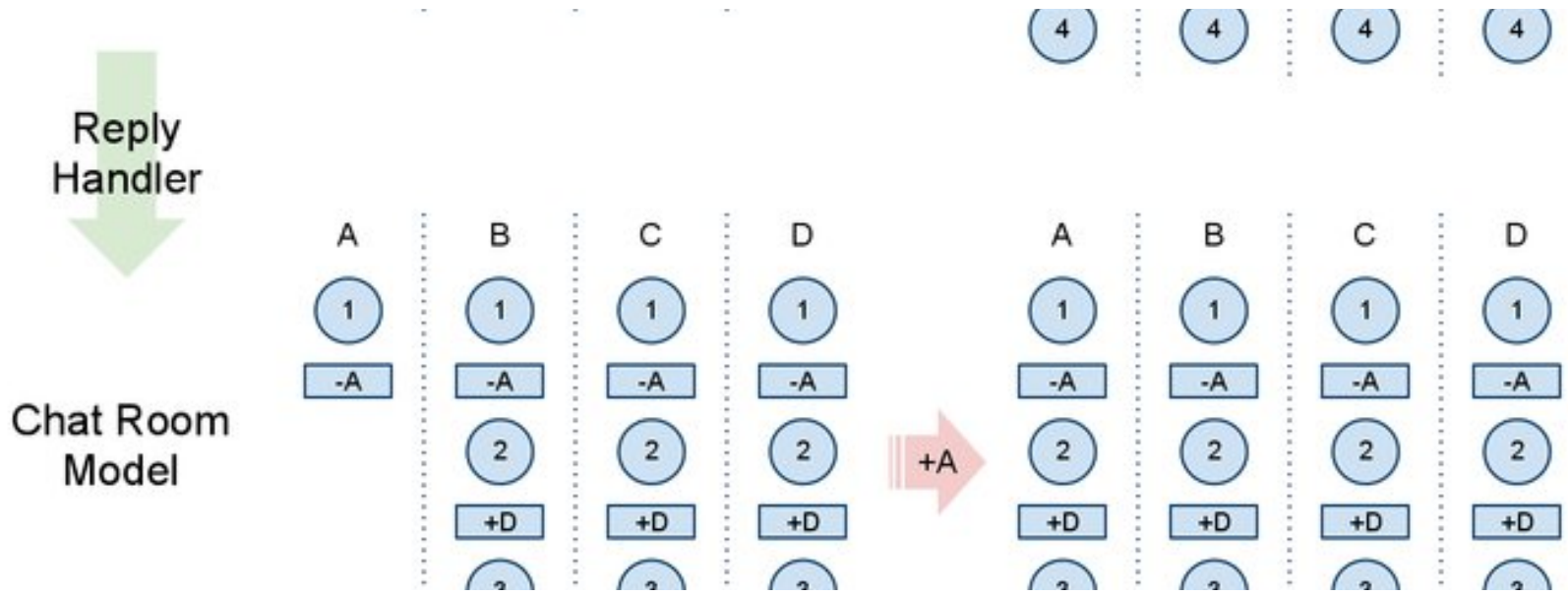
POSTED ON NOV 5, 2014 TO NETWORKING & TRAFFIC, PRODUCTION ENGINEERING

Introducing Proxygen, Facebook's C++ HTTP framework



By Daniel Sommermann Alan Frindell

We are excited to announce the release of Proxygen, a collection of C++ HTTP libraries, including an easy-to-use HTTP server. In addition to HTTP/1.1, Proxygen (rhymes with "oxygen") supports SPDY/3 and SPDY/3.1. We are also iterating and developing support for HTTP/2.

Proxygen is not designed to replace Apache or nginx — those projects focus on building extremely flexible HTTP servers written in C that offer good performance but almost overwhelming amounts of configurability. Instead, we focused on building a high performance C++ HTTP framework with sensible defaults that includes both server and client code and that's easy to integrate into existing applications. We want to help more people build and deploy high performance C++ HTTP services, and we believe that Proxygen is a great framework to do so. You can read the documentation and send pull requests on our Github site.

Background

Proxygen began as a project to write a customizable, high-performance HTTP(S) reverse-proxy load balancer nearly four years ago. We initially planned for Proxygen to be a software library for generating proxies, hence the name. But Proxygen has evolved considerably since the early days of the project. While there were a variety of software stacks that provided similar functionality to Proxygen at the time (Apache, nginx, HAProxy, Varnish, etc), we opted go in a different direction.

Why build our own HTTP stack?

- **Integration.**The ability to deeply integrate with existing Facebook infrastructure and tools was a driving factor. For instance, being able to administer our HTTP infrastructure with tools such as Thrift simplifies integration with existing systems. Being able to easily track and measure Proxygen with systems like ODS, our internal monitoring tool, makes rapid data correlation possible and reduces operational overhead. Building an in-house HTTP stack also meant that we could leverage improvements made to these Facebook technologies.
- **Code reuse.** We wanted a platform for building and delivering event-driven networking libraries to other projects. Currently, more than a dozen internal systems are built on top of the Proxygen core code, including parts of systems like Haystack, HHVM, our HTTP load balancers, and some of our mobile infrastructure. Proxygen provides a platform where we can develop support for a protocol like HTTP/2 and have it available anywhere that leverages Proxygen's core code.
- **Scale.** We tried to make some of the existing software stacks work, and some of them did for a long time. Operating a huge HTTP infrastructure built on top of lots of workarounds eventually stopped scaling well for us.

- **Features.** There were a number of features that didn't exist in other software (some still don't) that seemed quite difficult to integrate in existing projects but would be immediately useful for Facebook. Examples of features that we were interested in included SPDY, WebSockets, HTTP/1.1 (keep-alive), TLS false start, and Facebook-specific load-scheduling algorithms. Building an in-house HTTP stack gave us the flexibility to iterate quickly on these features.

Initially kicked off in 2011 by a few engineers who were passionate about seeing HTTP usage at Facebook evolve, Proxygen has been supported since then by a team of three to four people. Outside of the core development team, dozens of internal contributors have also added to the project. Since the project started, the major milestones have included:

- 2011 – Proxygen development began and started taking production traffic
- 2012 – Added SPDY/2 support and started initial public testing
- 2013 – SPDY/3 development/testing/rollout, SPDY/3.1 development started
- 2014 – Completed SPDY/3.1 rollout, began HTTP/2 development

There are a number of other milestones that we are excited about, but we think the code tells a better story than we can.

We now have a few years of experience managing a large Proxygen deployment. The library has been battle-tested with many, many trillions of HTTP(S) and SPDY requests. Now we've reached a point where we are ready to share this code more widely.

Architecture

The core HTTP layer is split into a four-part abstraction: the session, the codec, the transaction, and the handler. For each connection, a session is created. Each session has a codec, which is a protocol-specific framing object for serializing and de-serializing HTTP messages. The session is responsible for mapping each message from the codec to a particular HTTP transaction. Finally, the user of the library implements a handler that registers for HTTP-specific callbacks from a single transaction. This design has allowed us to support new multiplexing protocols like SPDY and HTTP/2 without having to duplicate code. You can see more detailed architecture diagrams in the project README.

Proxygen makes heavy use of the latest C++ features and depends on Thrift and Folly for its underlying network and data abstractions. We make use of move semantics to avoid extra copies for large objects like body buffers and header representations while avoiding typical pitfalls like memory leaks. Additionally, by using non-blocking IO and Linux's epoll under the hood, we are able to create a memory and CPU efficient server.

HTTP Server

The server framework we've included with the release is a great choice if you want to get set up quickly with a simple, fast-out-of-the-box event driven server. With just a few options, you are ready to go. Check out the echo server example to get a sense of what it is like to work with. For fun, we benchmarked the echo server on a 32 logical core Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz with 16 GiB of RAM,* *varying the number of worker threads from one to eight. We ran the client on the same machine to eliminate network effects, and achieved the following performance numbers.

```
# Load test client parameters:
# Twice as many worker threads as server
# 400 connections open simultaneously
# 100 requests per connection
# 60 second test
# Results reported are the average of 10 runs for each test
# simple GETs, 245 bytes of request headers, 600 byte response (in memory)
# SPDY/3.1 allows 10 concurrent requests per connection
```

While the echo server is quite simple compared to a real web server, this benchmark is an indicator of the parsing efficiency of binary protocols like SPDY and HTTP/2. The HTTP server framework included is easy to work with and gives good performance, but focuses more on usability than the absolute best possible performance. For instance, the filter model in the HTTP server allows you to easily compose common behaviors defined in small chunks of code that are easily unit testable. On the other hand, the allocations associated with these filters may not be ideal for the highest-performance applications.

Impact

Proxygen has allowed us to rapidly build out new features, get them into production, and see the results very quickly. As an example, we were interested in evaluating the compression format HPACK, however we didn't have any HTTP/2 clients deployed yet and the HTTP/2 spec itself was still under heavy development. Proxygen allowed us to implement HPACK, use it on top of SPDY, and deploy that to mobile clients and our servers simultaneously. The ability to rapidly experiment with a real HPACK deployment enabled us to quickly understand performance and data efficiency characteristics in a production environment. As another example, the internal configuration systems at Facebook continue to evolve and get better. As these systems are built out, Proxygen is able to quickly take advantage of them as soon as they're available. Proxygen has made a significant positive impact on our ability to rapidly iterate with our HTTP infrastructure.

Open Source

The Proxygen codebase is under active development and will continue to evolve. If you are passionate about HTTP, high performance networking code, and modern C++, we would be excited to work with you! Please send us pull requests on GitHub.

We're committed to open source and are always looking for new opportunities to share our learnings and software. The traffic team has now open sourced Thrift as well as Proxygen, two important components of the network software infrastructure at Facebook. We hope that these software components will be building blocks for other systems that can be open sourced in the future.

Thanks to all the engineers who have contributed to this project, including Ajit Banerjee, David Gadling, Claudiu Gheorghe, Rajat Goel, Peter Griess, Martin Lau, Adam Lazur, Noam Lerner, Xu Ning, Brian Pane, Praveen Kumar Ramakrishnan, Adam Simpkins, Viswanath Sivakumar, and Woo Xie.

TAGS: BACKEND FRAMEWORK INFRA LANGUAGE TOOLS OPEN SOURCE PERFORMANCE

Related Posts