

Network File System Version 4  
Internet-Draft  
Intended status: Standards Track  
Expires: 3 November 2025

B. Coddington  
S. Mayhew  
Red Hat  
C. Lever, Ed.  
Oracle  
2 May 2025

Remote Procedure Call over QUIC Version 1  
draft-cel-nfsv4-rpc-over-quicv1-02

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](mailto: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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 November 2025.

## Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Motivation For a New RPC Transport . . . . .	3
2. Requirements Language . . . . .	3
3. RPC-over-QUIC Framework . . . . .	4
3.1. Establishing a Connection . . . . .	4
3.2. RPC Service Discovery . . . . .	4
3.2.1. Transport Layer Security . . . . .	5
3.3. QUIC Streams . . . . .	5
3.4. RPC Message Framing . . . . .	6
3.5. Stream Count . . . . .	7
4. RPC Authentication Flavors . . . . .	8
5. Implementation Status . . . . .	9
6. Security Considerations . . . . .	9
7. IANA Considerations . . . . .	9
7.1. Netids for RPC-over-QUIC . . . . .	9
7.2. ALPN Identifier for SunRPC on QUICv1 . . . . .	10
8. References . . . . .	10
8.1. Normative References . . . . .	10
8.2. Informative References . . . . .	11
Acknowledgments . . . . .	12
Authors' Addresses . . . . .	12

## 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 QUICv1. Readers should already be familiar with the fundamentals of ONC RPC (see [RFC5531]).

#### 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, the peer that initiates QUICv1 connection establishment is referred to as an "RPC client" peer. The peer that passively accepts the connection is referred to as an "RPC server" peer.

QUICv1 connections are not defined by the classic 5-tuple (IP proto, source address, source port, destination address, and destination port). Rather, each connection is defined by its connection ID. Thus, 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 to belong to an established connection.

As a result, due to network conditions or administrative actions, an RPC-over-QUICv1 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-QUICv1. A more complete discussion can be found in Section 9 of [RFC9000].

#### 3.2. RPC Service Discovery

Because all QUICv1 traffic goes to a single port and is demultiplexed on the receiving peer, the usual precursor step of the RPC client sending an rpcbind query to the RPC server before an RPC-over-QUICv1 connection is to be established is unneeded.

Notice that we now have to invent a new way to perform proper RPC service discovery. The historical method of discovering an RPC service is to send an rpcbind query to a well-known port number (port 111). The result of an rpcbind query contains the IP address and port number of the requested RPC service.

For QUICv1, the port number is part of the QUICv1 connection, and is the same for all QUICv1 traffic directed to that network host.

### 3.2.1. Transport Layer Security

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 authentication identity that was used during connection establishment.

RPC-over-QUICv1 provides peer authentication and encryption services using a framework based on Transport Layer Security (TLS). Ergo, RPC-over-QUICv1 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. Thus, there is no need or use for the STARTTLS mechanism described in Section 4 of [RFC9289].
- \* The discussion in [RFC9289] about the opportunistic use of TLS does not apply to RPC-over-QUICv1.
- \* The peer authentication requirements in Section 5.2 of [RFC9289] do apply to RPC-over-QUICv1.
- \* The PKIX Extended Key Usage values defined in [RFC9289] are also valid for use with RPC-over-QUICv1.
- \* The ALPN defined in Section 8.2 of [RFC9289] is also used for RPC-over-QUICv1.

### 3.3. QUIC Streams

RPC-over-QUICv1 connections are mediated entirely by each peer's RPC layer and are not visible to RPC applications. An RPC client establishes a QUICv1 connection whenever there are application RPC requests 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-QUICv1, this is a stream. As an example, an NFSv4.1 BIND\_CONN\_TO\_SESSION operation [RFC8881] binds to a QUICv1 stream. As another example, to signify the loss of an RPC request, an NFS server closes the QUICv1 stream that received that request, but it does not close the encompassing QUICv1 connection.

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

### 3.4. RPC Message Framing

RPC-over-QUICv1 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 its Requester/Responder role.

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.

Because each QUICv1 stream is an ordered-byte stream, an RPC-with-QUICv1 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.

### 3.5. Stream Count

Given the above definition of RPC message framing on QUICv1 streams, it is possible for a Requester to create a stream, send one RPC Call, receive one RPC Reply, then destroy the stream. That is a degenerate case that might be used for simple RPC-based protocols like rpcbind.

For protocols that carry a significant workload, this style of stream allocation would generate needless overhead. Moreover, stream identifiers cannot be re-used on one QUICv1 connection, so eventually a QUICv1 connection can no longer create a new stream for each RPC XIDs. And finally, a connection peer may advertise a `max_streams` value that is significantly lower than  $2^{60}$ .

Instead, RPC clients may create as many streams as is convenient to their design, but should reuse streams on each connection efficiently.

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 session.

Servers that implement RPC-over-QUICv1 must be mindful 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 this.

bfields: Open question whether we should do something more complicated that adds RDMA-like features or at least provides some minimal help with data alignment. Possibilities might include a single additional integer giving the offset of a payload, serving only as a hint; or reference additional streams in the same connection for payloads; or even looking into full RDMA--long-term there may be interest in supporting RDMA over QUIC, and we may be able to piggyback on that effort.

cel: Direct data placement over TCP can already be accomplished today using MPA/DDP protocols (formerly known as iWARP). Using a software iWARP implementation means no special hardware is necessary. Likewise, if MPA/DDP can be made to support QUIC, much of the need for a separate RPC-over-QUIC is moot. In addition, it would automatically bring transport layer security to other RDMA-enabled protocols (such as RPC-over-RDMA).

cel: I will say however that it should be possible to create a "chunk" abstraction (similar to RPC-over-RDMA) where the RPC-over-QUICv1 protocol header identifies octet ranges of an RPC

message that are to be sent via other QUIC streams. This would assume 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.

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]?

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: Still not clear how RPC program/version discovery will work in a world with no endpoint port numbers.

cel: Should we limit each stream to carry only on 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.

#### 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 [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-QUICv1 as described in this document.

## 6. Security Considerations

Readers should refer to the discussion of QUIC's transport layer security in Section 21 of [RFC9000].

## 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 QUICv1

RPC-over-QUICv1 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.

## 8. References

### 8.1. Normative References

- [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

- [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>>.
- [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, and Martin Thomson 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  
Red Hat  
United States of America  
Email: [bcodding@redhat.com](mailto:bcodding@redhat.com)

Scott Mayhew  
Red Hat  
United States of America  
Email: [smayhew@redhat.com](mailto:smayhew@redhat.com)

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