**Implementations Survey**

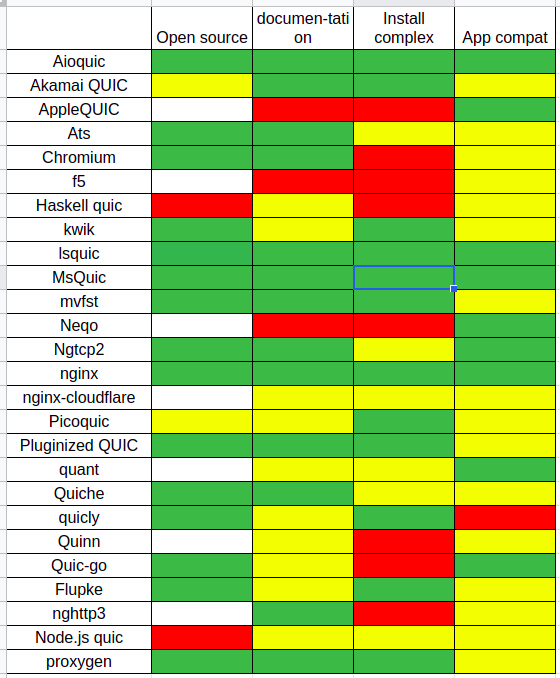


Table overlooking the implementations -   
<https://docs.google.com/spreadsheets/d/1F7fPKucheRRg9_yTh8xFDNG9CCPbf69jbfyka3TwtPw/edit?usp=sharing>

**Best-Match recommendations (Omer):**

1. Isquic
2. Quant

**Best-Match recommendations (Din):**

1. Nginx
2. Mvfst
3. Pquic
4. Optional - Chromium (depending on how hard is the installation)

**Best-Match recommendations (Dor):**

1. aioquic
2. MsQuic
3. Ngtcp2

**Extras -**

Table 1 - [Copy of IETF QUIC Interop Matrix](https://docs.google.com/spreadsheets/d/111amIanSl3qBvR6nZje_7JE-JSDJzqnPwN0QvNqf82E/edit?usp=sharing)  
Table 2- [Performance analysis of modern QUIC implementations](https://aaltodoc.aalto.fi/bitstream/handle/123456789/107677/master_Halme_Calle_2021.pdf?sequence=1&isAllowed=y)  
Table 3- <https://interop.seemann.io/>

1. **Aioquic**

**Open Source:** Yes (11 contributors)  
 - mostly python 3.6+ (3 required packages), cryptography - C  
 - had work done for the past month

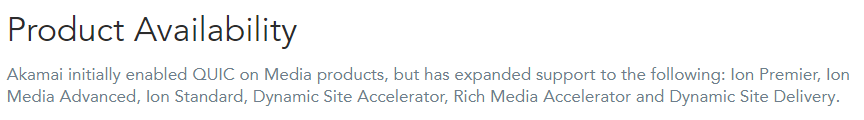
**Documentation level:** good  
 - In code, which seems very readable  
 - documented thoroughly in [GitHub](https://github.com/aiortc/aioquic)  
 - aioquic comes with a number of [examples](https://github.com/aiortc/aioquic/tree/main/examples) illustrating various QUIC use cases.

**Installation complexity:** seems feasible on both virtual and physical machines  
 requires Python and OpenSSL development headers  
 Windows, Linux, OS X  
 - advantage: logging quic events in QLOG format

**Applicative Compatibility (protocol’s version):**  
 - version 1 through draft 29  
 - QUIC stack conforming with RFC 9000  
 - HTTP/3 stack conforming with draft-ietf-quic-http-34

1. **Akamai Quic - Server Side**  
   Interesting information:  
   Successfully negotiated handshakes, streamed data, properly closed connections, resumed connections, and completed HTTP/3 transactions with Akamai’s server implementation with multiple clients (quant, picoQUIC, msquic, ATS, lsquic, aioquic, gQUIC, and Haskell QUIC).

Quic is already 8 years old and Akamai had it for most of that time.

Akamai was the first third party to deploy Google QUIC when it became available.  
  
Most of the implementation is written in Go.  
  


* 1. Open source - <https://github.com/akamai>  
     They provide useful tools to make it easy creating the environment and using it.

But I don’t think they have an open source, it looks like they implemented the server side and we can use it with their apps.

* 1. Level of documentation - There is documentation but to me it seems really short.
  2. Using Akamai CLI makes it easy to install on our machines (In case they are really open source).
  3. Protocol - draft-29.

1. **AppleQUIC**

a. קוד פתוח–

b. רמת התיעוד – אין.

c. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- קשה.

d. תאימות היישום –

AppleQUIC is a client and server implementation.

* **Language:** C, Objective-C, Swift
* **Version:** draft-29 and version 1
* **Roles:** Client, Server
* **Handshake:** TLS 1.3 RFC
* **Protocol IDs:** 0xff00001d + 0x00000001
* **Public server:** N/A

אין מידע ברשת חוץ מזה.

1. **ATS**

**Open Source:** Yes (243 contributors),  
 - Mostly C++, little C & Python  
 - seems like in constant work

**Documentation level:**

- low in code  
- detailed design, build, configuration and testing information in [confluence](https://cwiki.apache.org/confluence/display/TS/QUIC)  
- structure, reqs for different OS, building and installation described in [GitHub](https://github.com/apache/trafficserver)  
- examples (i.e. client request & server response include printing)

**Installation complexity:** seems feasible on both virtual and physical machines,  
 but a bit complex –  
 2 different sources for same documentation (GitHub & Confluence),  
 8+ packages (sometimes different) for different OS  
 Ubuntu, Linux, macOS  
 Weirdly no mentioning of specific requirements for Windows

**Applicative Compatibility (protocol’s version):** draft-29  
 from documentation it seems like-  
 data flow consistent with draft-05  
 client (“traffic\_quic”) consistent with draft-17  
 compatible with HTTP/1.1 & HTTP/2

1. **Chromium “quiche”- Server, client and library**  
   Interesting information:  
   It is Google's production-ready implementation of QUIC.  
     
   To embed quiche in our project platform APIs need to be implemented and build files need to be created. Chromium is the API open source which we can use in our project.
   1. Open source - there are currently two repositories:  
      <https://quiche.googlesource.com/quiche>  
        
      <https://github.com/google/quiche>
   2. Because the code is by Google's teams, every code file has documentation.
   3. Because Chromium is still not an official release by Google and only a temporary API we might have trouble implementing our own API.  
      The implementation of quiche on our server might take work to implement.
   4. Q043, Q046, Q050, T050, T051, draft-27, draft-29.
2. **F5**
3. Open Source –
4. רמת התיעוד – אין.
5. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- קשה.
6. תאימות היישום – draft-29 and 32.

QUIC implementation in F5 TMOS

* **Language:** C
* **Version:** draft-29 and 32
* **Roles:** Server, Client
* **Handshake:** RFC 8446
* **Protocol IDs:** 0xff00001d, 0xff000020
* **ALPN:** h3-29, h3-32. hq-29 and hq-32 available upon request.
* **Public server:** f5quic.com:4433. spin, logging, and retry on f5quic.com:4434.

1. **Haskell quic**

**Open Source:** Yes (2 contributors),  
 - 55% Haskell, 45% C  
 - untouched since June

**Documentation level:** - almost none in source code - almost none in [GitHub](https://github.com/kazu-yamamoto/quic)  
 - modules partition looks good and code files are short

**Installation complexity:**

Couldn’t find installation instructions

**Applicative Compatibility (protocol’s version):** draft-29  
 - HTTP/3 and QPACK implementation in different project,  
 for draft-ietf-quic-http and draft-ietf-quic-qpack accordingly

1. **Kwik - Client and client library**  
   Interesting information:  
   They are still working on the server side, we can use their server side in our implementation but it might not work.  
     
   Most of the information on Kwik is on their bitbucket: <https://bitbucket.org/pjtr/kwik/src/master/>  
     
   This implementation is mostly written in Java.  
     
   HTTP/3 is supported with an add-on called “Flupke”, here is their bitbucket: <https://bitbucket.org/pjtr/flupke/src/master>
   1. Open source - the code is available on their bitbucket link.
   2. There is documentation but to me it seems really short.
   3. Installing on our machines shouldn’t be hard, there’s instructions on how to install on their bitbucket site.
   4. draft-29, draft-30, draft-31, draft-32.
2. **Isquic**
3. Open Source – כן.
4. רמת התיעוד – רמה גבוהה של תיעוד, הדרכות וכו'. מצורף לינק לתיעוד: https://lsquic.readthedocs.io/en/latest/

c. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- יש הסבר טוב איך להתקין את המימוש.

d. תאימות היישום - – v1, Draft-34, Draft-29, Draft-27, Q043, Q046, and Q050.

LSQUIC compiles and runs on Linux, Windows, FreeBSD, Mac OS, and Android. It has been tested on i386, x86\_64, and ARM (Raspberry Pi and Android)

.

LightSpeed QUIC and HTTP/3 library. Works on Linux, FreeBSD, MacOS, Android, and Windows. Turn-key open-source web server that uses lsquic is available at [openlitespeed.org](https://openlitespeed.org/) in both source and package forms. Bindings are available for [Crystal](https://github.com/iv-org/lsquic.cr) and [Lisp](https://github.com/AeroNotix/cl-lsquic).

* **Language:** C
* **Version:** v1, Draft-34, Draft-29, Draft-27, Q043, Q046, and Q050.
* **Roles:** Client, Server, Library
* **Handshake:** QUIC Crypto, RFC 8446
* **Protocol IDs:** 0x00000001, 0xFF000022, 0xFF00001D, 0xFF00001B
* **Public server:**
  + http3-test.litespeedtech.com:4433, http3-test.litespeedtech.com:4434 (sends stateless retry packets), and http3-test.litespeedtech.com:4435 (faster downloads due to less logging), and :4437 (siduck-00 ) for v1, ID-34, ID-29, and ID-27 as well as Google QUIC versions Q043, Q046, and Q050
    - This server supports HTTP/3 and QPACK and provides some services to test transfer of data each way. GET / for details.
    - Delayed ACKs are supported -- versions -01 and -02.
  + [www.litespeedtech.com:443](http://www.litespeedtech.com:443/) is our production web server.

1. **MsQuic**

**Open Source:** Yes (28 contributors)  
 - Mostly C, little C++ & C#  
 - seems like it is in constant work

**Documentation level:** very good

**-** documented thoroughly in [GitHub](https://github.com/microsoft/msquic)

**Installation complexity:  
 -** Seems feasible on both virtual and physical machines, - Supported OS: Windows, Linux, macOS (alpha)

**Applicative Compatibility (protocol’s version):** version 1  
 draft-29

1. [**Mvfst**](https://github.com/facebookincubator/mvfst) **- Server, client and library**  
   Interesting information:  
   It is being developed by Facebook.  
     
   This implementation mostly uses two libraries [folly](https://www.github.com/facebook/folly) and [fizz](https://www.github.com/facebookincubator/fizz).  
     
   To support HTTP protocols we will also need to install [Proxygen](https://github.com/facebook/proxygen).  
     
   This implementation is in C++.
   1. Open source - <https://github.com/facebookincubator/mvfst>
   2. There is documentation in the code.
   3. The only problem I see in installing on our machines is that in their instructions on how to build we need an environment running ubuntu, Proxygen on the other hand should work on other OSs.
   4. draft-29.
2. **Neqo**
3. Open Source –
4. רמת התיעוד - התיעוד המסופק הוא קצר למדי, עם מעט מידע על פרטי הפרוטוקול.
5. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- קשה.
6. תאימות היישום – draft-27 through version 1.

Mozilla/Firefox QUIC and HTTP3 implementation.

* **Language:** Rust
* **Version:** draft-27 through version 1.
* **Roles:** library, client, server (server is primarily for client testing)
* **Handshake:** TLS 1.3
* **Protocol IDs:** 0x1, h3
* **Public server:** None

1. **NGTCP2**

**Open Source:** Yes (26 contributors), C  
 - seems like it is in constant work

**Documentation level:**

**-** documented thoroughly in [GitHub](https://github.com/ngtcp2/ngtcp2)  
 (requirements, build, client/server, configuring Wireshark, etc.)

**Installation complexity:**

Seems feasible on both virtual and physical machines

**Applicative Compatibility (protocol’s version):** version 1  
 draft-29  
 draft-30  
 draft-31  
 draft-32

1. [**Nginx**](https://hg.nginx.org/nginx-quic/) **- Server**  
   Interesting information:  
   They participated in [**QUIC interop testing**](https://interop.seemann.io/) so that their implementation is continuously tested with multiple clients, and in comparison with other server implementations.

Their implementation is written mostly in C.

* 1. Open source - <https://hg.nginx.org/nginx-quic>
  2. Their code has documentation.
  3. simply point your browser at [**https://quic.nginx.org/**](https://quic.nginx.org/) and check the [README](https://quic.nginx.org/readme.html) to learn more.
  4. Draft 23 - 29 and 34 (published standard).

1. **Nginx-Cloudfire**
2. Open Source –
3. רמת התיעוד – יש קצת.
4. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- יש הסבר איך להתקין.
5. תאימות היישום – draft-27, draft-28, draft-29.

Implementation of QUIC for NGINX based on quiche, by Cloudflare.

* **Language:** C
* **Version:** draft-27, draft-28, draft-29
* **Roles:** server
* **Handshake:** TLSv1.3 (RFC8446)
* **Protocol IDs:** 0xff00001b, 0xff00001c, 0xff00001d
* **Public server:** cloudflare-quic.com:443 (HTTP/3 only)

1. **PICOQUIC**

**Open Source:** Yes (27 contributers)  
 - Mostly C, little C++  
 - had work done for the past month  
 - Seems like it’s mainly for testing   
 minimalist implementation of test client and test server  
 - Secondary goal of experimenting with API for non-HTTP develeopment  
 i.e. DNS over QUIC  
 - Their goal was to have the “version 1” ready as soon as the RFC were  
 published. doesn’t seem ready

**Documentation level:** good  
 - Code is documented  
 - Building and installation instructions are documented  
 - 1st documentation recently just published

**Installation complexity:**

**-** Seems feasable on both virtual and physical machines, - Supported OS: Windows, Linux, macOS

**Applicative Compatibility (protocol’s version):** draft-27  
 draft-28  
 draft-29  
 draft-30  
 draft-31  
 draft-32

1. **Pluginized quic “pquic” - Server, client and library**  
   Interesting information:  
   This implementation enables QUIC clients and servers to dynamically exchange protocol plugins that extend the protocol on a per-connection basis.  
     
   This implementation is written mostly in C.
   1. Open source - <https://github.com/p-quic/pquic>.
   2. Their code has documentation.
   3. It can be built in Linux but the support of Windows is not provided yet.  
      If we are building on Linux then there are instructions in: [https://pquic.org](https://pquic.org/)
   4. Draft-29.
2. **Quant**
3. Open Source –
4. רמת התיעוד – יש קצת.
5. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- יש הסבר איך להתקין.

d. תאימות היישום – draft-33, draft-34, v1.

QUANT (QUIC Userspace Accelerated Network Transfers), a BSD-licensed C11 implementation on top of the zero-copy [warpcore](https://github.com/NTAP/warpcore) userspace UDP/IPv4 stack for the [netmap](http://info.iet.unipi.it/~luigi/netmap/) packet I/O framework. (Also works over the standard Sockets API.)

QUANT is a general transport library and does *NOT* implement H3.

* **Language:** C
* **Version:** draft-33, draft-34, v1
* **Roles:** client, library, server
* **Handshake:** TLS1.3
* **Protocol IDs:** 0xff000021, 0xff000022, 0x1
* **Public server:** quant.eggert.org:4433 (and more, see [wiki](https://github.com/NTAP/quant/wiki) for description)

1. **QUICHE (not the Google implementation)**

**Open Source:** Yes (43 contributors), Rust 1.50+  
 - had work done for the past month  
 - exposes thin C API – can be used to integrate into C/C++ applications

**Documentation level:** very good **-** documented thoroughly in [GitHub](https://github.com/cloudflare/quiche) – Building and usage examples

**Installation complexity:**

- Documentation only on building for Android, iOS  
- Seem feasible on both virtual machines using docker

**Applicative Compatibility (protocol’s version):** draft-27  
 draft-28  
 draft-29

1. **Quickly - Server and client**  
   Interesting information:  
   This implementation is made specifically to be used within the H2O HTTP server.  
     
   Written mostly in C.
   1. Open source - <https://github.com/h2o/quicly>.
   2. There is documentation but to me it seems really short.
   3. Building the software requires OpenSSL 1.0.2 or above, other than that they have instructions on how to build in their github webpage.
   4. Draft-27.
2. **Quinn**
3. Open Source –
4. רמת התיעוד – יש קצת.
5. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- יש הסבר איך להתקין, הוא לא כל כך טוב.

d. תאימות היישום – draft-28(או יותר נכון draft 32 לפי הגיטהאב שלהם).

Rust implementation based on tokio/futures, using rustls for TLS.

* **Language:** Rust
* **Version:** draft-28
* **Roles:** library, client, server
* **Handshake:** TLS 1.3
* **Protocol IDs:** 0xff00001c
* **Public server:** h3.stammw.eu:443

1. **QUIC-GO**

**Open Source:** Yes (45 contributors), Go

- had work done for the past month

**Documentation level:** medium

- almost none in [GitHub](https://github.com/lucas-clemente/quic-go)  
- little in code  
- minimal examples of server and client

**Installation complexity:**

Couldn’t find installation instructions

**Applicative Compatibility (protocol’s version):** version 1  
 draft-29

1. **Flupke - HTTP/3 client build**  
   Interesting information:  
   Using Flupke on top of kwik enables the use of HTTP/3 in kwik.  
     
   Flupke is a Java HTTP3 Client.
   1. Open source - <https://bitbucket.org/pjtr/flupke/src/master>.
   2. It seems there is very little documentation in the code itself.
   3. They have instructions on how to build in their bitbucket webpage.
   4. Draft-29.
2. **nghttp3**
3. Open Source –
4. רמת התיעוד – נרחבת.
5. רמת הקושי בהתקנה אצלנו במעבדה על מכונות פיזיות או וירטואליות- קשה, אמנם יש הסברים אך הם פחות מובנים מתיעוד של מימוש אחר שקראתי.
6. תאימות היישום – draft-32.

nghttp3 is an implementation of HTTP/3 mapping over QUIC and QPACK in C. It does not depend on any particular QUIC transport implementation.

* **Language:** C
* **Transport library:** It does not depend on any particular QUIC transport library.
* **HTTP over QUIC Version:** draft-32
* **Roles:** library
* **Public server:** nghttp2.org:4433

1. **Node.js QUIC**

**Open Source:** Yes\* (2733 contributors)  
 - Mainly JavaScript, little C++ and Python  
 - \* untouched for over a month, and Read-Only (Repository archived by owner)

**Documentation level:** low to none, or irrelevant to QUIC

**Installation complexity:**

**seems irrelevant**

**Applicative Compatibility (protocol’s version):** Draft-25

1. **Proxygen - Library and sample client and server**  
   Interesting information:  
   Proxygen implements HTTP/3 mapping over QUIC and QPACK, with MVFST as the transport.  
     
   Proxygen is written mostly in C++
   1. Open source - <https://github.com/facebook/proxygen>.
   2. There is documentation in the code.
   3. On their github page there are instructions on how to build.
   4. Draft-23 (but notice that it runs on top of mvfst which is draft-29).