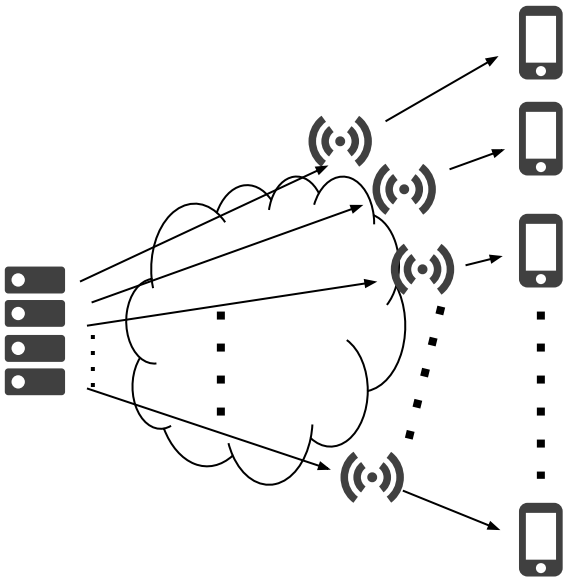
TCP Silent Close:
For Cases Where Silence is Golden

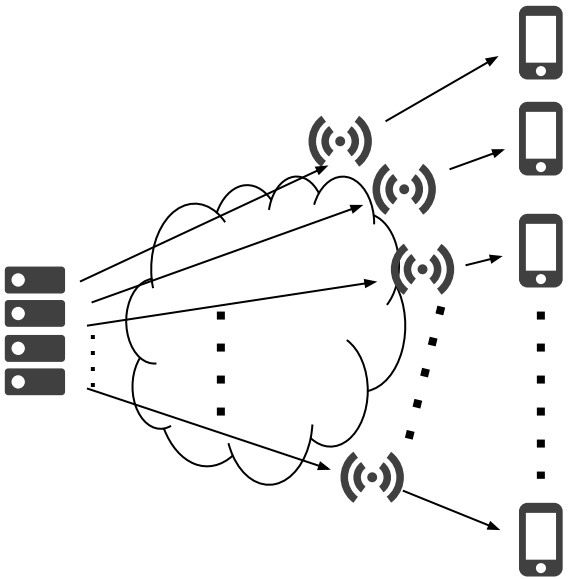
Neal Cardwell, Yuchung Cheng, Eric Dumazet, Luke Hsiao
{ncardwell, ycheng, edumazet, lukehsiao}@google.com
tcpm at IETF 112 / 2021-11-11

Motivating Scenario



- Large-scale Internet service with TCP connections to millions of cell phones
- Server machine's kernel closes all TCP connections quickly due to either:
  - Server application exit/restart for updating executable or configuration
  - Server application crash
- TCP bursts millions of FINs to cell phones in the local region

Problematic Impact



- Heavy network and energy resource usage
  - To find, contact, wake up millions of dormant phone radios
  - Network queuing and retries for phones with radios off or powered off
- Can cause mid-sized cellular networks to become overloaded and fail
  - A potential scalability/reliability vulnerability for some cellular networks

Is this Specific to TCP? No, but...

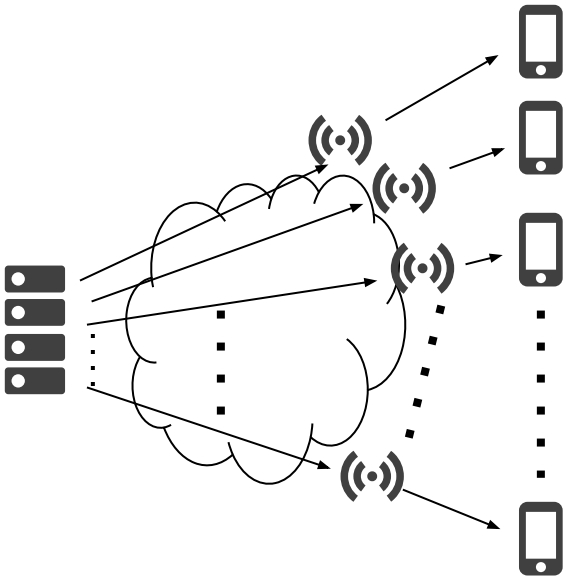
- TCP: this problem happens because responsibilities are split between application and kernel
  - If application crashes or exits, kernel tries to finish the FIN/ACK, FIN/ACK exchange
    - For potentially millions of connections in a burst!
- QUIC: this problem does not happen, since kernel has no transport state
  - If application crashes or exits, kernel takes no special networking action
    - No extra network load!
- In general, the problem presumably could happen with any kernel transport

Goal of the Silent Close Effort

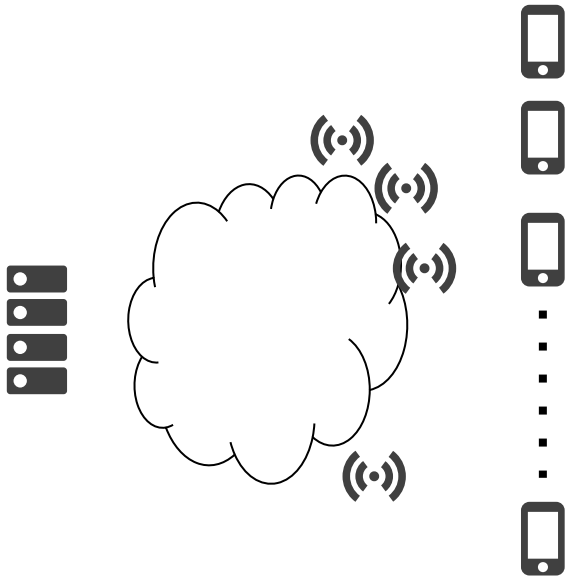
- Upon application exit or crash, instead of sending millions of FINs, manifest the traditional behavior for kernel or machine crash:
  - Connections go silent
  - Any later incoming traffic for those connections trigger RST
- Upon normal/healthy close of an individual server TCP connection (e.g., client closes connection, or client or server times out connection due to idleness):
  - Connection still uses normal FIN/ACK, FIN/ACK handshake
- Key design feature: no new network behaviors introduced
  - Apps and TCP stacks already have to be prepared for silence and then a RST
  - Before: could happen with kernel crash, machine power-off, link failure
  - After: could also happen with app crash or app exit

Comparing server process exit/crash...

Before: Traditional TCP



After: TCP_SILENT_CLOSE



TCP_SILENT_CLOSE Mechanism

- A boolean TCP_SILENT_CLOSE per-socket option
  - Set via setsockopt(); get via getsockopt()
- When enabled (TCP_SILENT_CLOSE=1), and TCP connection is closed or on shutdown(),
  - Kernel sends no FIN or RST
  - Kernel frees socket state immediately
- Effectively, the goal is that upon application exit or crash, the connection manifests the traditional behavior for kernel crash or machine power-off
  - Connection goes silent
  - Any later incoming traffic for that connection triggers RST
- We have used this option for years for particular, limited set of large-scale services
  - Plan to send upstream to Linux when net-next opens this month

Usage Model

- Enable TCP_SILENT_CLOSE on listener or immediately after connection is established
  - Thus a process crash/exit() does not result in large-scale bursts of FINs
- When a process wants to close a single TCP connection:
  - Disable TCP_SILENT_CLOSE before close() or shutdown()
  - Thus a normal close of a single connection does use normal FIN/ACK, FIN/ACK

Example API Usage

```
int one = 1;

if (setsockopt(fd, IPPROTO_TCP, TCP_SILENT_CLOSE, &one, sizeof(one)) < 0)

        perror("setsockopt TCP_SILENT_CLOSE");
```

API details

- Feature availability is controlled by a new sysctl, tcp_silent_close
  - Defaults to disabled; system administrator must explicitly enable
- Child server sockets inherit the value of the TCP_SILENT_CLOSE option from parent
  - Avoids requirement for a new system call per accepted child socket
- TCP_SILENT_CLOSE overrides SO_LINGER behavior
  - No point in lingering or sending RST if you don't want to send any packets
- close() always returns immediately if TCP_SILENT_CLOSE is enabled

Alternative considered for TCP_SILENT_CLOSE semantics:

- A close() or shutdown() system call always attempts FIN/ACK, FIN/ACK handshake
- Only process exit is silent and omits FIN/RST
- But this would prevent apps from programatically silent-closing millions of connections

TCP_SILENT_CLOSE Usage Considerations

- Before deploying, applications should consider negative impacts:
  - Extra memory use for state in TCP peer and/or middleboxes, e.g.:
    - NAT
    - Firewall
    - L4 load balancer using connection-tracking
  - Latency impact on client
    - Later client request will trigger RST, forcing re-connect and retry of request
- So TCP_SILENT_CLOSE should be used carefully, only in outage-prevention scenarios

Related Work

- Silent TCP Connection Closure for Cellular Networks [pdf], CoNext '13
  - Feng Qian, Subhabrata Sen, and Oliver Spatscheck. AT&T Labs.
  - Motivations:
    - Reduce energy consumption on mobile handsets
    - Reduce signaling load in cellular networks
  - Why not use this? Requires upgrading 4 pieces: {client, server} x {OS, application}.
- setsockopt(SO_LINGER) with timeout of 0
  - On close(), sends RST and frees socket
  - Motivation: avoids TIME_WAIT state scalability issues on busy servers
  - Why not use this? did not want to change the longstanding SO_LINGER API or semantics.
- "Silent close modification" in Linux TCP "repair mode" used for TCP connection migration: "Silent close modification: The close just aborts the connection (similar to SO_LINGER with 0 time) but without sending any FIN/RST-s to peer."
  - Why not use this? Semantics do not support silent close later upon process exit/crash.

TCP_SILENT_CLOSE Effort Status

- We have used this option for years for particular, limited set of large-scale services
  - Motivation: to reduce impact on cellular providers and mobile devices
- Plan to offer code upstream to Linux when net-next opens this month
- Inviting discussion of these problems and this proposed solution

Thanks!

Q&A