TCP Fast Open

References

TCP Fast Open (TFO) mechanism is described in https://tools.ietf.org/html/rfc7413

Other considerations can be found in this white paper.

Patches

TFO is currently available available on Linux (full implementation in 4.1), OS X 10.12 and server side will be available for FreeBSd 10/11. There are differences between the implementations described below.

Patches for ldns, NSD and Unbound can be found in our Stash repo, which currently support TFO as implemented on Linux.

https://portal.sinodun.com/stash/projects/TDNS/repos/tcp_fast_open_patches/browse

Note that getdns 1.0 supports TFO on Linux and OS X.

To use these patches:

1. Download the source code from the relevant repository
2. Apply the relevant patch
3. **IMPORTANT**! Run *autoreconf*
4. Add *--enable-tcp-fastopen* to the *configure* options
5. *make & make install* as normal

Quick guide

TCP fastopen [I-D.ietf-tcpm-fastopen] (TFO) allows data to be carried in the SYN packet (and optionally in the SYN-ACK). It therefore saves up to one RTT compared to standard TCP. TFO clients request a server cookie in the initial SYN packet at the
start of a new connection. The server returns a cookie in its SYN-ACK. The client caches the cookie and reuses it when opening subsequent connections to the same server. The cookie is stored by the client's TCP stack (kernel) and persists if either the client or server processes are restarted. TFO also falls back to a regular TCP handshake gracefully.

Message flow

```
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
```

Implementation details

Adding support for this to existing name server implementations is relatively easy, but does require source code modifications. It is also controlled via various kernel parameters documented below.

Client side code changes

Linux API

- On the client, the call to connect() is replaced with a call to sendmsg() or sendto() with the *flags* parameter set to *MSG_FASTOPEN*.
    - For blocking sockets this performs both the handshake and sends the data before returning.
    - For non-blocking socket it returns the number of bytes buffered and sent in the SYN packet.
        - If the cookie is available it will send the data before returning.
        - If the cookie is not available locally, it returns -1 with errno EINPROGRESS, and sends a SYN with TFO cookie request automatically. The caller needs to write the data again when the socket is connected with a call to send() or similar.
- Subsequent writes on this socket must also be done with send() or similar. A subsequent call with sendto() will return the error EISCONN.

OS X API

- The normal connect() call is replaced by connectx(), which requires 2 OS X specific flags:

```
connectx(fd, &endpoints, SAE_ASSOCID_ANY,
             CONNECT_DATA_IDEMPOTENT | CONNECT_RESUME_ON_READ_WRITE,
              NULL, 0, NULL, NULL)
```

- The the usual socket functions (e.g. write()) can be used to send data. Initiating the TCP handshake is delayed until the first data is actually send.

The Linux implementation follows the pattern suggested in the appendix of the RFC for maximum backwards compatibility, however the OS X API offers some advantages because TFO can be used in conjunction with libraries that require a 'connected' file descriptor to be available (e.g. OpenSSL). The hope is the Linux API will be extended to offer both modes (no FreeBSD client implemented yet).

Server side code changes

- The server must simply set the TCP_FASTOPEN flag using setsockopt() on the listening socket. On linux/FreeBSD the *qlen* value passed in to the function limits the number of outstanding TFO requests as a simple defense against IP spoofing attacks (see RFC7413).
    - Note on OS X the socket MUST be listening already for this flag to be set, and the qlen MUST be 1 (the actual value is set via the *net.inet.tcp.fastopen_backlog* kernel parameter.
    - On linux this call can be done after bind() is called.

Kernel Parameters

Linux

- The kernel parameter *net.ipv4.tcp_fastopen* controls TFO and since 4.1 has been set to 1 by default. This enables client mode but not server mode. To act in pure server mode set the integer value to **2**. To enable both client and server mode, set it to **3**, for example:

```
sysctl -w net.ipv4.tcp_fastopen=2
```

OS X

- The analogous kernel parameter is *net.inet.tcp.fastopen* but is set to 3 by default. The fastopen backlog and fallback minimum can also be set via kernel parameter.
- The parameter *net.inet.tcp.clear_tfocache* can be used to reset the TFO back-off when problems are encountered (this can be helpful when testing).

FreeBSD

- 6 kernel parameters are available, TFO is controlled by *net.inet.tcp.fastopen.enabled* which is 0 by default and must be 1 to enable TFO.

On-the-wire

The implementations have slightly different behaviour on the wire. Observations from testing include:

Clients

- The current Linux client implementation (4.4 at the time of writing) does not currently support receiving data in the SYN-ACK (although it should to be compliant with the spec). But a patch for this has been submitted. This can cause interop problems because the server must re-transmit the data. OS X does support this (no FreeBSD client implemented yet).
- The back-off algorithms also appear different. For example, the OS X implementation will fallback to normal TCP for a long period of time if it detects problems during cookie or TFO data exchange.
- Prior to 4.1 Linux used the experimental option code (254) and format for TFO, in 4.1 the default is to use the official option code (34) and format but fallback to the experimental code is still supported.

Servers

- Neither the Linux nor OS X servers send data in the SYN-ACK. The Linux server will send data after the SYN-ACK, without waiting for the ACK. My testing shows the OS X server always waits until after the handshake is complete, but the Apple folks tell me it should behave as Linux....
- The FreeBSD server implementation does send data in the SYN-ACK if it is available quickly enough.