it wastes server and network resources by initiating and maintaining idle connections.

8.4. Fast Open Cookie-in-FIN

An alternate proposal is to request a TFO cookie in the FIN instead, since FIN-drop by incompatible middleboxes does not affect latency. However, paths that block SYN cookies may be more likely to drop a later SYN packet with data, and many applications close a connection with RST instead anyway.

Although cookie-in-FIN may not improve robustness, it would give clients using a single connection a latency advantage over clients opening multiple parallel connections. If experiments with TFO find that it leads to increased connection-sharding, cookie-in-FIN may prove to be a useful alternative.

8.5. TCP Cookie Transaction (TCPCT)

TCPCT [RFC6013] eliminates server state during the initial handshake and defends spoofing DoS attacks. Like TFO, TCPCT allows SYN and SYN-ACK packets to carry data. But the server can only send up to MSS bytes of data during the handshake instead of the initial congestion window, unlike TFO. Therefore, the latency of applications (e.g., Web applications) may be worse than with TFO.

9. IANA Considerations

IANA has allocated one value, 34, in the "TCP Option Kind Numbers" registry. See Section 4.1.1. The length of this new TCP option is variable, and the Meaning as shown in the "TCP Option Kind Numbers" registry is set to "TCP Fast Open Cookie". Current and new implementations SHOULD use option (34). Existing implementations that are using experimental option 254 per [RFC6994] with magic number 0xF989 (16 bits) as allocated in the IANA "TCP Experimental Option Experiment Identifiers (TCP ExIDs)" registry by this document, SHOULD migrate to use this new option (34) by default.

10. References

10.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002, <http://www.rfc-editor.org/info/rfc3390>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, October 2008, <http://www.rfc-editor.org/info/rfc5382>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009, <http://www.rfc-editor.org/info/rfc5681>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, August 2013, <http://www.rfc-editor.org/info/rfc6994>.

10.2. Informative References

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

[BELSHE11] Belshe, M., "The Era of Browser Preconnect", February 2011, <http://www.belshe.com/2011/02/10/the-era-of-browser-preconnect/>.

[BRISCOE12] Briscoe, B., "Some ideas building on draft-ietf-tcpm-fastopen-01", message to the tcpm mailing list, July 2012, <http://www.ietf.org/mail-archive/web/tcpm/current/msg07192.html>.

[Chrome] Google Chrome, <https://www.google.com/intl/en-US/chrome/browser/>.

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

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

[JIDKT07] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., and D. Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone" IEEE/ACM Transactions on Networking (TON), Volume 15, Issue 1, pp 54-66.

[LANGLEY06] Langley, A., "Probing the viability of TCP extensions", <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] Wang, Z., Qian, Z., Xu, Q., Mao, Z., and 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, Article 6, July 8, 1998, <http://www.phrack.com/issues.html?issue=53&id=6>.

[RCCJR11] Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., and B. Raghavan, "TCP Fast Open", in Proceedings of the 7th ACM CoNEXT Conference, December 2011.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992, <http://www.rfc-editor.org/info/rfc1323>.

[RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994, <http://www.rfc-editor.org/info/rfc1644>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007, <http://www.rfc-editor.org/info/rfc4987>.

[RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011, <http://www.rfc-editor.org/info/rfc6013>.

[RFC6247] Eggert, L., "Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status", RFC 6247, May 2011, <http://www.rfc-editor.org/info/rfc6247>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, September 2014, <http://www.rfc-editor.org/info/rfc7323>.

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

Appendix A. Example Socket API Changes to Support TFO

A.1. Active Open

The active open side involves changing or replacing the connect() call, which does not take a user data buffer argument. We recommend replacing the connect() call to minimize API changes, and, hence, applications to reduce the deployment hurdle.

One solution implemented in Linux 3.7 is introducing a new flag, MSG_FASTOPEN, for sendto() or sendmsg(). MSG_FASTOPEN marks the attempt to send data in the SYN like a combination of connect() and sendto(), by performing an implicit connect() operation. It blocks until the handshake has completed and the data is buffered.

For a non-blocking socket, it returns the number of bytes buffered and sent in the SYN packet. If the cookie is not available locally, it returns -1 with errno EINPROGRESS, and sends a SYN with a TFO cookie request automatically. The caller needs to write the data again when the socket is connected. On errors, it returns the same errno as connect() if the handshake fails.

An implementation may prefer not to change the sendmsg() call because TFO is a TCP-specific feature. A solution is to add a new socket option, TCP_FASTOPEN, for TCP sockets. When the option is enabled before a connect() operation, sendmsg() or sendto() will perform a Fast Open operation similar to the MSG_FASTOPEN flag described above. This approach, however, requires an extra setsockopt() system call.

A.2. Passive Open

The passive open side change is simpler compared to the active open side. The application only needs to enable the reception of Fast Open requests via a new TCP_FASTOPEN setsockopt() socket option before listen().

The option enables Fast Open on the listener socket. The option value specifies the PendingFastOpenRequests threshold, i.e., the maximum length of pending SYNs with data payload. Once enabled, the TCP implementation will respond with TFO cookies per request.

Traditionally, accept() returns only after a socket is connected. But, for a Fast Open connection, accept() returns upon receiving a SYN with a valid Fast Open cookie and data, and the data is available to be read through, e.g., recvmsg(), read().

Acknowledgments

We thank Bob Briscoe, Michael Scharf, Gorry Fairhurst, Rick Jones, Roberto Peon, William Chan, Adam Langley, Neal Cardwell, Eric Dumazet, and Matt Mathis for their feedback. We especially thank Barath Raghavan for his contribution on the security design of Fast Open and proofreading this document numerous times.

Authors' Addresses

Yuchung Cheng
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States

EMail: ycheng@google.com

Jerry Chu
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States

EMail: hkchu@google.com

Sivasankar Radhakrishnan
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0404
United States

EMail: sivasankar@cs.ucsd.edu

Arvind Jain
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States

EMail: arvind@google.com