Template-Driven HTTP CONNECT Proxying for TCP

Ben Schwartz, Meta Platforms Inc.
HTTPBIS @ IETF 123

Template-driven TCP Transport Proxy (i.e. MASQUE for TCP)

Proxy is identified by a template:

```
https://proxy.example/tcp
{?target_host,target_port}
```

In HTTP/1.1:

```
GET /tcp?
    target_host=192.0.2.1&
    tcp_port=443 HTTP/1.1
Host: proxy.example:443
Connection: Upgrade
Upgrade: connect-tcp
capsule-protocol = ?1
```

In HTTP/2 & HTTP/3:

```
:method = CONNECT
:protocol = connect-tcp
:scheme = https
:authority = proxy.example:443
:path = /tcp?
        target_host=192.0.2.1&
        target_port=443
capsule-protocol = ?1
...
```

Changes since -07

- New connection closure mechanism for FIN vs. RST distinction
- New section on resource exhaustion attacks by malicious clients
- New advice to avoid HTTP/1.1 if at all possible
- Minor editorial changes

New connection closure rules

- Normal TCP data ↔ DATA capsule.
- FIN flag ↔ new FINAL_DATA capsule (with payload or empty).
- Clean TCP connection close → Graceful send stream close
- Receive TCP RST → SHOULD close send stream abruptly.
- Receive stream closes abruptly or without FINAL_DATA → SHOULD emit TCP RST

Results:

- **Got FINAL_DATA?** Incoming TCP data is **guaranteed** complete.
- **Got graceful receive stream close?** Outgoing TCP data SHOULD have been delivered.

Thanks to David Schinazi, Martin Thomson, Mike Bishop, Willy Tarreau, Kazuho Oku, Piotr Sikora, Lucas Pardue (#2949, #3000)

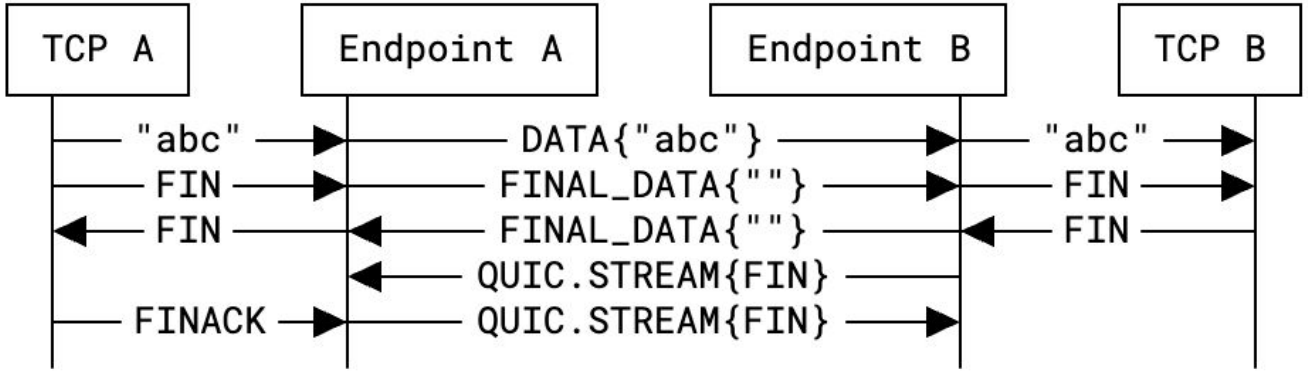

Figure 3: Simple graceful termination example (HTTP/3)

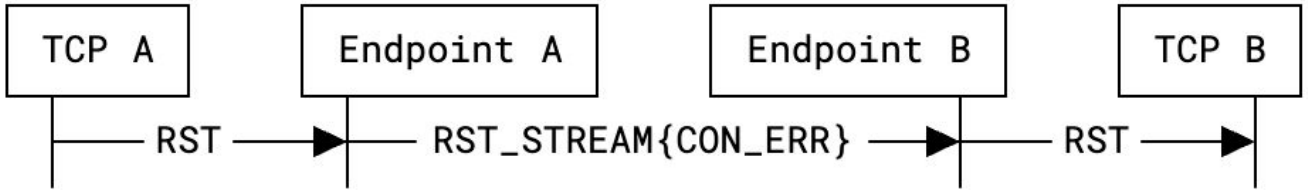

Figure 4: Simple TCP RST termination example (HTTP/2)

Resource Exhaustion Attacks

- **Malicious clients can overload your proxy server.**
  - More easily than you might think due to some asymmetric costs.
- Text describes several attacks and recommends three mitigations:
  - *“Limit the number of concurrent connections per client.”*
  - *“Limit the maximum receive window for TCP and HTTP connections, and the size of userspace buffers used for proxying. Alternatively, monitor the connections' send queues and limit the total buffered data per client.”*
  - *“Limit the number of connections for each client to each destination, even if those connections are in a waiting state and the corresponding CONNECT stream is closed. Alternatively, allocate a large range of IP addresses for TCP connections (especially in IPv6).”*

Avoid HTTP/1.1

“While this specification is fully functional under HTTP/1.1, performance-sensitive deployments SHOULD use HTTP/2 or HTTP/3 instead.”

- It wastes CPU, memory, time, ports, etc.
- For secure unclean shutdown you SHOULD inject a TLS alert, but most TLS libraries don’t let you do that.

Implementation Notes

Basic implementation: https://github.com/bemasc/masque-go/tree/capsules

Discoveries:

- Golang TLS doesn't let you send TLS alerts, but it does let you kill the connection without sending Finished. Should we recommend that instead for abrupt closure in HTTP/1.1?

WGLC!