Improving Tor using a TCP-over-DTLS Tunnel

Joel Reardon
Google Zurich

Ian Goldberg
University of Waterloo

18th USENIX Security Symposium

August 12th, 2009
Tor: Internet anonymity tool
Tor Network
Tor: circuit construction
Tor: circuit construction
Tor: circuit construction
Tor: circuit construction
Privacy for usable, low-latency communication.
However it can be slow, and that discourages casual usage.
Where is the observed latency?
Tor’s Datapath
Output buffers do introduce some latency

Output Buffer Size and Latency

(a) Waiting Times over Time

(b) Waiting Times CDF

(c) Buffer Length over Time

(d) Buffer Length CDF
This occurs when the socket is unwritable
A brief outline of TCP

TCP is designed to reliably send streams of data using packets.
Congestion controls throttles sending to maximize throughput while avoiding packet drops.
Of what are TCP output buffers composed?

![Graph showing Socket Output Buffer Size and Unacknowledged Packets over time.

- **X** - Unwritable Socket
- **•** - Unacknowledged Packets
- **-** - Socket Output Buffer Size

Time (seconds)

Socket Output Buffer Size (KB)
TCP Congestion Control (C/C) is to blame.
If C/C is applied to a, then it is also applied to b.

This is suboptimal; TCP is designed to throttle individual connections based on whether they witness a packet drop—proportional to their traffic.
An example of cross-circuit interference
Experiment to observe interference by bulk senders

![Bar graph showing delay comparison between Circuit Delay and Circuit delay while peer under load. The y-axis represents delay in milliseconds (ms), and the x-axis represents the two different conditions. The y-axis range is from 0 to 1600 ms. The graph shows a significant delay increase when the peer is under load.]
Packet Dropping / Reordering

TCP Stream (over network)

Kernel TCP
Buffered / Waiting
Readable
We want to use a separate TCP connection for each circuit
Concerns for separate TCP connections

- Individual TCP streams leak precise information about the size and rate of data to an adversary.
- Tor already faces some scalability concerns regarding its clique topology.
- Some versions of Windows suffer when opening many TCP sockets already.
- Any modification must be backwards compatible with the existing Tor network.
Our novel proposal: a TCP-over-DTLS tunnel

- DTLS - a secure (cf. TLS) protocol for transporting datagrams (UDP sockets)
- TCP implementation in user-space is used to generate TCP/IP packets, which are sent over DTLS
- The other end injects the received packet into their user-level TCP stack, and reads from user-level sockets
How TCP-over-DTLS addresses our issues

<table>
<thead>
<tr>
<th></th>
<th>IP</th>
<th>TCP</th>
<th>TLS</th>
<th>Application Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>UDP</td>
<td>DTLS</td>
<td>TORTP</td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How TCP-over-DTLS addresses our issues

- UDP operates in an unconnected mode, so it accepts packets from any destination.
- Each node advertises a UDP socket that multiplexes data for all connections.
- The sender is used to demultiplex the proper connection that is used to decrypt the DTLS payload.
- Nodes that do not offer a UDP socket will use the existing transport, assuring backwards compatibility.
Tor Interface

Tor Processing → cell_pack → Streams

cell_unpack → recv() → Packets

UDP Socket → DTLS decrypt → rx() → User-level TCP

DTLS encrypt → tx() → TCP Buffers

User-level TCP

TCP Buffers

TCP Sockets

TCP Re/Transmit Thread
How TCP-over-DTLS addresses our issues

Diagram:
- UDP Stream (over network)
- Kernel UDP
  - RXed Packets
  - Buffered/Waiting
  - Readable
- User TCP
  - Buffered/Waiting
  - Readable

Legend:
- Black: Buffered/Waiting
- Gray: Readable
- White: Not read
- Cross: Error or Not applicable
Experimental results from our implementation
Circuit latency comparison

Delay (ms)

TCP-over-DTLS Tor
TCP Tor

Base Delay
Delay under Load
Future Work
### Improved Memory Management

**Cell Pool**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

- **empty list:** 2, 5, 9, 4
- **socket 1’s input buffer:** 0, 6, 7
- **socket 1’s output buffer:** 1, 3, 11, 8
- **cell_t data:** 10
Back-propagation of Congestion Window

TCP cwnd=15
actual cwnd=10

ack, cwnd=10
data

TCP cwnd=20
actual cwnd=10

ack, cwnd=10
data

TCP cwnd=10
actual cwnd=10
We determined that TCP congestion control introduces latency into Tor’s datapath.

We determined that multiplexing circuits over TCP results in the unfair application of congestion control.

We proposed TCP-over-DTLS: a solution to address this issue that also addresses scalability issues and is backwards compatible with the existing Tor network.

We implemented our proposal and showed it successfully addressed cross-circuit interference.