

An HTTP/3 Module for the Apache HTTP Server Using OpenSSL QUIC and nghttp3

By: Tarek Ibrahim

4th Year Software Engineering Student

Carleton University

Department of Systems and Computer Engineering

Ottawa, Ontario, K1S 5B6

March 31, 2026

Abstract

OpenSSL 3.2 introduced native QUIC support, enabling HTTP/3 on top of the world's most widely deployed TLS library. Apache httpd, however, is still architected around TCP-based connections and its Multi-Processing Modules (MPMs), which makes integrating QUIC and HTTP/3 non-trivial. An example codebase (openssl-h3-examples) already demonstrates HTTP/3 using OpenSSL's QUIC support and nhttp3 in a standalone server, and reuses patterns inspired by mod_http2, but it remains experimental and is not integrated into httpd as a module. This project proposes a new HTTP/3 module for Apache httpd, building on OpenSSL QUIC and nhttp3, and aims to resolve the architectural mismatch between httpd's TCP-oriented worker model and QUIC's UDP-based, stream-multiplexed transport. The goal is a maintainable, reviewable prototype module that the Apache HTTP Server community can evaluate and evolve toward production readiness.

Abbreviations

Abbreviation	Definition
QUIC	Quick UDP Internet Connections (RFC 9000)
HTTP/3	Hypertext Transfer Protocol version 3
MPM	Multi-Processing Module
TLS	Transport Layer Security
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
SSL	Secure Sockets Layer
APR	Apache Portable Runtime
ALPN	Application-Layer Protocol Negotiation
HOLB	Head-of-Line Blocking
Alt-Svc	Alternative Service HTTP header
RFC	Request for Comments (IETF standard)
I/O	Input/Output
AEAD	Authenticated Encryption with Associated Data
IP	Internet Protocol
API	Application Programming Interface
GSoC	Google Summer of Code
PR	Pull Request

Table of Contents

Abstract	1
Abbreviations	2
1 Information	4
1.1 Project Information	4
1.2 Contact Information	4
2 Motivation	4
2.1 Why Apache	4
2.2 Why This Project	4
3 Community Engagement	5
3.1 Resources	5
3.2 Development Workflow	5
3.3 Planned Code Reviews	5
3.4 Results for the Apache community	5
4 Technical Plan	5
4.1 UDP and TCP	5
4.2 HTTP/2 and HTTP/3	6
4.3 HTTP/3 Module	6
4.4 Design and Architecture	6
4.5 Implementation Strategy	7
4.6 Testing and Validation Approach	7
5 Milestones and Deliverables	8
5.1 Phase 1: Analysis	8
5.2 Phase 2: Implementation	8
5.3 Phase 3: Verification and Validation	8
6 Timeline	8
6.1 May: Analysis (Discovery and Setup)	8
6.2 June: Implementation Begins	8
6.3 July: Implementation Continues	8
6.4 August: Verification and Validation (Wrap-Up)	9
7 About Me	9
7.1 Background	9
7.2 Open Source Contributions	9
7.3 Internships	10
7.4 Academia	10
7.5 Personal Statement	11
8 AI Disclosure	11
References	11

1 Information

1.1 Project Information

Field	Details
Google Summer of Code	2026
Organization	Apache Software Foundation
Project Size	Large (350 hours)
Project Difficulty	Hard
Mentor(s)	Jean-Frédéric

1.2 Contact Information

Field	Details
Name	Tarek Ibrahim
University Email	tarekibrahim3@cmail.carleton.ca
Personal Email	soccerspace04@gmail.com
GitHub	https://github.com/machine-moon
LinkedIn	https://www.linkedin.com/in/im-tarek/
Portfolio	https://t-ibrahimm.github.io/
Timezone	UTC-4 (Ottawa, ON)
Known absences or commitment conflicts	No conflicts expected; I manage my commitments

2 Motivation

2.1 Why Apache

The Apache Software Foundation hosts some of the most widely used open source infrastructure projects in the world. Apache projects are known for their stability, thorough review, and real-world usefulness, not just as short-lived experiments. That makes contributions here feel meaningful and enduring. Apache HTTP Server alone powers nearly a quarter of the internet, this is exactly the kind of impactful, community-driven engineering I want to be part of.

2.2 Why This Project

HTTP/3 and QUIC represent the most significant change to the transport stack underlying the web since the introduction of TLS. QUIC offers real performance and resilience benefits that are increasingly demanded in production deployments. Apache httpd is the most widely deployed web server on the internet, and it already has a proven HTTP/2 implementation in `mod_http2` built on top of `libnghttp2`. Work done here does not stay as a toy experiment. If it is accepted, it can follow the same path as `mod_http2`: from an external module to a maintained, in-tree implementation and be used in real deployments and maintained by an active community. Having prior experience with safety-critical C++ networking systems and a strong grounding in networking protocols and systems design makes this project a natural and exciting fit.

3 Community Engagement

3.1 Resources

Resource	Link
Repository	svn.apache.org/repos/asf/httpd/httpd/trunk
Modules	httpd.apache.org/modules/
HTTPD Test Utils	httpd.apache.org/test/
Mailing Lists	dev@httpd.apache.org & docs@httpd.apache.org
Bugzilla	httpd.apache.org/bug_report.html
Confluence Wiki	cwiki.apache.org/confluence/display/httpd
Jira	issues.apache.org/jira/
CI/CD	github.com/apache/httpd/actions
Libera IRC Chat	#httpd & #httpd-dev
Guidelines	httpd.apache.org/dev/guidelines.html
Style Guide	httpd.apache.org/dev/styleguide.html

3.2 Development Workflow

The development workflow follows a simple cycle of analysis, implementation, and review, aligned with Apache httpd's existing practices. Development will follow the Apache HTTP Server Project's normal contribution process: design discussions and patch proposals on dev@httpd.apache.org, small and reviewable changes, and adherence to existing coding, documentation, and testing conventions. Where appropriate, work items and status will be tracked in the usual project STATUS mechanisms.

3.3 Planned Code Reviews

Code will be developed in small patches and submitted regularly for review, aiming for at least one reviewable change set every one to two weeks. The exact cadence and review process will follow the mentor's preferences and the httpd community's norms.

3.4 Results for the Apache community

The most valuable outcome is not just the code itself but a well-documented architecture that finally addresses the long-standing MPM and QUIC mismatch. This gives future contributors a solid, reviewed foundation to build on instead of starting from scratch. Delivering a working prototype that reaches the dev list and passes community review brings Apache httpd one step closer to full HTTP/3 support, helping it narrow the gap with nginx and Caddy. That is a meaningful step forward for one of the most widely used open source web servers in the world.

4 Technical Plan

4.1 UDP and TCP

Apache httpd's connection handling is built around its MPM layer, which binds to network ports, accepts connections, and dispatches them to worker threads or processes. This model, and all deployed HTTP versions up to and including HTTP/2, assume a TCP transport, where each connection is a reliable, ordered byte stream and work is driven by sockets becoming readable or writable.

QUIC operates over UDP, where the kernel does not maintain connection or stream state. A single UDP socket can receive packets for many independent QUIC connections, each multiplexing multiple streams. Connection state, loss detection, congestion control, and retransmission are implemented by the QUIC stack in user space rather than by the kernel's TCP implementation. The lifetime and scheduling model for QUIC streams is therefore fundamentally different from that of TCP connections, and this mismatch is the central architectural challenge for `mod_h3`.

4.2 HTTP/2 and HTTP/3

OpenSSL 3.2 provides a QUIC transport via its SSL API, and `nghttp3` implements HTTP/3 semantics on top of that transport, including header compression via QPACK. HTTP/3 over QUIC addresses the head-of-line blocking limitations of TCP-based HTTP: individual streams can make progress independently even when packets are reordered or lost, while still maintaining efficient header compression and high concurrency across many requests. The existing prototype in `openssl-h3-examples` combines OpenSSL QUIC and `nghttp3` in a simple single-threaded server that returns a fixed 20-byte ASCII response for any request, demonstrating that the QUIC + HTTP/3 stack is functional but not yet integrated with `httpd`'s connection, request, and worker model.

4.3 HTTP/3 Module

Apache `httpd` already provides HTTP/2 support through `mod_http2`, which uses `libnghttp2` as its engine and integrates it with the connection and request infrastructure. The existing `mod_h3` prototype in `openssl-h3-examples` intentionally reuses `mod_http2` patterns: it creates a “fake” `conn_rec` associated (for now) with an `nghttp3_conn`, constructs a `request_rec` in the `on_recv_header` callback, and relies on standard `httpd` hooks and filters to process the request, with connection and request pools freed by `httpd` once the HTTP/3 connection completes. The goal of this project is to extend that `mod_http2`-inspired design into a robust, QUIC-aware module that fits cleanly within `httpd`'s existing MPM and request-processing model.

4.4 Design and Architecture

The proposed architecture for `mod_h3` is organized into three cooperating layers that build on one another. At the base, the OpenSSL QUIC layer manages the UDP socket, handles encryption and decryption, and maintains per-connection transport state, including congestion control and loss recovery. Above it, the `nghttp3` layer implements the HTTP/3 framing and stream logic, translating between HTTP/3 frames and the application data passing through each QUIC stream. At the top, the Apache `httpd` integration layer maps `nghttp3` events into the standard `httpd` request and response lifecycle, reusing existing hooks and filters wherever possible to stay consistent with Apache's module design.

The key design decision is how to map QUIC connections and their streams onto Apache's worker model. One practical approach is to dedicate a listener-style thread to the UDP socket that routes incoming packets into per-connection QUIC state, then dispatches individual streams to worker threads for request handling. This preserves the concurrency of `httpd`'s event-driven model while ensuring that all packets belonging to a single QUIC connection are handled by a component that maintains its transport state correctly.

The exact scheduling mechanism and concurrency model will be refined during Phase 1 in consultation with the mentor, since this choice has important implications for complexity, performance, and testability.

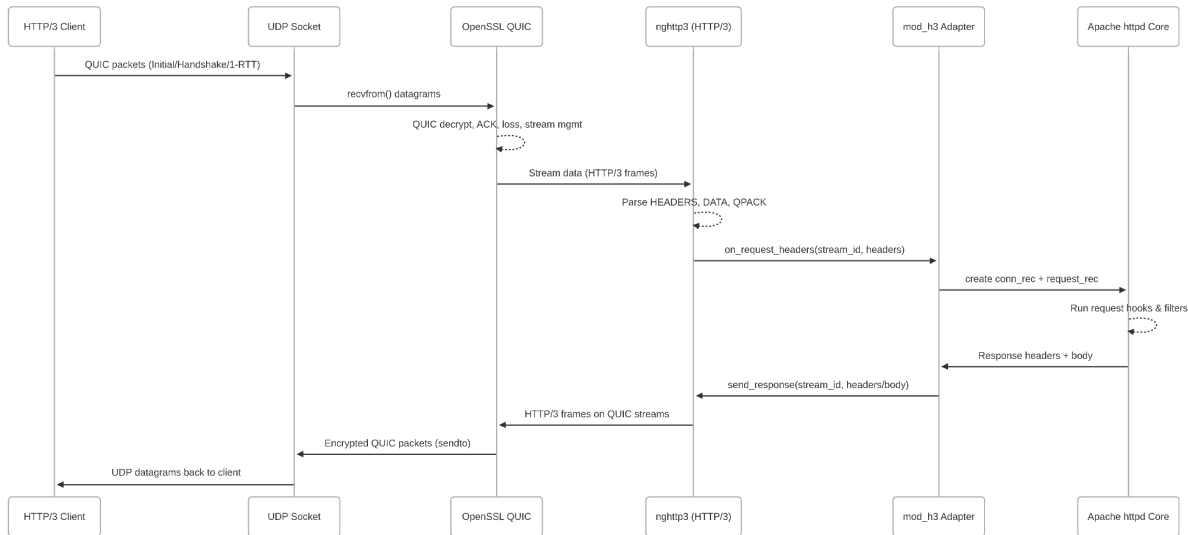


Figure 1: High-level architecture of the proposed HTTP/3 (mod_h3) module, showing how Apache httpd interacts with nhttp3 and OpenSSL QUIC over UDP.

4.5 Implementation Strategy

Implementation begins with a detailed review of the existing openssl h3 examples and mod_h3 code, along with the QUIC and HTTP/3 APIs provided by OpenSSL and nhttp3. The first goal is to bring the prototype to a point where it can reliably serve basic HTTP/3 responses in a controlled single process environment. This establishes a working integration baseline before tackling the more complex MPM integration.

The second phase focuses on integrating QUIC connection and stream handling with Apache's worker model. This milestone will produce a functional prototype meant to demonstrate a credible and reviewable architecture rather than a production grade feature. Configuration directives for enabling mod_h3, setting the UDP port, and tuning core parameters will follow Apache's standard style. The module will also generate Alt Svc headers so that clients can discover and negotiate HTTP/3 support.

Error handling, graceful shutdown, and worker exhaustion under heavy load are recognized challenges that will be analyzed and documented.

4.6 Testing and Validation Approach

Testing will focus on protocol correctness, integration, and robustness. Validation will include interoperability with HTTP/3 capable clients such as curl with QUIC support, Firefox, and public HTTP/3 test tools to verify negotiation, request handling, and response delivery. Apache's existing test framework will be reused or lightly extended to cover core mod_h3 request and response scenarios, while additional tests and simple stress cases (for example, concurrent connections and stream resets) will be run to assess stability. Interoperability will also be checked against publicly available HTTP/3 servers and clients, with any failures documented and refined based on mentor and httpd community feedback.

5 Milestones and Deliverables

5.1 Phase 1: Analysis

Phase 1 establishes an understanding of the openssl-h3-examples codebase, OpenSSL's QUIC API, nghttp3's design, and Apache httpd's MPM and module infrastructure. The primary deliverable is a short design document capturing: the proposed QUIC/httpd boundary, the chosen strategy for mapping QUIC connections to Apache workers, the initial set of configuration directives, and a clear statement of what is out of scope for this project. This document will be shared with the mentor for feedback before Phase 2 begins. A working build environment for httpd with OpenSSL QUIC and nghttp3 is also confirmed during this phase.

5.2 Phase 2: Implementation

Phase 2 implements `mod_h3` according to the Phase 1 design. The deliverables are: a functional `mod_h3` that can serve basic HTTP/3 responses to standard clients, integration with httpd's connection and module lifecycle hooks, initial configuration directives, and Alt-Svc header support. Developer documentation describing the internal architecture, build requirements, and known limitations is produced alongside the code.

5.3 Phase 3: Verification and Validation

Phase 3 refines the implementation based on testing and mentor feedback. Interoperability tests with multiple HTTP/3 clients are run and the results documented. Known limitations and future work items are written up clearly. A patch series or contribution formatted according to httpd community conventions is prepared, together with user-facing documentation describing how to enable and configure `mod_h3` in a test environment.

6 Timeline

6.1 May: Analysis (Discovery and Setup)

During May, development and test environments are configured: Apache httpd is built from source with OpenSSL 3.2+ and nghttp3, and the existing openssl-h3-examples code is studied in detail. The event MPM and module hook system are reviewed. The Phase 1 design document is drafted and reviewed with the mentor. By the end of May, the architecture is agreed and the build setup is confirmed.

6.2 June: Implementation Begins

June focuses on a working baseline: a `mod_h3` skeleton that integrates with httpd's module system, establishes a UDP socket for QUIC, and can serve at least one HTTP/3 response end-to-end using OpenSSL and nghttp3. Configuration directives are stubbed out. Early integration tests confirm that the basic path functions before the more complex worker-mapping work begins.

6.3 July: Implementation Continues

July addresses the harder integration work: mapping QUIC connections and stream events into Apache's worker model, and adding Alt-Svc header generation. Concurrent connection scenarios are exercised, and issues found during testing are addressed. Developer documentation is updated to reflect the actual design. A mid-summer review with the mentor assesses scope and identifies any necessary adjustments.

6.4 August: Verification and Validation (Wrap-Up)

August is reserved for validation, documentation polish, and patch preparation. Interoperability tests are run against multiple clients, results are documented, and the implementation is refined based on findings. A patch series and design note are prepared in a form suitable for httpd community review, together with a summary of current scope, known limitations, and recommended future work.

7 About Me

7.1 Background

I am a 4th year Software Engineering student at Carleton University in Canada's capital, with a strong interest in real-time systems, networking, and embedded software. My coursework includes Real-Time Embedded Systems, Computer Architecture, Operating Systems, and Computer Networks, which together provide a solid theoretical basis for RTOS design, concurrency, and low-level I/O. Professionally, I have developed safety-critical C++ microservices for air traffic control systems, and I am comfortable working under strict reliability and performance constraints. I also have experience with CUDA/HIP, CMake, and networking tools, and have dabbled with microcontroller platforms like the Raspberry Pi Pico and their SDKs.

7.2 Open Source Contributions

7.2.1 DreamerV3 - danijar/dreamerv3 PR #144 (Rejected)

My first open source contribution. I tried to apply single-layer bloating advice universally to the DreamerV3 world model, which was incorrect; the architecture requires different layer configurations for different components. The PR was rejected. What I learned was to read the full codebase before proposing changes. Understand why something is the way it is before trying to "improve" it. Contribution guidelines exist for a reason. This failure taught me more about open source collaboration than any success could have.

7.2.2 BitChat - bitchatz/bitchatz-cpp PR #1 (Merged)

My first meaningful open source impact. BitChat was a Bluetooth-based chat application that only worked on mobile. I ported it to desktop Linux by integrating BlueZ (the Linux Bluetooth stack), handling the platform-specific Bluetooth socket APIs, and getting it to compile and run on non-mobile targets. What I learned was that Platform abstraction is hard. Hardware interfaces (Bluetooth, in this case) have wildly different APIs across platforms. Build systems need to handle conditional compilation gracefully targeting multiple platforms with platform-specific optimizations.

7.2.3 CuMind - carletonai/cumind (Led)

Founded and led this project as VP Technology of the Carleton AI Society. CuMind is a modular JAX-based RL library with Gymnasium integration for environment management, configurable profiling, tracing, and checkpointing, a reimplementation of DeepMind's MuZero algorithm (proprietary to open source), and runs across CPU/GPU/TPU backends on Google Cloud.

I led weekly planning sessions, mentored 5+ contributors, reviewed PRs, and built the collaborative development environment. The MuZero reimplementation required reading the original paper, understanding the algorithm deeply, and implementing it from scratch. What I learned was that leading an open source project is different from contributing to one. You need clear architecture, good

documentation, and consistent review standards. You also need to scope ambition realistically; we started wanting to implement 10 algorithms and focused on doing MuZero really well instead.

7.3 Internships

7.3.1 Searidge Technologies (SRT)

At Searidge Technologies, experience centred on safety-critical C++ microservices for air traffic control, improving multithreaded components and performance in real-time environments while conforming to ED-109A software assurance standards. Work there also included building a multithreaded video-processing microservice and contributing to code reviews and architectural improvements.

7.3.2 Agriculture and Agri-Food Canada (AAFC)

At Agriculture and Agri-Food Canada, work focused on deploying bioinformatics pipelines on Canada's largest supercomputers and engineering infrastructure for large-scale LLM inference on NVIDIA A100 GPUs using containerization and HPC schedulers. This role also involved writing and refining technical documentation for CUDA, SLURM, and related HPC workflows, and supporting researchers using the supercomputing environment.

7.3.3 James Evan & Associates (JEA)

At James Evan & Associates, the emphasis was on modernizing a large TypeScript codebase for a production web application, resolving dependency and CI issues, and strengthening automated testing. This experience reinforced good practices around regression safety, CI integration, and long-term maintainability of user-facing software.

7.4 Academia

7.4.1 Coursework

Coursework in Algorithms and Data Structures, Advanced C++ Programming, Computer Architecture, Operating Systems, Computer Networks, and Real-Time Embedded Systems has built a strong foundation in systems programming, concurrency, and performance-oriented design. Additional courses in Software Design, Software Requirements, and Database Management supplied experience with software engineering methodology and data-centric applications, complementing the systems focus.

7.4.2 Teaching

Teaching assistant roles in Digital Systems, Foundations of Programming, and Computer Architecture involved mentoring and evaluating large cohorts of students in laboratory and tutorial settings. Responsibilities included explaining topics such as circuit design, C programming and memory management, and basic computer organization, as well as providing detailed feedback on assignments and lab work. These roles reinforced core material and developed skills in technical communication and code review.

7.4.3 Research and Laboratory Work

Research activities include work as an undergraduate ML researcher, co-authoring a NeurIPS 2025 paper on efficient visual recognition from self-supervision and deploying large-scale Vision Transformer training workloads on TPU pods. Additional experience comes from volunteering in the ARSLab simulation laboratory, contributing tests to the Cadmium v2 simulation engine and gaining familiarity

with discrete-event simulation tooling and research workflows. Together, these experiences reflect a trajectory oriented toward advanced study and research-grade software development.

7.5 Personal Statement

I got my first computer at a very young age, used parts that could barely play games. I taught myself to code by breaking things and reading Stack Overflow. Since then, computers have been the constant in my life. Not just using them, but understanding them: from digital systems to operating systems to compilers to neural networks. Every layer of the stack fascinates me. What draws me to open source specifically is the idea of building something that outlasts you. I want to write software that other engineers use, extend, and build on top of, software that becomes infrastructure. That's why I started CuMind, why I contribute to other projects, and why I'm applying to GSoC.

8 AI Disclosure

GitHub Copilot and Perplexity AI were used during the research and exploration phase of this proposal: Copilot assisted with codebase navigation and code reading, while Perplexity AI was used to surface relevant documentation, RFCs, mailing list threads, and related literature. Proposal text, technical analysis, design decisions, and implementation plans in this proposal were written by me.

References

- [1] J. Iyengar and M. Thomson, Eds., "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, RFC 9000, May 2021. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc9000>
- [2] M. Bishop, Ed., "HTTP/3," Internet Engineering Task Force, RFC 9114, Jun. 2022. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc9114>
- [3] M. Thomson and S. Turner, "Using TLS to Secure QUIC," Internet Engineering Task Force, RFC 9001, May 2021. [Online]. Available: <https://www.rfc-editor.org/info/rfc9001>
- [4] ngtcp2 Project, "nghttp3: HTTP/3 Library Written in C," GitHub repository, 2018. [Online]. Available: <https://github.com/ngtcp2/nghttp3>
- [5] OpenSSL Project, "OpenSSL 3.2 Series Release Notes," OpenSSL Software Foundation, Nov. 2023. [Online]. Available: <https://openssl-library.org/news/openssl-3.2-notes/>
- [6] OpenSSL Project, "openssl-quic(7): OpenSSL QUIC API Manual," OpenSSL 3.2 Documentation, 2023. [Online]. Available: <https://docs.openssl.org/3.2/man7/openssl-quic/>
- [7] The Apache Software Foundation, "event: Apache HTTP Server Version 2.4 Multi-Processing Module," Apache httpd Documentation, 2024. [Online]. Available: <https://httpd.apache.org/docs/2.4/mod/event.html>
- [8] The Apache Software Foundation, "Multi-Processing Modules (MPMs): Apache HTTP Server Version 2.4," Apache httpd Documentation, 2024. [Online]. Available: <https://httpd.apache.org/docs/2.4/mpm.html>
- [9] J. F. Clere, "HTTP/3 Where Are We Now? State of the Art in Our Servers," ApacheCon North America 2024 (presentation slides), Sep. 2024. [Online]. Available: https://www.apachecon.com/acna2024/slides/01_HTTP_3_where_are_we_now.pdf
- [10] J. F. Clere, "Trying to Get a mod_h3 Working Using OpenSSL and nghttp3," Apache httpd Developers Mailing List, Feb. 2026. [Online]. Available: <https://lists.apache.org/thread/dspmtv3t9zp7y0xd47vp54jtsm6n4ckx>
- [11] Y. Chen, B. Jaeger, and J. Zirngibl, "RFC 9000 and its Siblings: An Overview of QUIC Standards," Chair of Network Architectures and Services, School of Computation, Information and Technology, Technical University of Munich, Seminar Report, doi: 10.2313/NET-2024-04-1_02, Apr. 2024. [Online]. Available: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2024-04-1/NET-2024-04-1_02.pdf