Internet Draft
draft-cheng-tcpm-fastopen-00.txt
Intended status: Experimental
Creation date: March 7, 2011
Expiration date: September 8, 2011

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

TCP Fast Open

Status of this Memo

Copyright Notice

Abstract

TCP Fast Open (TFO) allows data to be carried in the SYN or SYN-ACK packets and consumed by the receiving end during the initial connection handshake, thus providing a saving of up to one full round trip time (RTT) compared to standard TCP requiring 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 connection handshake.

This document describes a design that enables qualified applications to attain a round trip saving 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. The 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 on deployment issues and related proposals. TFO requires extensions to the existing socket API, which will be covered in a separate document.

TFO is motivated by the performance need 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 transmission delay and propagation delay. Network bandwidth has grown substantially over the past two decades, much reducing the transmission 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 [CDCM10].

Standard TCP only permits data exchange after 3WHS [RFC793], which introduces one RTT delay to the network latency. For short transfers, e.g., web objects, this additional RTT becomes 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. E.g., the Chrome browser keeps TCP connections idle up to 4 minutes but 35% of Chrome HTTP requests are made on new TCP connections.

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

- Old or duplicate SYNs

- 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 dubious SYN packets above.

2.1. TCP Semantics and Duplicate SYNs

A past proposal called T/TCP 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 per remote peer basis, and is vulnerable to IP spoofing attack. Moreover, it has been shown that even with all the complexity, T/TCP is still not 100% bullet proof. Old or duplicate SYNs may still slip through and get accepted by a T/TCP server [PHRACK98].

Rather than trying to capture all the dubious SYN packets to make TFO 100% compatible with TCP semantics, we've made a design decision early on to accept old SYN packets with data, i.e., to allow TFO for a class of applications that are tolerant of duplicate SYN packets with data, e.g., idempotent or query type transactions. We believe this is the right design trade-off balancing complexity with usefulness. There is a large class of applications that can tolerate dubious transaction requests.

For this reason, TFO MUST be disabled by default, and only enabled 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 faked SYN requests taking up memory space. This is normally not an issue today with typical servers having 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 much 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 risks introduced by TFO. A more thorough discussion on SYN flood attack against TFO is deferred 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 Fast Open Cookie:

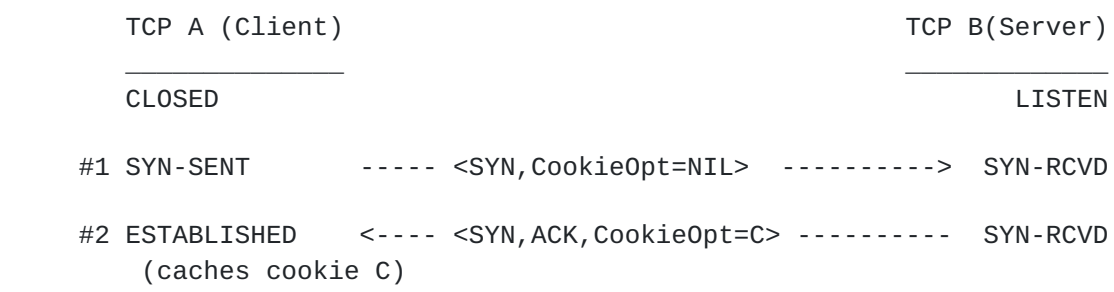
1. The client sends a SYN with a Fast Open Cookie Request option.
2. The server generates a cookie and sends it through the Fast Open Cookie option of a SYN-ACK packet.
3. The client caches the cookie for future TCP Fast Open connections (see below).

Performing TCP Fast Open:

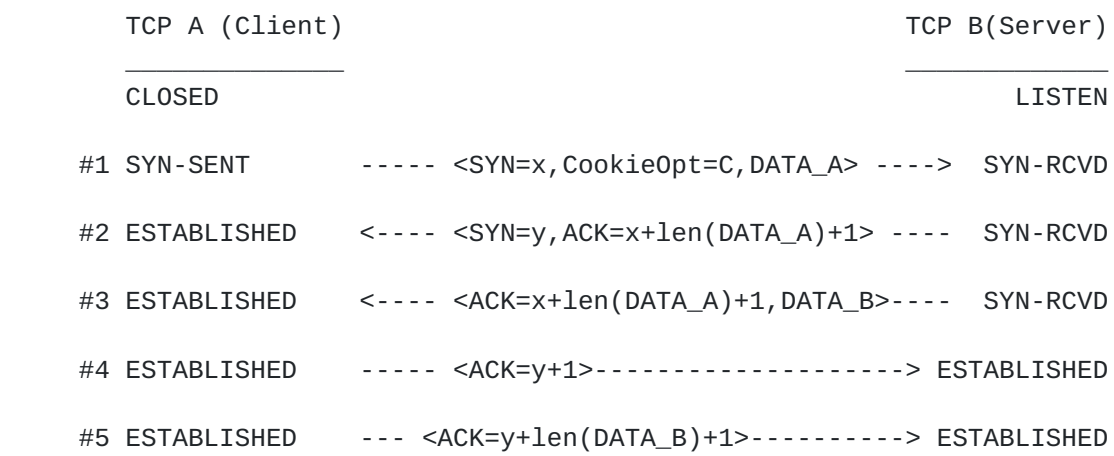
1. The client sends a SYN with Fast Open Cookie option and data.
2. The server validates the cookie:
   a. If the cookie is valid, the server sends a SYN-ACK acknowledging both the SYN and the data. The server then delivers the data to the application.
   b. Otherwise, the server drops the data and sends 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 perform many TFO 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:



Performing TCP Fast Open in connection 2:



4. Protocol Details

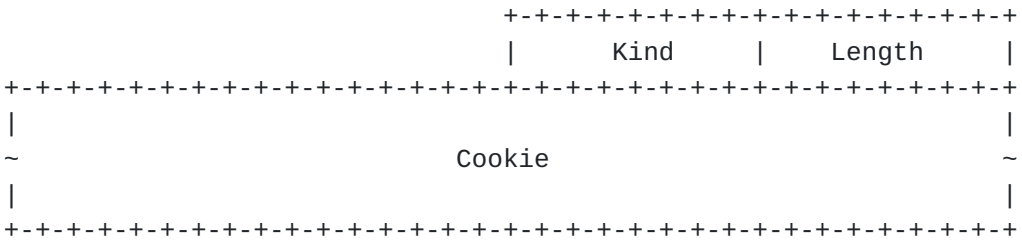
4.1. Fast Open Cookie

The Fast Open Cookie is invented 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.

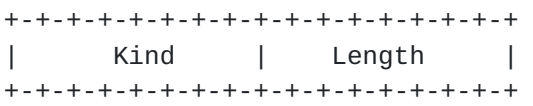


Kind	1 byte: constant TBD (assigned by IANA)
Length	1 byte: range 6 to 18 (bytes); limited by remaining space in the options field. The number MUST be even.
Cookie	4 to 16 bytes (Length - 2)

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, that 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	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 cookie expires after a certain amount of time. The reason 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 in the cookie.

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

5. A server may encode other information in the cookie, and allow more than one valid cookie per client at any given time. But this is all server implementation dependent and transparent to the client. A client only needs to remember one valid cookie per server IP.

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.

4.1.3. Client Cookie Handling

The client MUST cache cookies from different 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].

Caching RTT allows seeding a more accurate SYN timeout than the default value [RFC2988]. 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.

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 the system limit, the server SHOULD set FastOpenEnabled off. 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 valid 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. We also RECOMMEND the client to record servers that failed to respond to cookie requests and only attempt another cookie request after certain period.

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 [RFC2988].

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. If the cookie is invalid, i.e., the cookie does not authenticate the source IP address of the SYN packet, send a SYN-ACK packet acknowledging only the SYN sequence. In addition, include a Fast Open Cookie Option with a new cookie. Go to step 7.

2. If PendingFastOpenRequests is over the system limit, reset FastOpenEnabled flag and send a SYN-ACK acknowledging only the SYN sequence. Go to step 7.

3. Send the SYN-ACK packet acknowledging the SYN and data sequence. The server MAY include data in the SYN-ACK packet.

4. Buffer the data and notify the application.

5. Set FastOpened flag and increment PendingFastOpenRequests.

6. The server MAY send more data packets before the handshake completes. The maximum amount is ruled by the initial congestion window and the receiver window [RFC3390].

7. Advance to the SYN-RCVD state.

If the SYN-ACK timer fires, the server SHOULD retransmit a SYN-ACK packet without data and 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.

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 will retransmit it in the ACK packet. The data exchange will start after the handshake like a regular TCP connection.

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. Another study [LANGLEY06] found that 6% of the probed paths on the Internet drop SYN packets with data. 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].

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 two 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 to not respond with data until handshake finishes, i.e., disallow step 6 in "Server receiving SYN-

ACK" section. 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.

7. Related Work

7.1. 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.

7.2. 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.

7.3. 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.

Without state kept on the server side, TCPCP relies on the client side to retransmit the SYN request with data in order to recover from possible loss of packet from server response. This may cause a lot more dubious connection requests. It also limits the response to only one packet, to fit completely within the SYN-ACK packet. For some TCP applications, in particular web applications, this does not provide enough latency benefit by sending one data packet one RTT earlier.

8. 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.

9. Acknowledgements

The authors would like to thank Tom Herbert, Adam Langley, Roberto Peon, Mathew Mathis, and Barath Raghavan for their insightful comments.

10. References

10.1. 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.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

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

10.2. Informative References

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

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

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

[CDCM10] Chu, J., Dukkipati, N., Cheng, Y. and M. Mathis, "Increasing TCP's Initial Window", Internet-Draft draft-ietf-tcpm-initcwnd-00.txt (work in progress), October 2010.

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

[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

[MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", Proceedings 4th ACM SIGCOMM/USENIX Conference on Internet Measurement, October 2004.

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

Author's Addresses

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

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

Sivasankar Radhakrishnan
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: sivasankar@google.com

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

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.