Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: 31 October 2026

B. Coddington
Hammerspace
S. Mayhew
Red Hat
C. Lever, Ed.
Oracle

29 April 2026

Remote Procedure Call over QUIC Version 1
draft-cel-nfsv4-rpc-over-quicv1-04

Abstract

This document specifies a protocol for conveying Remote Procedure (RPC) messages via QUIC version 1 connections. It requires no revision to application RPC protocols or the RPC protocol itself.

Note

This note is to be removed before publishing as an RFC.

Discussion of this draft occurs on the NFSv4 working group mailing list (nfsv4@ietf.org), archived at https://mailarchive.ietf.org/arch/browse/nfsv4/. Working Group information is available at https://datatracker.ietf.org/wg/nfsv4/about/.

Submit suggestions and changes as pull requests at https://github.com/chucklever/i-d-rpc-over-quicv1. Instructions are on that page.

Status of This Memo

Table of Contents

1. Introduction

The QUIC version 1 protocol is a secure, reliable connection-oriented network transport described in [RFC9000]. Its features include integrated transport layer security, multiple independent streams over each connection, fast reconnecting, and advanced packet loss recovery and congestion avoidance mechanisms.

Open Network Computing Remote Procedure Call (often shortened to "RPC") is a Remote Procedure Call protocol that runs over a variety of network transports [RFC5531]. RPC implementations so far use UDP [RFC0768], TCP [RFC0793], or RDMA [RFC8166]. This document specifies how to transport RPC messages over QUIC version 1.

1.1. Motivation For a New RPC Transport

Viewed at a moderate distance, RPC over QUIC provides a similar feature set as RPC over TCP with TLS (as described in [RFC9289]). However, a closer look reveals some essential benefits of using QUIC transports:

* Even though the QUIC protocol utilizes the same set of encryption algorithms as TLSv1.3, the QUIC record protocol encrypts nearly the entire transport layer header and authenticates each IP packet. Advanced traffic analysis which was possible with TLS on TCP is no longer possible. QUIC protects against transport packet spoofing and downgrade attacks better than TLS on TCP.

* Because many real IP networks are oversubscribed, packet loss due to momentary link or switch saturation continues to be likely even on well-maintained data center-quality network fabrics.

  The QUIC protocol utilizes packet loss recovery and congestion avoidance features that are lacking in TCP. Because TCP protocol design has ossified, it is unlikely to gain these improvements. QUIC is more extensible than TCP, meaning future improvements in this area can be designed and deployed without application disruption.

* Further, because QUIC handles packet loss on a per-stream rather than a per-connection basis, spreading RPC traffic across multiple streams enables workloads to continue largely unperturbed while packet recovery proceeds.

* The QUIC protocol is designed to facilitate secure and automatic transit of firewalls. Firewall transparency is a foundational feature of NFSv4 (which is built on RPC).

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. RPC-over-QUIC Framework

RPC is first and foremost a message-passing protocol. This section covers the implementation details of exchanging RPC messages over QUIC. Readers should already be familiar with the fundamentals of ONC RPC [RFC5531].

RPC-over-QUIC relies on QUIC version 1 as the underlying transport [RFC9000]. The use of other QUIC transport versions with RPC MAY be defined by future specifications.

3.1. Establishing a Connection

When a network host wishes to send RPC requests to a remote service via QUICv1, it must first find an established QUICv1 connection, or establish a new one.

For the purpose of explanation, this document refers to the peer that initiates QUICv1 connection establishment as an "RPC client" peer. This document refers to the peer that passively accepts an incoming connection request as an "RPC server" peer.

QUICv1 connections are not completely defined by the classic 5-tuple (IP proto, source address, source port, destination address, and destination port). Each connection is also defined by its QUIC connection ID. For instance, if the IP address of either peer should change, or a NAT/PAT binding and the source UDP port changes, the receiver can still recognize an ingress QUICv1 packet as belonging to an established connection.

As a result, due to network conditions or administrative actions, an RPC-over-QUIC connection can be replaced (a reconnect event) or migrated (a failover event) without interrupting the operation of an upper layer protocol such as RPC-over-QUIC. A more complete discussion can be found in Section 9 of [RFC9000].

3.1.1. Connection Transport Parameters

When establishing a connection, peers exchange transport parameters, as described in Section 7.4 of [RFC9000].

3.1.1.1. Initial Flow Control Limits

These limits control the amount of data that each peer may send on a newly-created stream. The limits are used for flow control and cap the amount of memory needed by both peers to keep data flowing on the connection. The value of these limits are typically based on the bandwidth-delay of the physical link between the peers, and are not exposed to RPC applications.

3.1.1.2. Number of Streams Per Connection

Each QUICv1 peer can limit the number of streams per connection; see Section 4.6 of [RFC9000].

Given the definition of RPC message framing in Section 3.4, it is possible for an RPC client to create a stream, send one RPC Call, receive one RPC Reply, then destroy the stream. That usage might be common with simple RPC-based protocols like rpcbind.

For protocols that carry a more intensive workload, this style of stream allocation generates needless overhead. Moreover, stream identifiers cannot be re-used on a single QUICv1 connection, so eventually a QUICv1 connection can no longer create a new stream for each RPC XID. Finally, a connection peer can advertise a max_streams value that is significantly lower than 2 ^ 60.

Instead, RPC clients can create enough streams to maximize workload parallelism, and ought to avoid sending only a few RPCs on each stream before creating a new one.

For example, an RPC client could allocate a handful of streams per CPU core to reduce contention for the streams and their associated data structures. Or, an RPC client could create a set of streams whose count is the same as the number of slots in an NFSv4.1 session.

Even so, to provide a framework that makes implementing RPC-over-QUIC as fast and simple as possible, this specification needs to focus on enabling the use of as few as a single stream per connection.

Servers that implement RPC-over-QUIC need to recognize that each additional stream amounts to incremental overhead. RPC servers MAY deny the creation of new streams if an RPC client already has many active streams. RPC clients need to be prepared for that behavior.

3.1.1.3. Maximum Frame Size

This size is the largest QUIC frame that can appear in any stream on this connection. The QUIC framing protocol is not visible to the RPC application. The RPC client and server can therefore negotiate a frame size that enables efficient transit of RPC traffic with minimal internal memory fragmentation.

3.2. RPC Service Discovery

For RPC, the destination port is special. RPC services may use a standardized destination port that is bound to an RPC program number. Such ports are assigned in the IANA Service Name and Transport Protocol Port Number registry [IANA].

For example, the rpcbind program, which is RPC program 100000, listens on port 111. This is done so that RPC clients can always contact the rpcbind service and discover the other RPC services that are operating on that network peer.

In other cases, an RPC service might use any available port. The RPC server registers its port number with the local rpcbind service so that RPC clients can contact that service.

This mechanism is no different for RPC-over-QUIC than it is for RPC on other network transports. rpcbind clients specify an RPC program number and either the "quic" or "quic6" netid when requesting information about a QUIC-based RPC service. More detail is available in Section 7.1.

3.2.1. Transport Layer Security

During connection establishment, the client peer indicates RPC-over-QUIC support by presenting the ALPN token "sunrpc" in the TLS handshake. Support for other application-layer protocols MAY be offered in the same handshake.

As part of establishing a QUICv1 connection, the two connecting peers authenticate to each other and choose encryption parameters to establish a confidential channel of communication. All traffic on one QUICv1 connection is thus bound to the authenticated identities that were presented during connection establishment. These peer identities apply to the streams and RPC messages carried by that connection.

RPC-over-QUIC provides peer authentication and encryption services using a framework based on Transport Layer Security (TLS). Ergo, RPC-over-QUIC inherently fulfills many of the requirements of [RFC9289]. The details of QUIC's use of TLS are specified in [RFC9001]. In particular:

* With QUICv1, security at the transport layer is always enabled. Therefore, the discussion in [RFC9289] about the opportunistic use of TLS does not apply to RPC-over-QUIC, and the STARTTLS mechanism described in Section 4 of [RFC9289] MUST NOT be used on RPC-over-QUIC connections.

* The peer authentication requirements in Section 5.2 of [RFC9289] apply to RPC-over-QUIC.

* The PKIX Extended Key Usage values defined in [RFC9289] are valid for use with RPC-over-QUIC.

* The ALPN defined in Section 7.2 of [RFC9289] is also used for RPC-over-QUIC.

3.3. QUIC Streams

RPC-over-QUIC connections are mediated entirely by each peer's RPC layer and, aside from authentication and connection transport parameters, are not otherwise visible to RPC applications. An RPC client establishes an RPC-over-QUIC connection whenever there are application RPC transactions to be executed.

QUICv1 provides a "stream" abstraction, described in Section 2 of [RFC9000]. A QUICv1 connection carries one or more streams. Once a QUICv1 connection has been established, either connection peer may create a stream. Typically, the RPC client peer creates the first stream on a connection.

Unless explicitly specified, when RPC upper-layer protocol specifications refer to a "connection", for RPC-over-QUIC, this is a QUIC stream. For instance, when NFSv4.1 layered on RPC-over-QUIC uses BIND_CONN_TO_SESSION [RFC8881] to associate a "connection" with a session, the bound entity is a QUICv1 stream rather than the encompassing QUICv1 connection.

A peer that closes a QUICv1 stream signals that any RPC request still in flight on that stream is to be treated as lost. Closing a stream does not affect the encompassing QUICv1 connection or any other stream within it.

In terms of TI-RPC semantic labels, a QUICv1 stream behaves as a "tpi_cots_ord" transport: connection-oriented and in order.

> | cel: There is an opportunity here to add a stream that acts as a control plane.
> |
> | cel: Should we limit each stream to carry only one RPC program and version combination? Doing so would delegate demultiplexing of ingress RPC traffic to QUIC -- eg, NFSACL and NFS would be required to flow over separate streams.

3.4. RPC Message Framing

RPC-over-QUIC uses only bidirectional streams.

When a connection peer creates a QUICv1 stream, that peer’s stream endpoint is referred to as a "Requester", and MUST emit only RPC Call messages on that stream. The other endpoint is referred to as a "Responder", and MUST emit only RPC Reply messages on that stream. Receivers MUST silently discard RPC messages whose direction field does not match their Requester or Responder role. A receiver MAY also signal the violation to the peer by closing the offending stream with a RESET_STREAM frame (Section 19.4 of [RFC9000]) carrying an application error code; specific error code values and an extension process for defining additional codes are left to a future revision of this specification.

Requesters and Responders match RPC Calls to RPC Replies using the XID carried in each RPC message. Responders MUST send RPC Replies on the same stream on which they received the matching RPC Call.

Each QUICv1 stream provides reliable in-order delivery of bytes. However, each stream makes no guarantees about delivery order with regard to bytes on other streams on the same connection.

The stream data containing RPC records is carried by QUIC STREAM frames, but this framing is invisible to the RPC layer. The transport layer buffers and orders received stream data, exposing only a reliable byte stream to the RPC layer. Although QUIC permits out-of-order delivery within a stream, RPC-over-QUIC does not make use of this feature.

Because each QUICv1 stream is an ordered-byte stream, an RPC-over-QUIC stream carries only a sequence of complete RPC messages. Although data from multiple streams can be interleaved on a single QUICv1 connection, RPC messages MUST NOT be interleaved on one stream.

Just as with RPC on a TCP socket, each RPC message is an ordered sequence of one or more records on a single stream. Such RPC records bear no relationship to QUIC stream frames; in fact, stream frames as defined in [RFC9000] are not visible to RPC endpoints.

Each RPC record begins with a four-octet record marker. A record marker contains the count of octets in the record in its lower 31 bits, and a flag that indicates whether the record is the last record in the RPC message in the highest order bit. See Section 11 of [RFC5531] for a comparison with TCP record markers.

| NFS requirement on resends: QUIC allows reconnecting using the same connection ID, so isn't breaking/reconnection somewhat ambiguous? When can a server drop or a client resend? Any advice needed for server-side DRC implementations?
|
| lars: I'm not sure I understand what is meant by "reconnecting" above. Is this referring to connection migration? Or a 0-RTT repeated connection instance? Something else?
|
| lars: Also, I'm not sure if the use of streams is fully specified by the above. Is the intent here to leave it to callers to decide if they want to use a fresh stream for each RPC, or reuse an existing stream for a series of RPCs?
|
| cel: We need to define a server backpressure mechanism akin to the TCP window.
|
| cel: An extension process for defining RPC-over-QUIC application error codes (used with RESET_STREAM and CONNECTION_CLOSE frames) needs to be specified, likely with an IANA registry. Initial code points are needed for at least protocol-violation signaling, server backpressure, and server-initiated request drop.

3.4.1. Receiver Data Placement Assistance

One recurring weakness with RPC on TCP is that large payloads (for instance, in NFS WRITEs) can land at arbitrary offsets in receive buffers, limiting the ability for receivers to handle the payloads with zero-touch tactics such as direct I/O.

It remains an open question whether RPC-over-QUIC should implement RDMA-like features or features that simply provide help with data placement on receivers. Possibilities include:

* A single additional integer giving the offset of a payload, serving only as a hint;

* Include references to separate streams in the same connection that contain opaque payloads, similar to RDMA chunks; this would presume that it is valid for some streams on a QUIC connection to carry traffic that is not in the form of an RPC message sequence.

Long-term there could be interest in supporting RDMA over QUIC. Direct data placement over TCP can already be accomplished today using MPA/DDP protocols (formerly known as iWARP; see [RFC5040]). Using a software iWARP implementation means no special hardware is required.

If the MPA/DDP protocols themselves can be made to operate directly on QUIC transports, much of the need for a separate RPC-over-QUIC becomes moot. It would bring transport layer security to other RDMA-enabled protocols, such as RPC-over-RDMA [RFC8166].

3.5. QUIC Load Balancing

The QUIC Load Balancing specification [I-D.ietf-quic-load-balancers] defines methods for encoding routing information in QUIC connection IDs, so that a load balancer can route packets to a particular backend server with minimal per-connection state, even when a client’s network address changes due to NAT rebinding or migration. Because RPC-over-QUIC may carry multiple streams over one connection (see Section 3.3), routing at the connection-ID layer keeps all streams of a given connection on the same backend.

The connection ID length self-encoding feature of QUIC-LB assists hardware cryptographic offload devices that look up connection-specific keys.

RPC-over-QUIC implementations MAY use QUIC-LB. A full specification of its use with RPC-over-QUIC is out of scope for this document. QUIC-LB requires no support from QUIC clients beyond conformance to [RFC9000].

4. RPC Authentication Flavors

Streams in a QUIC connection may use different RPC authentication flavors. One stream might use RPC_AUTH_UNIX, while at the same time, another might use RPCSEC_GSS.

GSS mutual (peer) authentication occurs only after a QUIC connection has been established. It is a separate process, and is unchanged by the use of QUIC. Additionally, authentication of RPCSEC_GSS users is unchanged by the use of QUIC.

RPCSEC_GSS can optionally perform per-RPC integrity or confidentiality protection. When operating within a QUIC connection, these GSS services become largely redundant. An RPC implementation capable of concurrently using QUIC and RPCSEC_GSS MUST use Generic Security Service Application Program Interface (GSS-API) channel binding, as defined in [RFC5056], to determine when an underlying transport already provides a sufficient degree of confidentiality.

RPC-over-QUIC implementations MUST provide the "tls-exporter" channel binding type, as defined in Section 2 of [RFC9266].

5. Implementation Status

This section is to be removed before publishing as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

There are no known implementations of RPC-over-QUIC as described in this document.

6. Security Considerations

RPC-over-QUIC inherits the transport-layer security properties of QUIC version 1, including always-on encryption, authentication, and integrity protection of QUIC packets, and the connection migration protections discussed in Section 21 of [RFC9000] and Section 9 of [RFC9001]. Implementers should be familiar with that material.

Because QUIC integrates TLS into the transport handshake, the peer identities established during connection establishment apply to every stream and RPC message carried on that connection, as described in Section 3.2.1. The peer authentication, PKIX Extended Key Usage, and certificate-handling considerations in Section 6 of [RFC9289] apply equally to RPC-over-QUIC.

When RPCSEC_GSS is used concurrently with RPC-over-QUIC, the GSS confidentiality and integrity services are largely redundant with the protection supplied by the QUIC connection. To detect this condition, implementations use the GSS-API channel-binding mechanism [RFC5056], as required by Section 4. The "tls-exporter" binding type [RFC9266] ties the GSS context to the specific TLS handshake of the underlying QUIC connection; a successful binding check confirms that the GSS context and the RPC traffic share the same protected channel.

QUIC permits the use of 0-RTT data, which is replayable as described in Section 9.2 of [RFC9001]. RPC procedures are generally not idempotent, so the same replay hazard that motivates the 0-RTT prohibition in Section 6 of [RFC9289] applies to RPC-over-QUIC. RPC-over-QUIC implementations MUST NOT use 0-RTT data.

7. IANA Considerations

RFC Editor: In the following subsections, please replace RFC-TBD with the RFC number assigned to this document. Furthermore, please remove this Editor's Note before this document is published.

7.1. Netids for RPC-over-QUIC

Each new RPC transport is assigned one or more RPC "netid" strings. These strings are an rpcbind [RFC1833] string naming the underlying transport protocol, appropriate message framing, and the format of service addresses and ports, among other things.

This document requests that IANA allocate the following "Netid" registry strings in the "ONC RPC Netid" registry, as defined in [RFC5665]:

NC_QUIC	"quic"
NC_QUIC6	"quic6"

These netids MUST be used for any transport satisfying the requirements described in this document. The "quic" netid is to be used when IPv4 addressing is employed by the underlying transport, and "quic6" for IPv6 addressing. IANA should use this document (RFC-TBD) as the reference for the new entries.

> | lars: Why one per IP address family? This seems common
> | practice with netids, but also seems to be a layering
> | violation?
> |
> | cel: That question might be out of scope for this document.
> | netids very nearly amount to technical debt at this point.

7.2. ALPN Identifier for SunRPC on QUIC

RPC-over-QUIC utilizes the same ALPN string as RPC-with-TLS does, as defined in Section 7.2 of [RFC9289]:

> Identification Sequence: 0x73 0x75 0x6e 0x72 0x70 0x63 ("sunrpc")

This document requests that a reference to (RFC-TBD) be added to the SunRPC protocol entry in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry.

> | lars: If changes to the RPC-over-QUIC binding might be desired
> | in the future, how would they be negotiated/expressed? Should
> | a versioned ALPN be used instead of the one from [RFC9289]?

8. References

8.1. Normative References

[IANA] "IANA Service Name and Transport Protocol Port Number registry", n.d., <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt>.

[RFC1833] Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, DOI 10.17487/RFC1833, August 1995, <https://www.rfc-editor.org/rfc/rfc1833>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/rfc/rfc5056>.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, DOI 10.17487/RFC5531, May 2009, <https://www.rfc-editor.org/rfc/rfc5531>.

[RFC5665] Eisler, M., "IANA Considerations for Remote Procedure Call (RPC) Network Identifiers and Universal Address Formats", RFC 5665, DOI 10.17487/RFC5665, January 2010, <https://www.rfc-editor.org/rfc/rfc5665>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC9266] Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, DOI 10.17487/RFC9266, July 2022, <https://www.rfc-editor.org/rfc/rfc9266>.

[RFC9289] Myklebust, T. and C. Lever, Ed., "Towards Remote Procedure Call Encryption by Default", RFC 9289, DOI 10.17487/RFC9289, September 2022, <https://www.rfc-editor.org/rfc/rfc9289>.

8.2. Informative References

[I-D.ietf-quic-load-balancers] Duke, M., Banks, N., and C. Huitema, "QUIC-LB: Generating Routable QUIC Connection IDs", Work in Progress, Internet-Draft, draft-ietf-quic-load-balancers-21, 27 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-load-balancers-21>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

[RFC5040] Recio, R., Metzler, B., Culley, P., Hilland, J., and D. Garcia, "A Remote Direct Memory Access Protocol Specification", RFC 5040, DOI 10.17487/RFC5040, October 2007, <https://www.rfc-editor.org/rfc/rfc5040>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8166] Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call Version 1", RFC 8166, DOI 10.17487/RFC8166, June 2017, <https://www.rfc-editor.org/rfc/rfc8166>.

[RFC8881] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <https://www.rfc-editor.org/rfc/rfc8881>.

Acknowledgments

The authors express their deepest appreciation for the contributions of J. Bruce Fields who was an original author of this document. In addition, we are indebted to Lars Eggert and the QUIC working group for the creation of the QUIC transport protocol.

The editor is grateful to Bill Baker, Greg Marsden, Richard Scheffenegger, Martin Thomson, and Long Xin for their input and support.

Special thanks to Area Director Gorry Fairhurst, NFSV4 Working Group Chairs Brian Pawlowski and Christopher Inacio, and NFSV4 Working Group Secretary Thomas Haynes for their guidance and oversight.

Authors' Addresses

Benjamin Coddington
Hammerspace
United States of America
Email: bcodding@hammerspace.com

Scott Mayhew
Red Hat
United States of America
Email: smayhew@redhat.com

Charles Lever (editor)
Oracle Corporation
United States of America
Email: chuck.lever@oracle.com