TFE listener architecture

Matt Klein, Staff Software Engineer
Twitter Front End

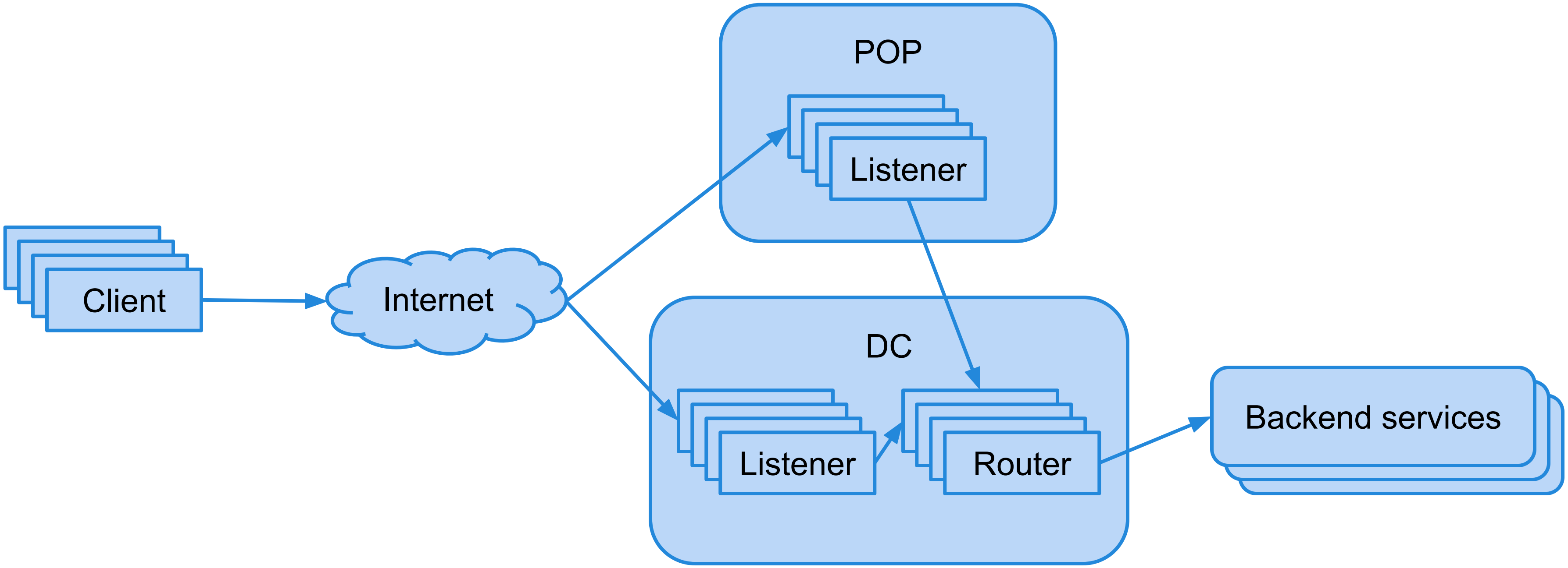
Agenda

- TFE architecture overview
- TSA architecture overview
- TSA hot restart
- Future plans
- Q&A

TFE architecture overview

- **Listener**: L7 reverse proxy. Terminates SSL, speaks HTTP 1.0/HTTP 1.1/SPDY 3.1/HTTP 2.0. Multiplexes requests coming from many connections onto a few high BW router links via SPDY 3.1
- **Router**: Accepts multiplexed requests from the listener. Interfaces with backend services via Finagle, server sets, etc. Authentication, rate limiting, geotagging. Complex path/request processing

TFE architecture overview



Multiple DCs omitted for simplicity

Multiple POPs omitted for simplicity

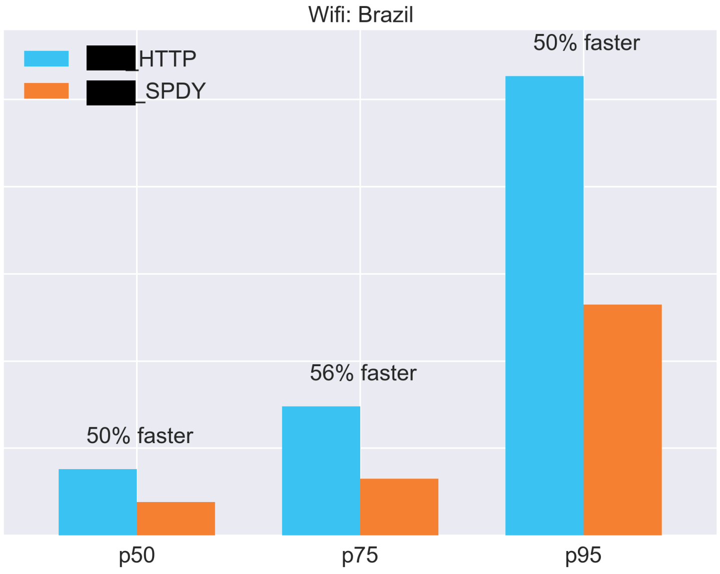
TFE architecture overview

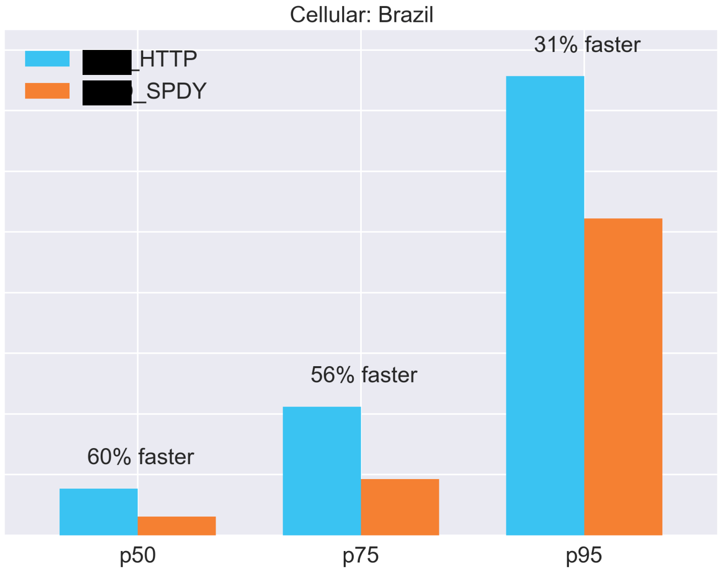
- Connection setup very expensive
  - SSL handshake
  - TCP window expansion (CWND)
- Bringing connection termination closer to the user yields faster RTTs, more reliable local links (less packet loss), and thus faster connection setup
- Put listeners in POPs and then backhaul requests over reliable high BW links
- Interesting design decisions related to routing/peering

TFE architecture overview

- First listeners installed in POPs for World Cup ‘14 in strategic locations
- Future POPs planned
- Target emerging markets with poor connectivity (e.g., India)
- Performance improvements via POPs impressive

TFE architecture overview





TFE architecture overview

- Performance improvements from POPs not just due to forward network locations
- With POPs came new listener, TSA-L
- Same core server that has been deployed as the streaming API reverse proxy since December ‘13
- C++, highly parallel, capable of “eternal” connections. Since connection setup is so expensive this ends up driving a lot of perf improvements

Agenda

- TFE architecture overview
- TSA architecture overview
- TSA hot restart
- Future plans
- Q&A

TSA architecture overview

- Design goals
  - Long lived connections
  - Reliable performance at high load
  - 100% streaming in both directions
  - Back pressure capable in both directions
  - Abstractions for protocols and applications that allow the same server to be used in multiple scenarios
  - Smaller HW footprint (more efficient), especially for POPs

TSA architecture overview

- C++, 100% asynchronous and non-blocking
- “Embarrassingly” parallel implementation means almost entirely lock free. Static thread count (1 per HW thread)
- “Hot restart” capability means the server can restart with zero downtime. Existing connections continue to be processed
- Custom stats implementation designed to work with hot restart in shared memory

TSA architecture overview

Streaming Server	One per listening port, multiplied by # HW threads (all cores listen on all ports)
User Session	Handles downstream connection/request(s) lifecycle. Connection affinitized to 1 thread
Codec	Encapsulates protocol (HTTP 1.0/HTTP 1.1/SPDY 3.1/HTTP 2.0)
Request	Encapsulates asynchronous multiplexed request streaming
Pipeline	Encapsulates application level proxying (e.g., TFE vs. streaming APIs)

TSA architecture overview

- Multiple layers of abstraction
    - **User Session** handles connection and request lifecycle (watchdogs, idle timeouts, etc.)
    - **Codec** wraps the underlying HTTP like protocol. Abstracts away events like receiving headers, body data, connection/stream window updates, etc.
    - **Pipeline** allows the server to handle multiple application level proxy scenarios based on the selected virtual host

TSA architecture overview

- TFE pipeline
  - Requests are RR between all DC routers over persistent SPDY/3.1 connections. TCP windows are always large in practice
  - Decider like failover possible between different router clusters
  - SSL mutual auth for POP security

TSA architecture overview

- Connection lifetime
  - “Legacy” TFE has a default idle timeout of 30s and a lifetime timeout of 45s
  - TSA launched with 15 minute idle timeouts and no lifetime timeouts (connections are “eternal”)
  - Future increases to idle timeout are a possibility
  - Enables new scenarios such as presence, active push, etc.

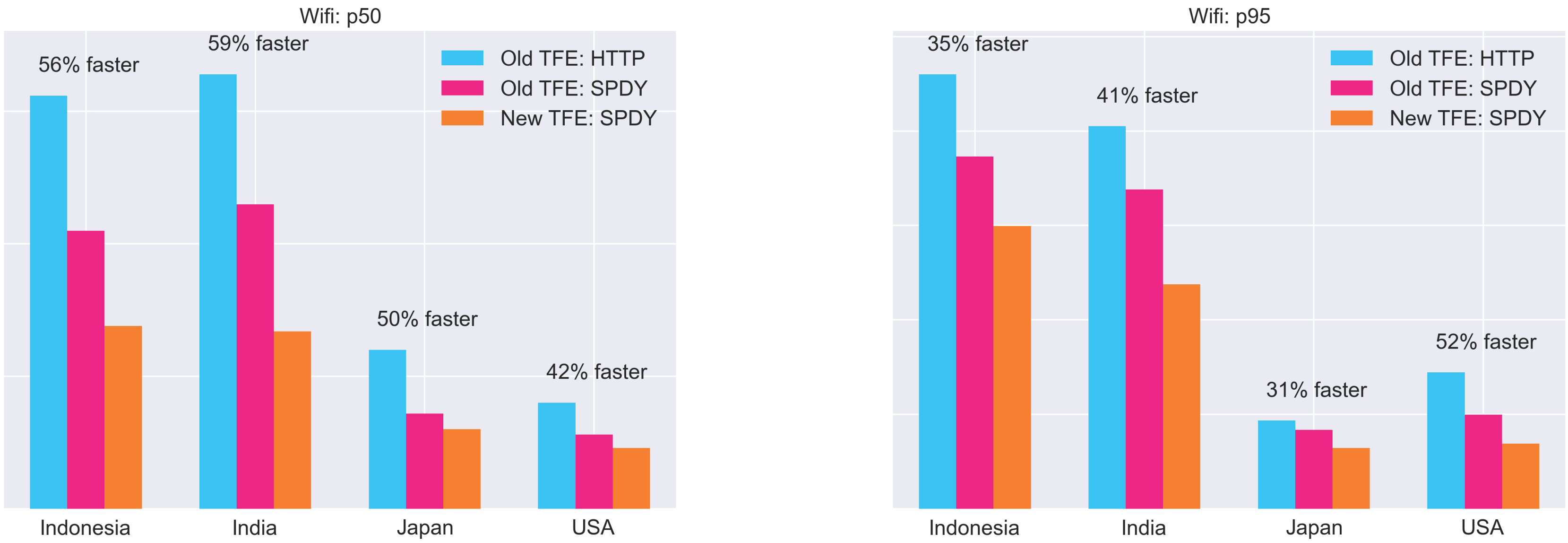
TSA architecture overview

- Performance
  - SNB/IVB 24 core server: 4K SSL CPS, 20K proxied RPS, >500K active connections, 80% load
  - CPU limited: SSL handshake for TFE, zlib for Hosebird
  - Memory: SPDY connections expensive due to how header compression is performed (deflate)
  - Most non-SSL/zlib CPU time spent processing HTTP headers. Room for optimization here

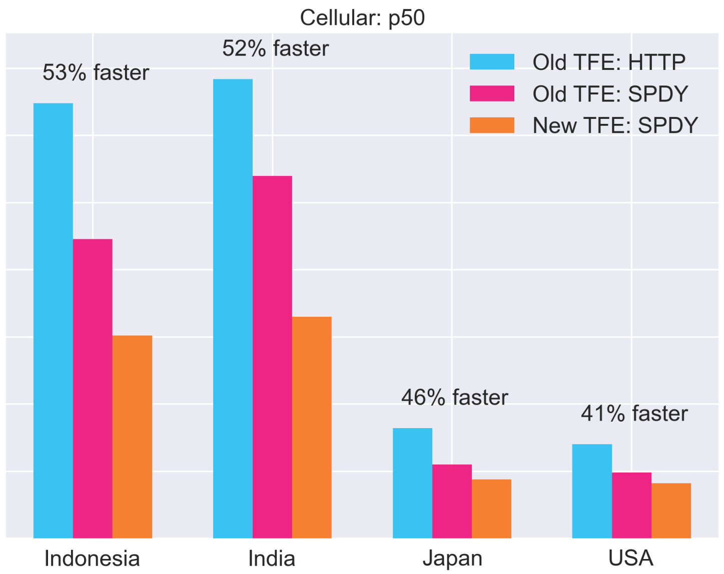
TSA architecture overview

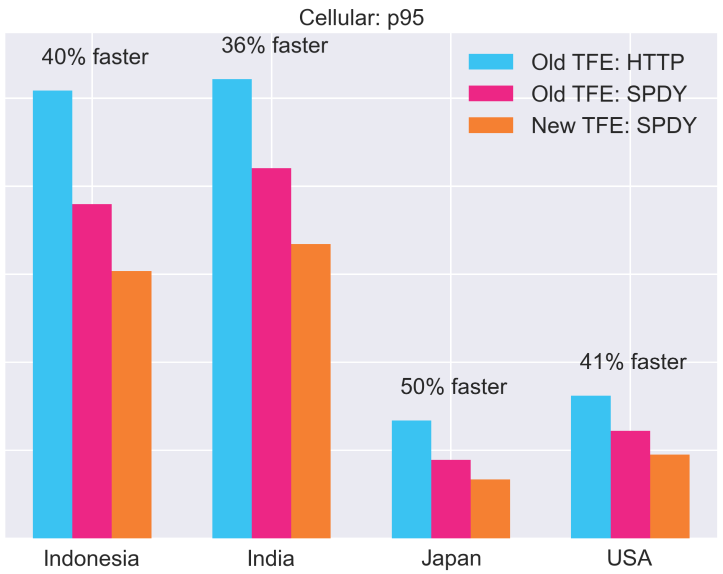
- Stability
  - We aim for zero crashes on production (non-canary) deployments. Track record very good
  - Even at very high load (80%+ CPU utilization) performance characteristics stable
  - Focus on sophisticated integration tests using fake clients, fake upstream servers, etc.

TSA architecture overview



TSA architecture overview





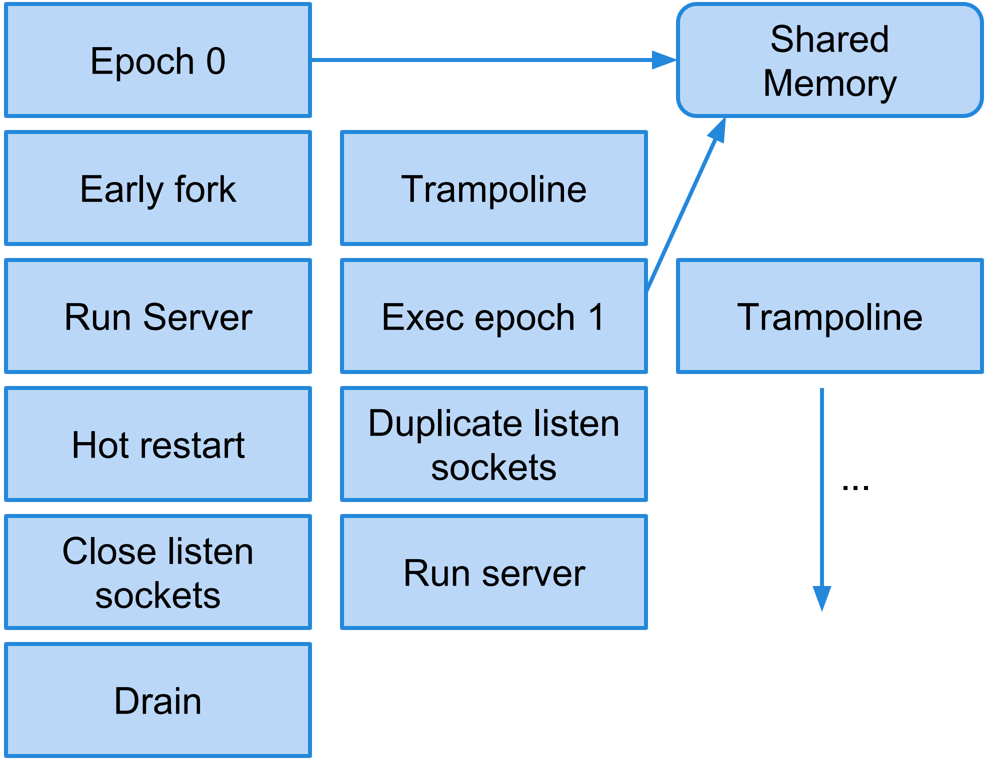
Agenda

- TFE architecture overview
- TSA architecture overview
- TSA hot restart
- Future plans
- Q&A

TSA hot restart

- With long lived connections, draining and restarting TSA is very time consuming
- We would like to be able to reload the server (code and configuration) without affecting existing connections
- If we program directly to the OS, some pretty cool stuff is possible and "hot restart" becomes a possibility
- Opens possibility of removing LBs in certain scenarios

TSA hot restart



- Stats and other shared control data kept in shared memory
- Forking restart trampoline early avoids complicated resource issues
- Unix domain sockets used for RPC and passing sockets
- "parent" process is controlled via new "primary" (admin, stats, etc.)
- N restarts possible. 2 processes allowed active at a time, the oldest process is terminated

TSA hot restart

- Forking restart trampoline early yields “clean” process to exec in with minimal state
- Unix domain sockets used for RPC / socket passing
- Shared memory stores stats, cross process log buffer flush lock, dynamic stat allocation lock, upstream health data, etc.
- Primary process responsible for health checking, admin, etc.
- In practice we drain old process slowly, but not required

Agenda

- TFE architecture overview
- TSA architecture overview
- TSA hot restart
- Future plans
- Q&A

Other features / future plans

- More POPs
- Policy based networking (allow developers to ask for specific connection QoS to mimic poor networking scenarios)
- Push
- CDN proxy
- Auth, limiting, geo, service discovery in TSA
- Previous enables direct proxy (router bypass) in certain scenarios

Future plans

- Further out:
  - Open source as a generic pluggable server
  - Factor out common libraries (admin, stats, hot restart, etc.) into Twitter C++ shared code
  - Use TSA as LB in certain deployments (direct connect to WAN)

Agenda

- TFE architecture overview
- TSA architecture overview
- TSA hot restart
- Future plans
- Q&A

Twitter Front End

- We work on a lot of really cool stuff
- SPDY/HTTP2 standards
- Mobile client network libraries (iOS/Android)
- L7 proxies for Twitter traffic
- L3/L4 software load balancing
- We are hiring systems programmers. Join us!

Q&A

- TSA is the result of the hard work of many teams and individuals too numerous to name here
- Thanks for coming!