Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 7 August 2026

C. Caini, Ed.
ARCES, University of Bologna
T. de Cola
German Aerospace Center, DLR
M. Moffa
ARCES, University of Bologna

3 February 2026

Delay-Tolerant Networking QUIC Convergence Layer Protocol Version 1
draft-caini-dtn-quiccl-00

Abstract

This document describes a QUIC convergence layer (QUICCL) for Delay-Tolerant Networking (DTN). QUICCL uses BPv7 bundles encoded by the Concise Binary Object Representation (CBOR) as its service data unit being transported and makes use of either QUIC streams (RFC 9001) or QUIC datagrams (RFC 9221) to transport such bundles. QUICCL design is largely based on TCPCLv4 specification (RFC9174), with three significant differences. First, as QUIC incorporates TLS security, all security related parts of TCPCLv4 were dropped. Second, QUICCL provides two new transport services, notified and unreliable, in addition to the reliable service; Third, in the reliable service, by taking advantage of QUIC streams, four levels of priority are offered.

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 7 August 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Document Description

This document describes the QUIC convergence-layer protocol (adapter) for Delay-/Disruption-Tolerant Networking (DTN). DTN is an end-to-end architecture enabling communications in "challenged networks", including networks with at least one of the following challenges: intermittent connectivity, long and/or variable delays, high packet loss due to link impairments. Among challenged networks, we have interplanetary networks, other space networks, and a variety of

peculiar terrestrial and maritime networks. See [RFC4838].

The goal of the DTN architecture is to enable communications in challenged networks; to this aim it relies on the insertion of a new layer, called Bundle layer, in between the Application layer and (usually) the Transport layer. The corresponding protocol is the Bundle Protocol (BP), whose version 7 (BPv7) has been standardized in [RFC9171]. BPv7 requires the services of a "Convergence Layer Adapter" (CLA) to send and receive bundles using the services offered by a lower protocol. This document describes one such convergence-layer adapter that uses the QUIC protocol [RFC9000], including its Datagram extension [RFC9221]. This CLA is referred to as QUIC Convergence Layer version 1 (QUICCLv1); the A of adapter is dropped to be consistent with the acronyms in use for other CLAs, such as TCPCL, UDPCL, etc. For the remainder of this document:

* the abbreviation "BP" without the version suffix refers to BPv7

* the abbreviation "QUICCL" without the version suffix refers to QUICCLv1, the QUICCL version described in this document.

QUICCL and the Bundle Protocol do not fit well in the Internet model protocol stack, because the DTN architecture differs from the usual TCP/IP architecture, as a result of the addition of the Bundle layer. Note that the Bundle layer, being present on all DTN nodes, not only on end-points, cannot be interpreted as an Application sublayer (Application is present only on end-nodes). As an example, a possible protocol stack, referring to an end-node, is presented in Figure 1. QUICCL is a slim protocol located between BP and QUIC; as its only possible use is below BP, we have located it in the Bundle layer. Note that QUIC and UDP are both at Transport, as QUIC makes use of UDP. A DTN intermediate node would have the same stack, but without the DTN application on top of BP. Non DTN nodes may be present in between two adjacent DTN nodes, i.e. inside one DTN hop. In such a case, they will be transparent to the DTN architecture.

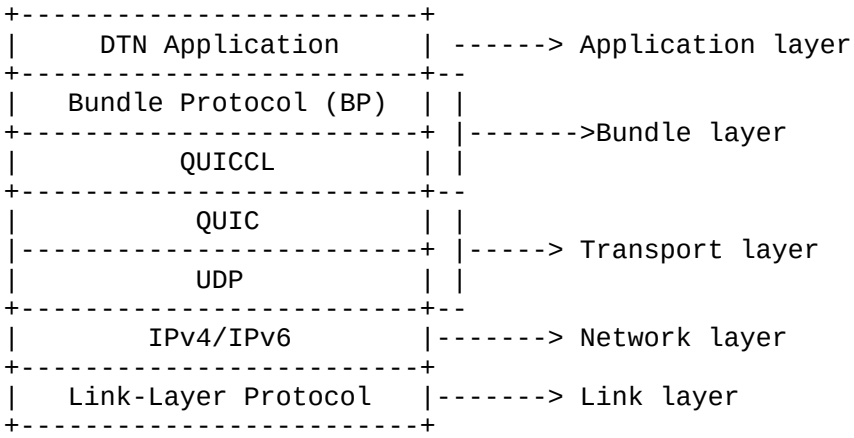

Figure 1: An example of Protocol stack in the DTN architecture; DTN end-node making use of QUICCL

1.2. Scope

This document describes the services provided by QUICCL to BP, the data to be exchanged between the BP and the QUICCL interface, the format of the protocol messages passed between QUICCL peers, and the internal mechanism of QUICCL necessary to provide the services offered to BP.

1.3. 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.

1.4. Definitions

This section contains a few definitions of terms used in this text.

Network Byte Order: Here, "network byte order" means most significant byte first, i.e. big endian. All of the integer encodings in this protocol SHALL be transmitted in network byte order.

QUICCL Entity: The notional QUICCL application that initiates QUICCL

sessions. This design, implementation, configuration, and specific behavior of such an entity is outside of the scope of this document. However, the concept of an entity has utility within the scope of this document as the container and initiator of QUICCL sessions. The relationship between a QUICCL entity and QUICCL sessions is defined as follows:

* A QUICCL entity MAY actively initiate any number of QUICCL sessions and should do so whenever the entity is the initial transmitter of information to another entity in the network.

* A QUICCL entity MAY support zero or more passive listening elements that listen for connection requests from other QUICCL entities operating on other entities in the network.

* A QUICCL entity MAY passively initiate any number of QUICCL sessions from requests received by its passive listening element(s) if the entity uses such elements.

These relationships are illustrated in Figure 2. For most QUICCL behavior within a session, the two entities are symmetric and there is no protocol distinction between them. Some specific behavior, particularly during session establishment, distinguishes between the active entity and the passive entity. For the remainder of this document, the term "entity" without the prefix "QUICCL" refers to a QUICCL entity.

QUIC Connection: The term "connection" in this specification exclusively refers to a QUIC connection [RFC9000].

QUICCL Session: A QUICCL session (as opposed to a QUIC connection) is a QUICCL communication relationship between two QUICCL entities. A QUICCL session operates within a single underlying QUIC connection, and the lifetime of a QUICCL session is bound to the lifetime of that QUIC connection. A QUICCL session is terminated when the QUIC connection ends, due to either (1) one or both entities actively closing the QUIC connection or (2) network errors causing a failure of the QUIC connection. Within a single QUICCL session, there are usually multiple Transfer flows, each corresponding to a QUIC stream plus one flow for QUIC datagrams, which are not associated to any flows. The Transfer flows are logically unidirectional, i.e. either outbound or inbound. In an outbound flow data (XFER_SEGMENT messages) are sent to the peer, while the reverse direction is reserved to acknowledgments (XFER_ACKs); in an inbound flow, the dual happen (see Figure 3. Note that mutiple parallel flows are allowed, although in the figure only one per each direction is shown. From the perspective of a QUICCL session, the Transfer flows do not logically interact

with each other. The flows operate over the same QUIC connection and between the same BPAs (Bundle Protcol Agents). For the remainder of this document, the term "session" without the prefix "QUICCL" refers to a QUICCL session.

Session Parameters: The set of values that affect the operation of QUICCL for a given session. The manner in which these parameters are conveyed to QUICCL is an implementation matter.

Transfer: The conveyance of one bundle from a node to the next. Each transfer within the QUICCL is identified by a Transfer ID number, which is guaranteed to be unique only to a single direction within a single session.

Transfer Segment: A transfer segment (XFER_SEGMENT) is a subset of the whole data communicated during a transfer, i.e. a portion of the original (CBOR encoded) bundle. It may coincide with the whole bundle if the bundle is relatively small.

Transfer flow: The unidirectional sequence of transfers carried out between two QUICCL entities by using the same QUIC stream (for the reliable service) or by using QUIC datagrams (for the notifid and the unrelaible services).

Idle Session: A QUICCL session is idle while there is no transmission in progress in either direction (i.e. when all Transfer flows are idle). While idle, the only messages being transmitted or received are KEEPALIVE messages.

Live Session: A QUICCL session is live while there is a transmission in progress in either direction (i.e. when at least one Transfer flow is live).

Reason Codes: The codes used to specify the origin of failures or error message types.

The relationship between QUIC connections, QUICCL sessions and QUICCL flows is shown in Figure 3. Note that to preserve the figure readibility, only one Transfer flow per each direction is presented, whule they could be many.

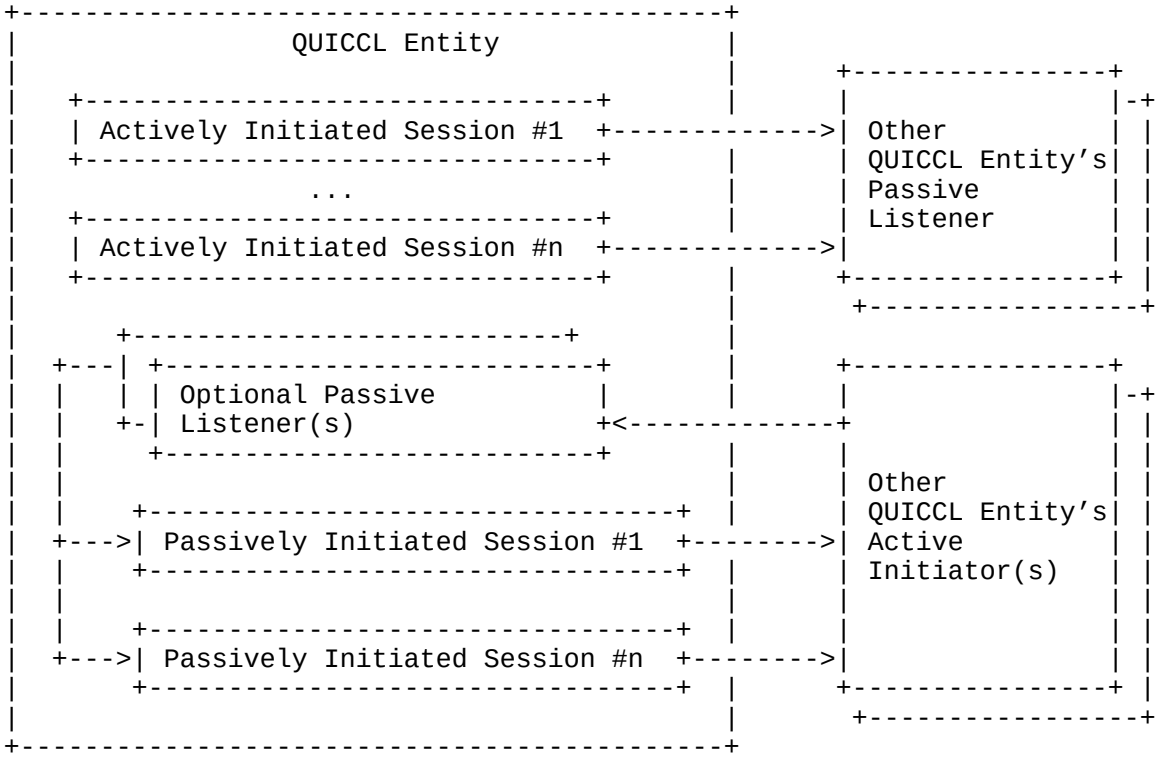

Figure 2: Relationship between QUICCL entities

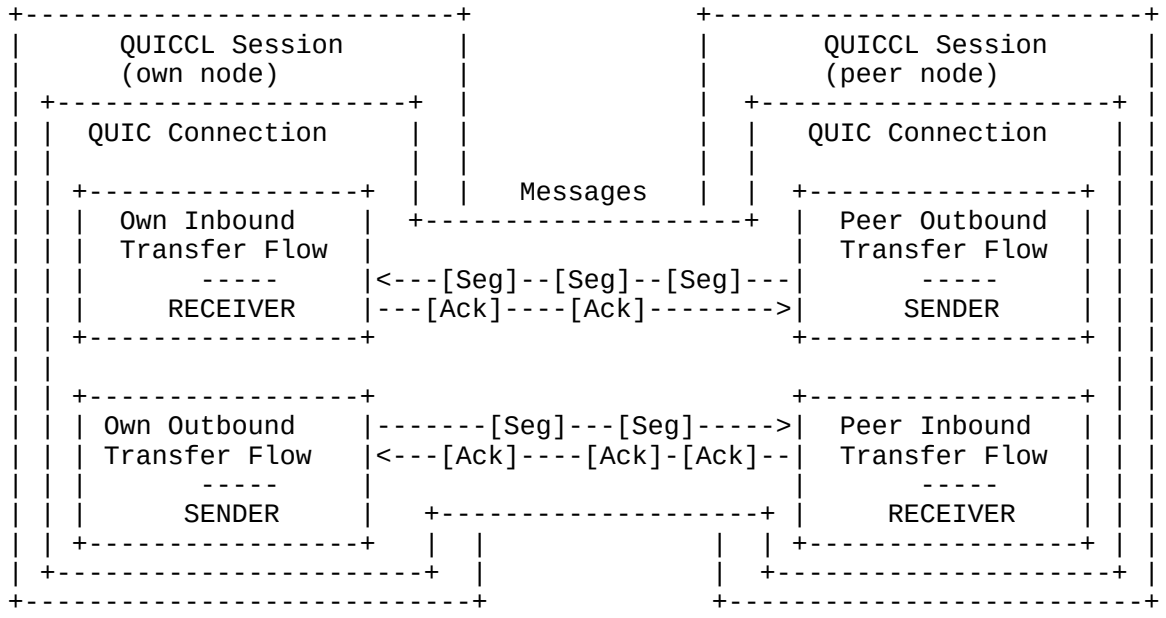

Figure 3: QUICCL Session, QUIC Connection and Transfer Flows (for better readibility, only one Transfer flow per each direction is presented, but we could have many in parallel).

2. Main Differences with TCPCLv4

QUICCL is designed after TCPCLv4 [RFC9174] to take the most of the commonality between TCP and QUIC. However, the integration of security in QUIC and the new services offered by QUICCL require the introduction of some modifications to TCPCLv4 messages and procedures. For a better comprehension of these modifications, described in the next sections, it is convenient to start with an indepth overview of the differences between QUICCL and TCPCL. The name of the few structures or parameters involved in the modifications will be anticipated, to the benefit of people already familiar with TCPCLv4.

2.1. Security Incorporated in QUIC

QUIC design incorporates TLS (Transport Layer Security) security, thus all TCPCLv4 specifications related to TLS security have been omitted in QUICCL. This also made redundant the exchange of TCPCLv4 CONTACT_HEADER messages, whose primary aim was to indicate that the sending peer had enabled TLS security. CONTACT_HEADERs are thus no more present in QUICCL.

2.2. QUICCL Transport Services

A second fundamental difference with TCPCLv4 is that QUICCL provides three different transport services, reliable, unreliable and notified, instead of the sole reliable service. Under this respect, it is like LTP [RFC5326], which was designed to provide both a reliable and an unreliable service. The third transport service, notified, is similar to the analogous service introduced in Multicolor LTP, an LTP enhancement proposed by Unibo and DLR in [Bisacchi_2022]. In most BP/LTP implementations, such as ION [Burleigh_2007] and Unibo-BP [Caini_2024], the transport service selection of LTP is made bundle by bundle, on the basis of QoS elements contained in a QoS extension of the bundle itself. For example, in the ION implementation by NASA-JPL, by default LTP provides BP with a reliable service (LTP red), but the bundle source can override this deafault in favor of the unreliable service (LTP green), by setting the "best effort" flag in the ECOS (Extended Class of Service) bundle extension [I-D.burleigh-dtn-ecos]. A similar mechanism is present in Unibo-BP. QUICCL formalizes and extends this mechanism, as shown below, by taking advantage of the fact that although QUIC was initially designed as a reliable-only Transport protocol, based on QUIC streams [RFC9000] , it was later extended to offer also an unreliable service, with QUIC datagrams [RFC9221].

2.2.1. Rationale

The selection of the transport service inside the same CL, as done in QUICCL, is an example of QoS mapping from one protocol to the protocol below: the QoS elements of a bundle are mapped to the different services offered by the CL. In other words, the internal selection is a particular case of a very general concept, which can be better formalized and standardized than its alternative, i.e. the choice between different CLs each of which offering a single service (TCPCL, UDPCL, etc.). Besides this theoretical consideration, having multiple transport services offered by the same protocol, as in QUICCL, offers also some practical advantages. First and foremeost, node configuration is easier and less error prone, as it is sufficient to configure one CL between two adjacent nodes instead of two or more CLs, such as TCPCL and UDPCL. Second, in the QUICCL case

we have also the advantages that derive from the introduction of datagrams in QUIC: common security, and common congestion and flow controls (see section 2 of [RFC9221]). In brief, all transport services in QUIC, and thus also in QUICCL, are covered by the QUIC security; one mechanism and one key for all services. Analogously, for congestion and flow controls: QUIC streams and QUIC datagrams are subjected to aggregated congestion and flow controls, while with TCPCLv4 and UDPCL used in parall, we would have these controls only on TCP traffic, which is clearly much less convenient.

2.2.2. Reliable Service

The reliable service of QUICCL makes use of bidirectional QUIC streams [RFC9000], used however by QUICCL in a logical unidirectional way. The bundle to send is divided by QUICCL into multiple XFER_SEGMENTs, all to be transferred on the same stream, and each segment has to be acknowledged by a XFER_ACK using the reverse direction of the same stream. As a QUIC connection may consists of multiple streams, QUICCL may transmit bundles in parallel. On each Transfer flow, corresponding to a different QUIC stream, bundles can be pipelined, as shown in Figure 3. The possible QUICCL mappings of bundles to streams, will be discussed below, when dealing with priorities.

2.2.3. Unreliable Service

The unreliable service is based on QUIC datagrams, whose content, if lost, is never retransmitted. A relatively large bundle is divided into multiple XFER_SEGMENTs, as in the reliable service, but now each segment must fit into a QUIC datagram (which in turns must fit into a UDP datagram, etc.), which poses a more stringent limit to the maximum transfer segment length than in the case of the reliable service. This has led to the introduction of the additional field "Datagram MRU" in the SESS_INIT message (see Section 4.4.1). A second important difference, is that segments sent with the unreliable service are never acknowledged, to avoid unnecessary traffic in the reverse direction. This has required the insertion of the "Transfer mode" field in the original TCPCLv4 XFER_SEGMENT message. Note that the establishment of a QUIC connection always require an initial handshake between peer, thus neither QUIC nor QUICCL can be used on a unidirectional channel, independently of the service requested, by contrast to UDPCL or LTP (the latter when limited to the unreliable service). On the sender peer, once all XFER_SEGMENTs have been sent, the BPA is notified a success, meaning successful sending, not succesful reception, as in the case of the reliable service. On the receiver side, it is necessary to reassemble the segments and to know if some of them are missing; this not to ask for retransmissions, but simply not to pass incomplete

bundles to the BPA. To manage reassembling, two fields, "Segment ID" and "Total number of segments" have been added to XFER_SEGMENT messages. To cover the corner case of the loss of the last segment, an intersegment timer must be introduced, whose value is to be set in QUICCL configuration. If a new segment does not arrive in the due time, this timer fires and the arrived segments of the incomplete transfer are discarded. Bundles sent via QUIC datagrams, as those of the unreliable and notified services, are sent in a serialized way, possibly in parallel to those sent on QUIC streams; as said, their transfer is subjected to the same aggregate QUIC congestion and flow controls. As QUIC datagrams do not belong to any QUIC streams, XFER_SEGMENTs of the unreliable and notified services are associated to an additional and unique Transfer flow, devoted to datagram traffic. As for other flows, XFER_SEGMENTs belonging to different transfers MUST NOT be inerlaced, but they can be pipelined (first all segments of the first transfer, then all of the second and so on).

2.2.4. Notified Service

The QUICCL notified service is analogous to the homonymous service offered by the Orange color in Multicolor LTP [Bisacchi_2022]. This notified service is based on QUIC datagrams as the unreliable service, but now XFER_SEGMENTS must be confirmed by XFER_ACK, as in the reliable service; in this way the QUICCL sender can deduce whether all segments have arrived or not at the receiver peer in the due time. In the former case, BP is notified a success, meaning successful reception of the bundle; in the latter, no retransmissions are performed by the QUICCL entity and a failure is notified to BPA. Two inter-segment timers (one on the sender and the other on the receiver) analogous to that of the unreliable service (on the receiver only) cover the case of either possible missed XFER_SEGMENTs or XFER_ACKs, respectively.

The advantage of the notified service is that neither the QUICCL sender nor the QUICCL receiver needs to buffer large amount of data for long time, waiting for QUIC retransmissions, as it could happen with the reliable service in large BDP (Bandwidth Delay Protuct) links. As a safety measure, to avoid possible consecutive failures in case BP retransmits the bundle after notification of a CL failure, a not obligatory policy that is however adopted by most BPv7 implementations, the notified service SHOULD be allowed only when a bundle is transmitted by BP for the first time. In case of failure, retransmissions of the same bundle by BP SHOULD be performed by using the reliable service instead. To this end, the BP implementation SHOULD pass the number of retransmissions to QUICCL, in addition to the bundle itself and to QoS elements.

2.2.5. Service Selection

As anticipated above, we can state that transport service selection is performed by QUICCL by mapping bundle QoS items into a different QUICCL service. At present, in Unibo-BP implementation of QUICCL this is accomplished by making use of two ECOS [I-D.burleigh-dtn-ecos] flags, "best effort" and "reliable", thus extending what already done with BP/LTP in ION. The very simple mapping rule applied is summarized in Table 1, which also considers the number of retransmissions. Note that this QoS mapping is the same as already considered in [Caini_2024] with reference to Multicolor LTP services. This mapping is however only an example, as the way bundle QoS items are passed to QUICCL, and thus how the mapping is performed, is left to BP and QUICCL implementations.

Best effort	Reliable	ReTx number	QUICCL service
0	0	whatever	default
1	0	whatever	unreliable
0	1	whatever	reliable
1	1	0	notified
1	1	>0	reliable

Table 1

2.3. Priorities

The use of multiple QUIC streams is advantageous in HTTP, as in case of a loss on a stream, data sent on other streams can be delivered before the loss is recovered, which solves the "head of line blocking problem". This distinctive feature of QUIC, however, when applied to bundles has two significant drawbacks. First, bundles (of the same dimension) sent in parallel on N streams tend to arrive in a burst, instead of one by one, as when sent on a single stream. Note that the total transfer time is roughly the same, because when bundles are sent in parallel the bandwidth is shared (QUIC congestion control works on the aggregate of streams and includes also also QUIC datagram, as previously said), but now bundles tend to arrive together instaed of paced, which is per se a disadvantage. Moreover, bundles may arrive out of order, either becasue of minimal time differences between streams, or because one tranfer needed more time in order to recover from a loss (loss recovery is per stream in

QUIC). Although not forbidden by BP specification, the introduction of disordered delivery is generally undesired; there is however an important exception, i.e. when bundles have different priorities. In this case, disordered delivery is actually what the user wants, if this means that bundles with higher priority arrive before bundles with lower priority.

2.3.1. Priorities in QUICCL

For this reason, QUICCL default mapping for the reliable service consists in using four (bidirectional) QUIC streams, one for each of the three bundle cardinal priorities, expedited, normal and bulk [RFC4838], plus one for bundles without any priority indication. These four streams SHOULD also be associated with different QUIC priorities, so that if two (equal dimension) bundles are passed concurrently to QUICCL by BP, the higher priority bundle is delivered first. Note that this is a distinguishing feature of QUICCL, as almost all other CLs are based on Transport protcols lacking priorities. A practical example may help: thanks to the priority support, QUICCL can prevent a large bulk bundle from clogging a link with limited bandwidth, as normal and expedited bundles sent later (if allowed by BP flow control) can "overtake" it, a remarkable property, indeed.

However useful, an immplementation MAY be able to override this default, by allowing the use of a different stream for each concurrent bundle or by using a fixed number of streams, say N, without priority. This includes the single stream as a particular case (N=1), in which the QUICCL behavior becomes more similar to that of TCPCLv4. Note that priorities can be enforced only when a reliable service is selected.

2.3.2. Priority-based Stream Selection

To select the right QUIC stream, when the reliable mode is selected and the default mapping applies, QUICCL needs to know the cardinal priority of a bundle. Cardinal priorities, defined in [RFC4838] and once inserted in the bundle primary block in BPv6 [RFC5050] have been removed in BPv7 [RFC9171], with the intent of putting them together in a new extension, with other QoS features. Passing from BPv6 to BPv7, most implementations, such as ION, just moved the cardinal priorities into the ECOS extension [I-D.burleigh-dtn-ecos], whose IETF draft has however expired in the meantime. An ESA proposal for a QoS extension is going to appear as a CCSDS Orange book, but Orange books are only experimental specifications, thus neither CCSDS standards nor documents in a standard track. In the lack of a unique standardized way to convey cardinal priority information the present QUICCL specification leaves implementations free to define the way

priority information is passed from BP to QUICCL. The task of QUICCL is to map the bundle priority to the right stream, and interact with the QUIC implementation to assign each stream a different priority. The special case of the absence of priority indication (e.g. if the bundle lacks any QoS extensions) is tackled by the fourth stream. The assignment of a different QUIC priority to each stream is left to the QUICCL implementation, also because it depends on the features offered by the QUIC implementation itself. The only requirement is that the priority order should be preserved: the highest bundle priority MUST be associated to the highest QUIC priority and so on.

3. General Protocol Description

The QUICCL aim is to transmit DTN bundles via QUIC; as said, to this end three transport services are provided to BP: reliable, unreliable and notified. The former uses QUIC streams, the others QUIC datagrams. The operation of the protocol follows that of TCPCLv4 as much as possible, and from now on the text and figures are often taken from [RFC9174]; the structure has however been reorganized and a few modifications have been introduced. QUICCL differences with TCPCLv4, anticipated in the previous section, will be thoroughly highlighted and discussed.

3.1. Convergence-Layer Interface to BPA

The QUICCL protocol defines the following services/notifications to support the overlaying BPA. This is not an API definition but a logical description of how the CL can interact with the BPA. Some services/notifications are optional as indicated below. Each of these interactions can be associated with any number of additional metadata items as necessary to support the operation of the CL or BPA.

BPA commands to QUICCL:

Attempt Session: The QUICCL MUST allow the BPA to preemptively attempt to establish a QUICCL session with a peer entity. Each session attempt may have a different set of session negotiation parameters as directed by the BPA.

Terminate Session: The QUICCL MUST allow the BPA to preemptively terminate an established QUICCL session with a peer entity.

Begin Transmission: The QUICCL MUST allow the BPA to transmit a bundle over an established QUICCL session.

Interrupt Reception (OPTIONAL): The QUICCL MAY allow the BPA to

interrupt the reception of a transfer before it has fully completed. Interruption can occur any time after the reception is initialized.

QUICCL notifications to BPA

Session State Changed: The QUICCL entity can indicate some session state changes to the BPA. Unless marked as OPTIONAL, the following changes MUST be provided.

Connecting (OPTIONAL): A QUIC connection is being established (active entity only).

Session Negotiating (OPTIONAL): After the QUIC connection has been established, session negotiation has begun.

Established: The session has entered the established state and is ready for its first transfer. The peer node ID and the negotiated session parameters are also to be communicated to the BPA.

Ending (OPTIONAL): The entity sent a SESS_TERM message and has entered the Ending state.

Terminated: The session has completed the normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed (OPTIONAL): The QUICCL entity MAY indicate to the BPA when the Live/Idle substate of the session changes. This occurs only when the top-level session state is "Established". The session transitions from Idle to Live at the start of a transfer in transfer streams; the session transitions from Live to Idle at the end of a transfer when the other transfer stream does not have an ongoing transfer. Note that because the QUICCL transmits in parallel over QUIC connection streams, by contrast to TCPCL it does not suffer from "head-of-queue blocking".

Transmission Success: The QUICCL entity MUST indicate to the BPA when a bundle has been fully transferred to a peer entity.

Transmission Intermediate Progress (OPTIONAL): The QUICCL entity MAY indicate the intermediate progress of a transfer to a peer entity. This intermediate progress is at the granularity of each transferred segment.

Transmission Failure: The QUICCL entity MUST indicate certain

reasons for bundle transmission failure, notably when the peer entity rejects the bundle or when a QUICCL session ends before transfer success, or when the transmission timer expires (notified service only).

Reception Initialized (OPTIONAL): The QUICCL entity MAY indicate the reception of the XFER_SEGMENT message with the START flag set to 1.

Reception Success: The QUICCL entity MUST indicate to the BPA when a bundle has been fully transferred from a peer entity.

Reception Intermediate Progress (OPTIONAL): The QUICCL entity MAY indicate the intermediate progress of a transfer from the peer entity. This intermediate progress is at the granularity of each transferred segment. An indication of intermediate reception gives a BPA the chance to inspect bundle header contents before the entire bundle is available and thus supports the "Interrupt Reception" capability.

Reception Failure (OPTIONAL): The QUICCL entity MAY indicate certain reasons for reception failure, notably when the local entity rejects an attempted transfer for some local policy reason or when a QUICCL session ends before transfer success, or when the reception timer fires (unreliable and notified services only).

3.2. Overview of a basic QUICCL session

This overview will give a brief description of a regular QUICCL session (i.e., without session failures). Its states and transitions are shown in Figure 4. Details and format of messages exchanged will be defined later.

One entity establishes a QUICCL session to the other by initiating a QUIC connection in accordance with [RFC9000]. Note that the exchange of "contact headers" of TCPCLv4 has been dropped, the reasons are fully detailed in Section 3.2.1. After the QUIC connection is established, session negatiation starts, with the exchange of two SESS_INIT messages (see Section 3.3.1) . In this phase the parameters to be used during the QUICCL session are set. Once negotiated, the QUICCL session is established and its parameters cannot change.

Once the session is established, bundles can be transferred in either direction; bundle tranmission is "full duplex": one side’s bundle transfer does not have to complete before the other side can start sending bundles on its own. Each transfer is performed by segmenting the transfer data into one or more XFER_SEGMENT messages (see Section 3.5). By contrast to TCPCLv4, multiple bundles can be transmitted in parallel in the same direction, by exploiting QUIC streams and Datagrams.

XFER_SEGMENTs are usually acknowledged by XFER_ACK messages. Although there is no explicit flow control on QUICCL, congestion and flow controls are performed by QUIC, running below.

A QUICCL receiver can interrupt the transmission of a bundle at any point in time by replying to a XFER_SEGMENT with a XFER_REFUSE message, which causes the sender to stop transmission of the associated bundle (if it has not already finished transmission).

An esatablised session can either be live, if transfers are on going, or idle, otherwise. For sessions that are idle, KEEPALIVE messages are sent at a negotiated interval.

Every peer can start the session termination procedure by sending a SESS_TERM message to the other peer, which will respond with another SESS_TERM message (see Section 3.3.3). During termination sequencing, in-progress transfers can be completed but no new transfers can be initiated. A SESS_TERM message can also be used to refuse a session setup by a peer.

At the end of the session termination procedure the QUIC connection is released.

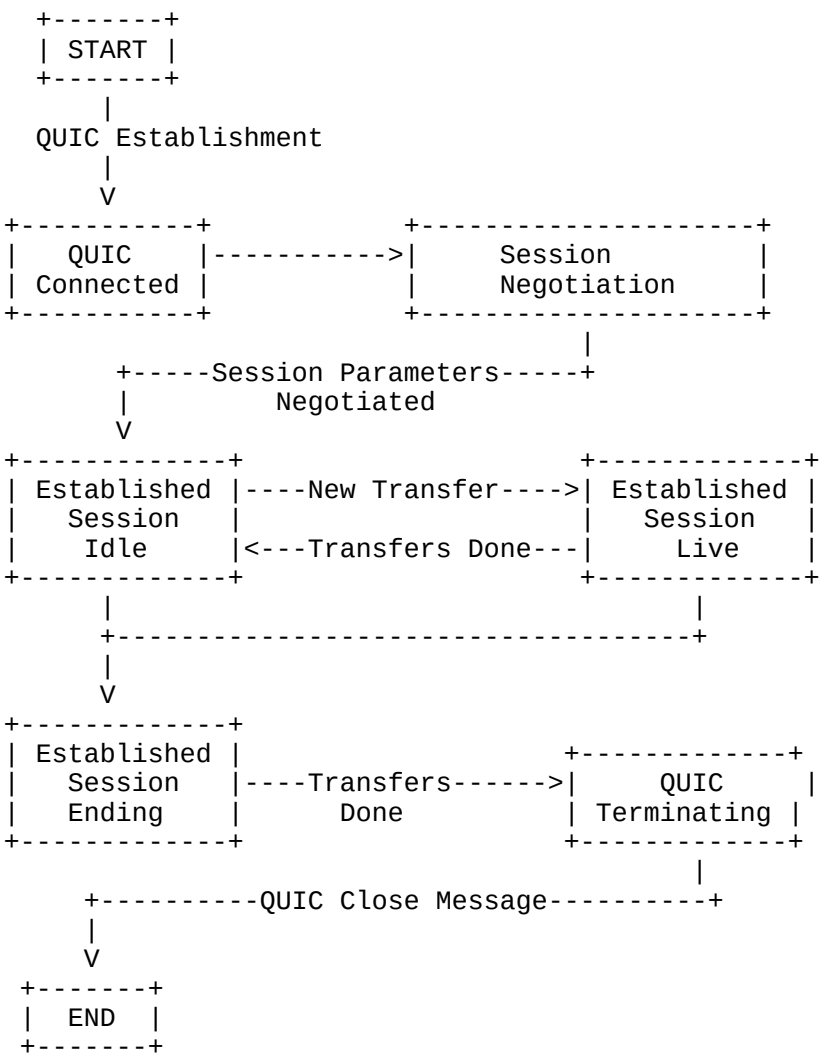

Figure 4: Top-Level States of a QUICCL Session

Notes on established session states:

* Session "Live" means transmitting or receiving over at least a Transfer flow.

* Session "Idle" means no transmission/reception over any Transfer flows.

* Session "Ending" means no new transfers will be allowed.

3.2.1. Justification of Contact Header Removal

The main function of the TCPCLv4 contact header was to negotiate the usage of Transport Layer Security. This function is no more necessary here, as in QUIC the TLS negotiation is incorporated in the connection handshake. Also the "dtn!" magic number and the version field present in the contact headers are no more needed, as QUIC makes use of TLS Application-Layer Protocol Negotiation extension (ALPN) [RFC7301]. For these reasons, contact headers have been completely stripped from the QUICCL protocol, and the only message exchange necessary for session establishment consists in the SESS_INIT exchange.

3.3. Detailed State Diagrams

3.3.1. Session Initiation State Diagarms

Session negotiation involves exchanging a session initiation (SESS_INIT) message in both directions and deriving a negotiated state from the two messages. The session negotiation sequencing is performed as either the active or passive entity and is illustrated in Figure 5 and Figure 6, respectively; in both figures, ST stands for "Session Termination" and PSI for "Processing of Session Initiation", which is expanded in Figure 7.

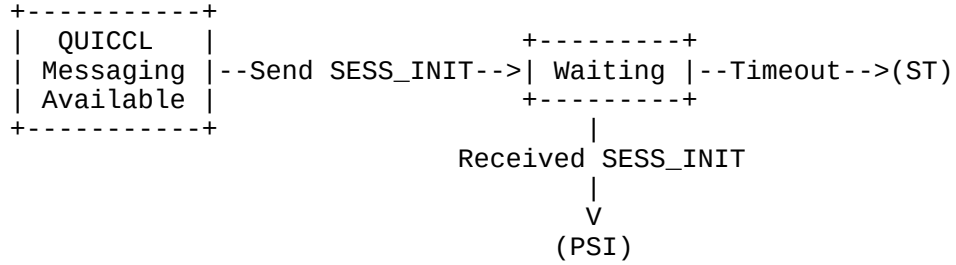

Figure 5: Session Initiation (SI) as Active Entity

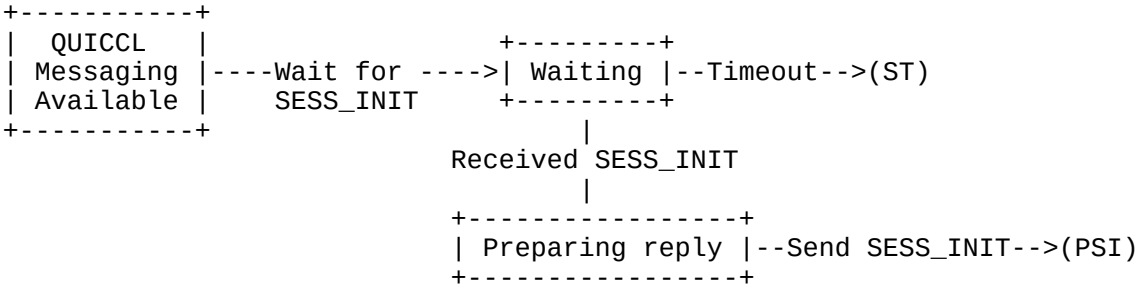

Figure 6: Session Initiation (SI) as Passive Entity

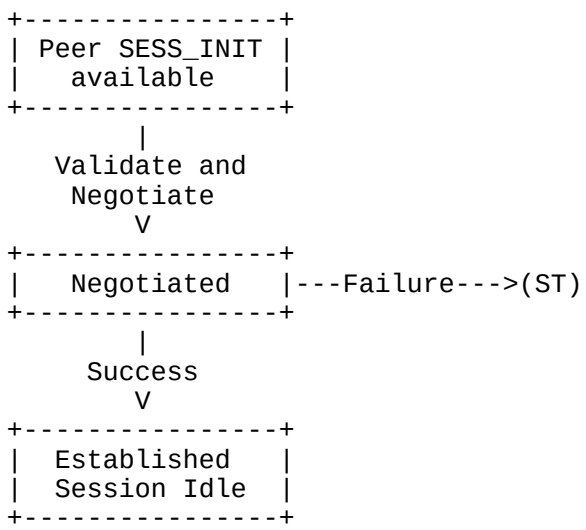

Figure 7: Processing of Session Initiation (PSI)

3.3.2. Transfer Flow State Diagrams

Transfers can occur after a session is established and it is not in the Ending state. Each transfer occurs within a single logical Transfer flow between a sender and a receiver. In a QUICCL session, we have one Transfer flow for each QUIC stream (for reliable service), plus one for QUIC Datagrams (unreliable and notified services), thus we usually have multiple Transfer flows in each direction, each of which has its own states. In Figure 8 and Figure 9, is shown the diagram of states in the case of reliable transfer, on the sender and receiver sides, respectively.

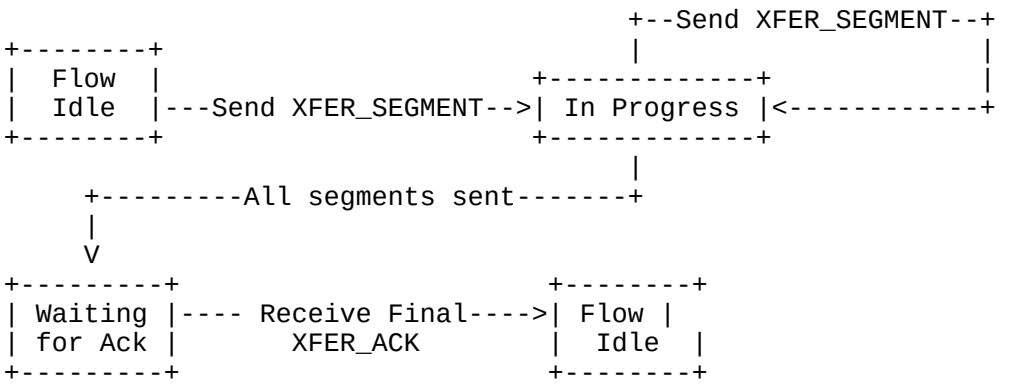

Figure 8: Transfer Sender States

Note on transfer sending: pipelining of transfers (on the same Transfer flow) can occur when the Transfer flow is in the "Waiting for Ack" state.

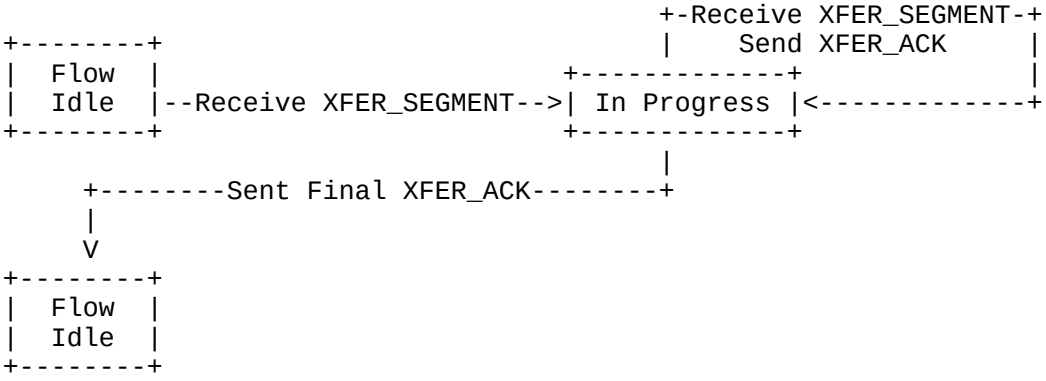

Figure 9: Transfer Receiver States

The two figures above are valid for notified transfers too; in the unreliable case, vice versa, there is a little simplification in the diagram of states, as there are not XFER_ACKs and thus the Transfer flow returns to the idle state as soon as the last XFER_SEGMENT of the transfer is either sent (on sender side) or received (on receiver side).

A QUIC connection is Idle when all its trasnsfer flows are Idle.

3.3.3. Session Termination State Diagrams

Session termination involves one entity initiating the termination of the session and the other entity acknowledging the termination. For either entity, it is the sending of the SESS_TERM message, which transitions the session to the Ending substate. While a session is in the Ending state, only in-progress transfers can be completed and no new transfers can be started.

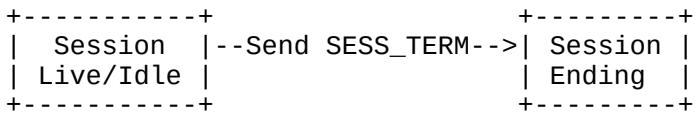

Figure 10: Session Termination (ST) from the Initiator

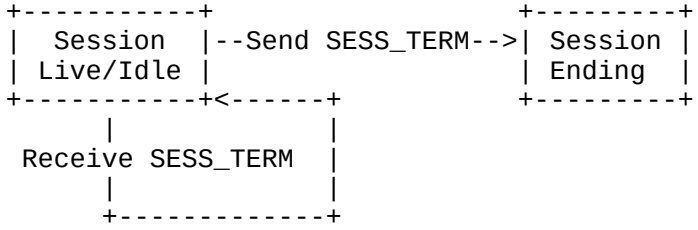

Figure 11: Session Termination (ST) from the Responder

3.4. Session-Keeping Policies

This specification defines requirements regarding how to initiate, sustain, and terminate a QUICCL session but does not impose any requirements on how sessions need to be managed by a BPA. It is a network administration matter to determine an appropriate session-keeping policy. Guidance given in [RFC9174] for TCPCL is still valid for QUICCL; the interested reader is referred to this document for further information.

3.5. Transfer Segmentation Policies

Each QUICCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. Segmentation for the reliable service based on QUIC streams necessarily differs from segmentation for services based on QUIC datagrams, as in this latter case the segment dimension (plus overhead) cannot exceed the maximum payload size of an UDP packet.

3.5.1. Reliable Service

A receiving entity can set the Segment Maximum Receive Unit (MRU) in its SESS_INIT message to determine the largest acceptable segment size for the reliable service, and a transmitting entity can segment a reliable transfer into any sizes smaller than the receiver’s Segment MRU. It is a network administration matter to determine an appropriate segmentation policy. Guidance given in [RFC9174] for TCPCLv4 is still valid for the QUICCL reliable service, which is based on streams; the interested reader is referred to this document.

3.5.2. Unreliable and Notified Services

A receiving entity can set the Datagram Maximum Receive Unit (MRU) in its SESS_INIT message to determine the largest acceptable segment size for the unreliable and notified services, which are based on QUIC datagrams, and a transmitting entity can segment an unreliable or notified transfer into any sizes smaller than the minimum between the receiver’s Datagram MRU and the local minimum. The latter can be derived from information provided by the QUIC implementation; when not available, it can be set by the user. The local minimum cannot theoretically exceed the maximum payload of an UDP packet (a little less than 2^16 byte), as a segment must fit into one UDP datagram; however, this limit is generally much more stringent if we want to avoid IP fragmentation; in this latter case, it must be a little smaller than the maximum transfer unit encountered between the two QUICCL peers (a margin is necessary to accomodate UDP and IP headers).

3.6. Example (reliable service)

To complete the general escription of the protocol, an example is in order. Figure 12 depicts the protocol exchange for a simple session, showing the session establishment and the transmission of a single bundle with the reliable service. The bundle is split into three data segments (of lengths "L1", "L2)", and "L3") from Entity A to Entity B. One bundle transfer is always performed by using one Transfer flow, i.e. one stream when in a relaible way, as here. This means that if the bundle is large enough to require multiple

XFER_SEGMENTS, each segment must be transferred on the same QUIC stream. Different bundles may be sent either sequentially on the same stream, or in parallel on multiple streams.

Note that the sending entity can transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on one stream. Although this example only demonstrates a single bundle transmission, it is also possible to pipeline multiple XFER_SEGMENT messages for different bundles without necessarily waiting for XFER_ACK messages to be returned for each one. However, interleaving data segments from different bundles on the same stream is never allowed. No errors or rejections are shown in this example.

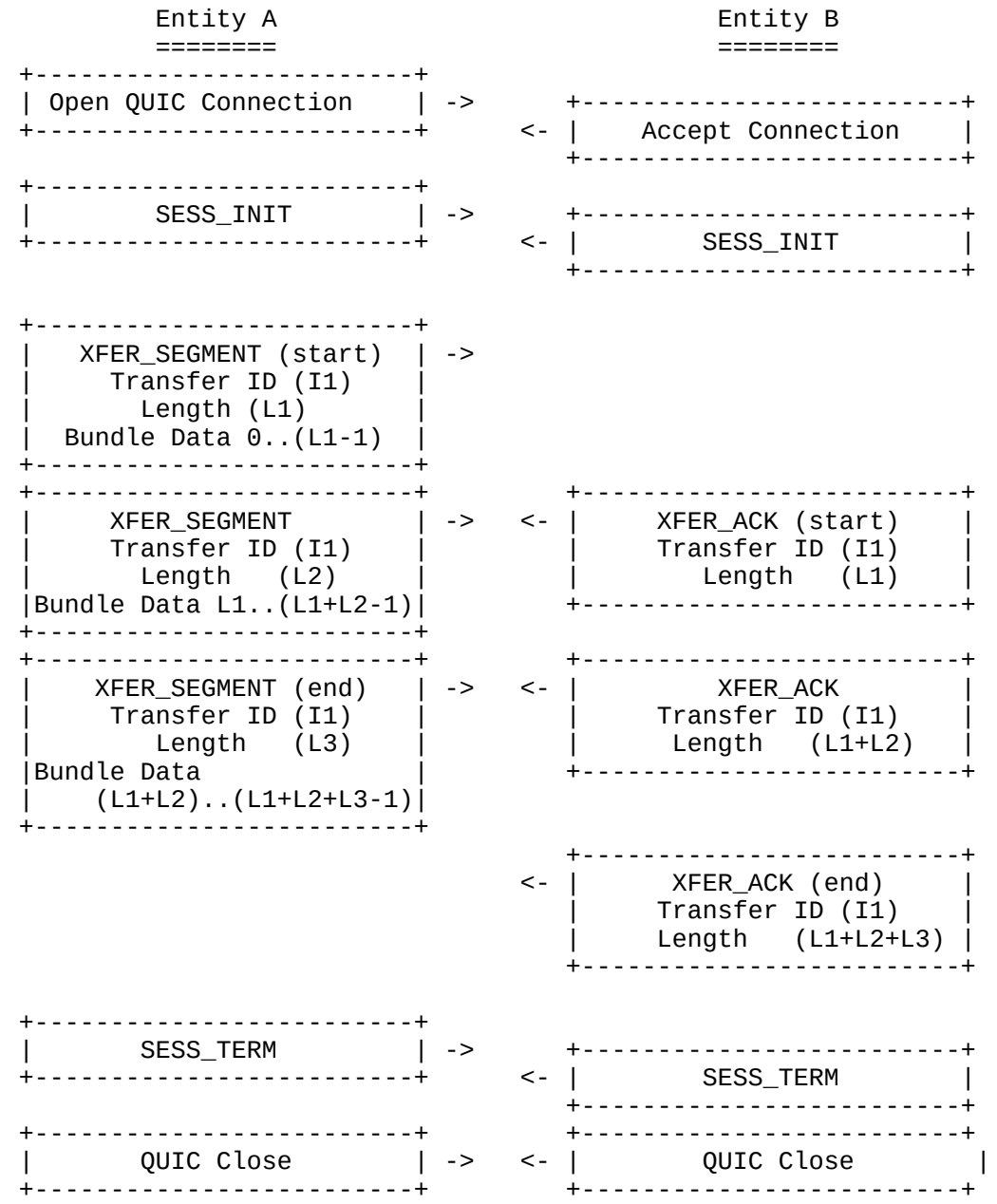

Figure 12: An Example of the exchange of Protocol Messages on a Single QUIC connection between Two Entities

The general protcol description given so far will be complemented by details and message formats in the next main section.

4. Detailed Protocol Procedures and Message Types

4.1. QUIC Connection Establishment notes

As QUIC is a transport protocol that makes use of UDP (essentially to bypass NATs), QUICCL must be associated to a UDP port. IANA is requested to assign the UDP port number 4560 to QUICCL, see Section 6.1, and the use of this port number as a default for a QUICCL passive entity is RECOMMENDED while waiting for the official IANA assignment. Other destination port numbers MAY be used per local configuration. Determining a peer’s destination port number (if different from the default QUICCL port number) is left up to the implementation. Any source port number MAY be used for QUICCL sessions. Typically, an operating system assigned number in the UDP ephemeral range (49152-65535) is used.

The TLS Application-Layer Protocol Negotiation identifier which MUST be used by the QUICCL peers when opening a QUIC session is "quicclav1".

A node may choose to either initiate a new session for each bundle transfer or keep a session open for as long as possible, using it when necessary. In the former case, it is convenient to make use of QUIC 0-RTT feature [RFC9000] [RFC9308], if available.

If the entity is unable to establish a QUIC connection for any reason, then it is an implementation matter to determine how to handle the connection failure. An entity MAY decide to reattempt to establish the connection. If it does so, it MUST NOT overwhelm its target with repeated connection attempts. Therefore, the entity MUST NOT retry the connection setup earlier than some delay time from the last attempt, and it SHOULD use a (binary) exponential backoff mechanism to increase this delay in the case of repeated failures. The upper limit on a reattempt backoff is implementation defined but SHOULD be no longer than one minute (60 seconds) before signaling to the BPA that a connection cannot be made.

4.2. QUIC Stream Types in QUICCL

QUIC streams are not negotiated when a QUIC connection is established, but are simply declared by a QUIC engine when data are sent. Streams can be of four types, depending whether they are unidirectional or bidirectional, and client or server-initiated. Unidirectional streams carry data in one direction, from the initiator of the stream to its peer; bidirectional streams allow for

data to be sent in both directions. Each stream is identified by an integer number, the stream ID, which also identify its type. For bidirectional streams, the only used by QUICCL, we have 0, 4, 8, 12, 16, etc. for the streams initiated by the client, and 1, 5, 9, 13, 17, etc. for those initiated by the server.

In QUICCL the stream 0 is reserved to the transmission of signalling QUICCL messages that are not associated to a Transfer flow, and it is the only stream used by QUICCL in a logical bidirectional way, i.e. by both the QUICCL active peer (the QUIC client) and passive peer (the QUIC server). The QUICCL active peer by default will use streams 4, 8, 12 and 16 to send XFER_SEGMENTs corresponding to bundles of expedited, normal, bulk and no priority; the return direction of these streams will be used only for corresponding XFER_ACKs. In a dual way, the passive peer will use streams 1, 5, 9, 13, to send data in the opposite direction.

These streams SHOULD be associated to decreasing QUIC priorities, in order to offer the fastest transmission to signalling messages on stream 0 and then to expedited, normal, bulk and no priority XFER_SEGMENTS. The way by which priority are associated to QUIC streams is left by [RFC9000] to the QUIC implementation. In the presence of many levels of QUIC priorities (e.g. 256), as in the Picoquic implementation used by Unibo-BP, the full priority range SHOULD be used to ensure a significant interval between QUIC priorities associated to the four streams.

A QUICCL implementation SHOULD allow the user to override the default priority mapping, e.g. by selecting the use of N streams without priorities. In the particular case of one stream only, i.e. N=1, the streams to be used should be 4 and 1 for the client and the server, respectively. Analogously for other values of N. When QUIC streams are not associated to bundle priorities, they SHOULD have the same priority, but stream 0, which MAY have a higher priority.

4.3. Message Types

Once the QUIC connection is established, the two QUICCL peers exchange QUICCL messages of various kind, most of which have been anticipated above. All messages have a header whose format is very simple, as it consists of a unique field, the message type, shown in Figure 13

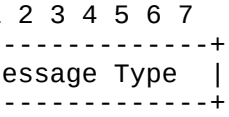

Figure 13: Format of the Message Header

Message types codes are shown in Table 2.

Code	Name	Description
0x00	Reserved	
0x01	SESS_INIT	Contains the session parameter inputs from one of the entities
0x02	XFER_SEGMENT	Contains a segment of bundle data
0x03	XFER_ACK	Acknowledges a segment of bundle data
0x04	XFER_REFUSE	Refuses a bundle transfer initiated by the peer
0x05	KEEPALIVE	Used to keep the QUICCL session active
0x06	SESS_TERM	Indicates a wish to terminate the session
0x07	MSJ_REJECT	Indicates that a message sent by the peer was not expected or understood
0x08-0xEF	Unassigned	
0xF0-0xFF	Reserved for Private or Experimental Use	

Table 2

The format and use of these messages will be shown in the next sub-sections.

4.4. Session Initialization (SESS_INIT)

Once the QUIC connection is established, both peers must send a SESS_INIT message on QUIC stream 0. Since stream 0 is client-initiated according to the QUIC specification, it follows that the passive QUICCL peer must receive the active peer's SESS_INIT before sending its own. The SESS_INIT message exchange is used to determine the per-session parameters.

4.4.1. SESS_INIT format

The format of a SESS_INIT message is shown in Figure 14.

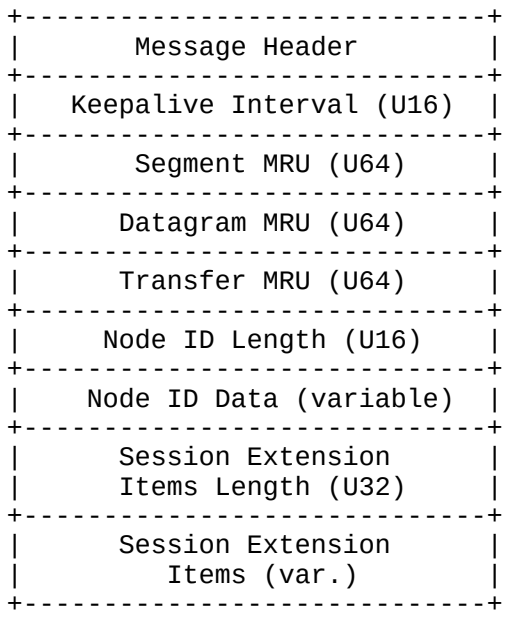

Figure 14: Format of the SESS_INIT message

The fields of the SESS_INIT message are as follows:

Keepalive Interval: A 16-bit unsigned integer indicating the minimum interval, in seconds, to negotiate as the Session Keepalive using the method described in Section 4.4.2.

Segment MRU: A 64-bit unsigned integer indicating the largest

allowable single-segment data payload size to be received in this session. Any XFER_SEGMENT sent to this peer SHALL have a data payload no longer than the peer's Segment MRU. The two entities of a single session MAY have different Segment MRUs, and no relationship between the two is required.

Datagram MRU: As the Segment MRU, but it applies to services based on QUIC datagrams.

Transfer MRU: A 64-bit unsigned integer indicating the largest allowable total-bundle data size to be received in this session. Any bundle transfer sent to this peer SHALL have a Total Bundle Length payload no longer than the peer's Transfer MRU. This value MAY be used to perform proactive bundle fragmentation. The two entities of a single session MAY have different Transfer MRUs, and no relationship between the two is required.

Node ID Length and Node ID Data: Together, these fields represent a variable-length text string. The Node ID Length is a 16-bit unsigned integer indicating the number of octets of Node ID Data to follow. A zero-length node ID SHALL be used to indicate the lack of a node ID rather than a truly empty node ID. This case allows an entity to avoid exposing node ID information on an untrusted network. A non-zero-length Node ID Data SHALL contain the UTF-8 encoded node ID of the entity that sent the SESS_INIT message. Every node ID SHALL be a URI consistent with the requirements in [RFC3986] and the URI schemes of the IANA "Bundle Protocol URI Scheme Types" registry [IANA-BUNDLE].

Session Extension Items Length and Session Extension Items list: Together, these fields represent protocol extension data not defined by this specification. The Session Extension Items Length is the total number of octets to follow that are used to encode the Session Extension Items list. The encoding of each Session Extension Item is within a consistent data container as described in Section 4.4.4. The full set of Session Extension Items apply for the duration of the QUICCL session to follow. The order and multiplicity of these Session Extension Items are significant, as defined in the associated type specification(s). If the content of the Session Extension Items list disagrees with the Session Extension Items Length (e.g., the last item claims to use more or fewer octets than are indicated in the Session Extension Items Length), the reception of the SESS_INIT is considered to have failed.

4.4.2. Session Parameter Negotiation

An entity calculates the parameters for a QUICCL session by negotiating the values from its own preferences (conveyed by the SESS_INIT it sent to the peer) with the preferences of the peer entity (expressed in the SESS_INIT that it received from the peer). The negotiated parameters defined by this specification are described in the following paragraphs.

Transfer MTU: The Maximum Transmission Unit (MTU) for whole transfers, is identical to the Transfer MRU advertised by the received SESS_INIT message.

Segment MTU: The same as the Transfer MTU, but it applies to XFER_SEGMENTS sent in a reliable way, i.e. on streams. A transmitting peer can send individual segments with any size smaller than the Segment MTU, or individual datagrams smaller than the Datagram MTU, depending on local policy, dynamic network conditions, etc. Determining the size of each transmitted segment is an implementation matter. I

Datagaram Segment MTU (Datagram MTU in the following): The same as the Trasnfer MTU, but it applies to XFER_SEGMENTS sent in a a notified or unreliable way, i.e. by means of QUIC datagrams.

If either the Transfer MRU, or the Segment MRU, or the Datagram MRU are unacceptable, the entity SHALL terminate the session with a reason code of "Init Failure".

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of the two Keepalive Interval values from the two SESS_INIT messages. The Session Keepalive Interval is a parameter for the behavior described in Section 4.8. If the Session Keepalive Interval is unacceptable, the entity SHALL terminate the session with a reason code of "Init Failure". Note: A negotiated Session Keepalive of zero indicates that KEEPALIVEs are disabled.

Once this process of parameter negotiation is completed, this protocol defines no additional mechanism to change the parameters of an established session; to effect such a change, the QUICCL session MUST be terminated and a new session established.

4.4.3. Peer Node ID Considerations

When the active entity initiates a QUICCL session, it is likely based on routing information that binds a node ID to CL parameters used to initiate the session. If the active entity receives a SESS_INIT with a different node ID than was intended for the QUICCL session, the session MAY be established. If this is the case, such a session SHALL be associated with the node ID provided in the SESS_INIT message rather than any intended value.

Note that BPA may, and often have, multiple node EIDs, one for each "scheme" (i.e. either "dtn" or "ipn"). In the presence of multiple node IDs, Which ID should be advertised by the SESS_INIT message is an implementation or configuration matter, but the network manager should be aware that if the scheme choice is not consistent with what expected by adjacent nodes, a mismatch results. This problem does not apply in space networks, where the "ipn" scheme is the only allowed.

4.4.4. Session Extension Item Format

Each of the extension Items SHALL be encoded in an identical Type-Length-Value (TLV) container form, as indicated in Figure 15 .

The fields of the Session Extension Item are as follows:

Item Flags: A one-octet field containing generic bit flags related to the Item, which are listed in Table 3. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver. If a QUICCL entity receives a Session Extension Item with an unknown Item Type and the CRITICAL flag set to 1, the entity SHALL terminate the QUICCL session with a SESS_TERM reason code of "Init Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification does not define any extension types directly but does create an IANA registry for such codes (see Section 8.3).

Item Length: A 16-bit unsigned integer field containing the number of Item Value octets to follow.

Item Value: A variable-length data field that is interpreted according to the associated Item Type. This specification places no restrictions on an extension’s use of available Item Value data.

Extension specifications SHOULD avoid the use of large data lengths, as no bundle transfers can begin until the full extension data is sent.

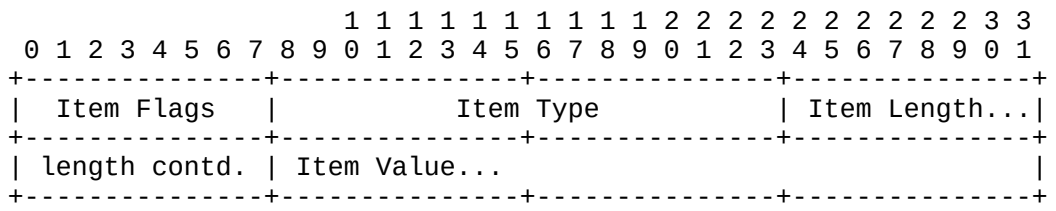

Figure 15: Format of one extension item

Code	Name	Description
0x01	CRITICAL	If set, it indicates that the receiving peer must handle the extension item.
Others	Reserved	

Table 3

4.5. Bundle Transfer (XFER_SEGMENT)

A single QUICCL transfer consists in the transmission of a bundle (handled by the convergence layer as opaque data) between two QUICCL entities. Bundle data is transmitted in one or more segments, each carried by a XFER_SEGMENT message.

In the QUICCL reliable service, based on QUIC streams, a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU, which is defined in Section 4.4.1. The choice of the length to use for segments is an implementation matter, but each segment MUST NOT be larger than the receiving entity’s Segment MRU. The first segment for a bundle is indicated by the START flag, and the last segment is indicated by the END flag.

In unreliable and notified services, based on QUIC datagrams, a transfer is accomplished exactly as before, with the only notable difference that segments now must fit into QUIC Datagrams, thus their dimension is based on the receiving-side Datagram MRU, instead of Segment MRU.

A single transfer (and, by extension, a single segment) SHALL NOT contain data of more than a single bundle.

4.5.1. XFER_SEGMENT Format

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message is shown in Figure 16 . It is the same for the reliable service, based on QUIC streams, and the unreliable and notified services, based on QUIC datagrams.

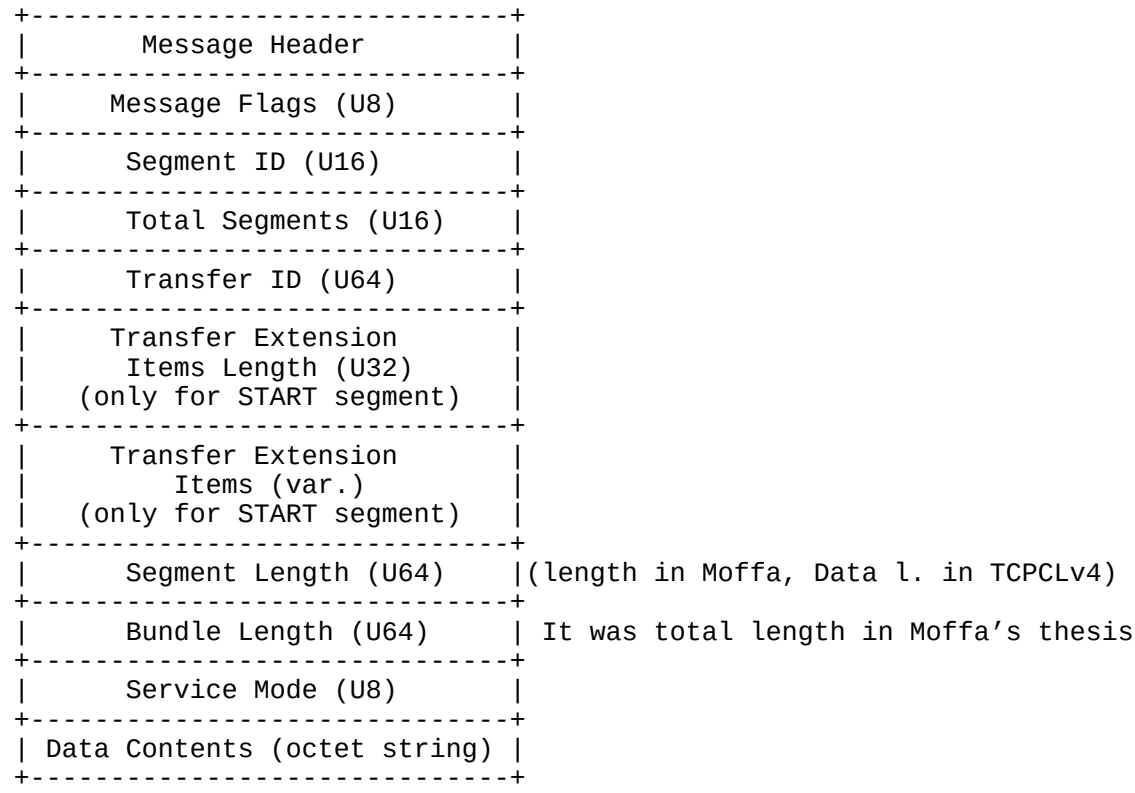

Figure 16: Format of XFER_SEGMENT messages

With respect to TCPCLv4 messages, we have a few new fields:

Segment ID: contains the id of the segment being sent.

Total Segments: contains the total number of segments being sent.

Bundle Length: contains the entire length of the bundle (i.e. the sum of all the lengths of all segments). In TCPCLv4 this field, called total length, was an extension item; in QUICCL it is a mandatory field.

Service Mode: It denotes the service used to send the bundle (0 for reliable, 1 for notified, 2 for unreliable); it is used by the QUICCL receiving peer to select the appropriate acknowledgment mode.

The other fields of the XFER_SEGMENT message are the same as in TCPCLv4. They are as follows:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 4. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Transfer ID: A 64-bit unsigned integer identifying the transfer being made.

Transfer Extension Items Length and Transfer Extension Items list: Together, these fields represent protocol extension data for this specification. The Transfer Extension Items Length and Transfer

Transfer Extension Items list SHALL only be present when the START flag is set to 1 on the message. The Transfer Extension Items Length is the total number of octets to follow that are used to encode the Transfer Extension Items list. The encoding of each Transfer Extension Item is within a consistent data container, as described in Section 4.5.4 . The full set of Transfer Extension Items apply only to the associated single transfer. The order and multiplicity of these Transfer Extension Items are significant, as defined in the associated type specification(s). If the content of the Transfer Extension Items list disagrees with the Transfer Extension Items Length (e.g., the last item claims to use more or fewer octets than are indicated in the Transfer Extension Items Length), the reception of the XFER_SEGMENT is considered to have failed.

Data Length: A 64-bit unsigned integer indicating the number of octets in Data contents to follow.

Data Contents: The variable-length payload of the message.

Code	Name	Description
0x01	END	If set, it indicates that this is the last segment of the transfer
0x02	START	If set, it indicates that this is the first segment of the transfer
Others	Reserved	

Table 4

4.5.2. START and END Flags Notes

The flags portion of the message contains two flag values in the two low-order bits, denoted START and END in Table 4. The START flag SHALL be set to 1 when transmitting the first segment of a transfer. The END flag SHALL be set to 1 when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the START flag and the END flag SHALL be set to 1.

4.5.3. Transfer ID Notes

The Transfer ID field is used to associate messages to the bundle being tranmitted. Each invocation of the QUICCL by the BPA, requesting transmission of a bundle (fragmentary or otherwise), results in the initiation of a single QUICCL transfer. Each transfer entails the sending of a sequence of some number of XFER_SEGMENT and XFER_ACK messages; all are correlated by the same Transfer ID. The sending entity originates a Transfer ID, and the receiving entity uses that same Transfer ID in acknowledgments.

Transfer IDs from each entity SHALL be unique within a single QUICCL session. Upon exhaustion of the entire 64-bit Transfer ID space, the sending entity SHALL terminate the session with a SESS_TERM reason code of "Resource Exhaustion". For bidirectional bundle transfers, a QUICCL entity SHOULD NOT rely on any relationship between Transfer IDs originating from each side of the QUICCL session. Although there is not a strict requirement for initial Transfer ID values or the ordering of Transfer IDs, in the absence of any other mechanism for generating Transfer IDs, an entity SHALL use the following algorithm: the initial Transfer ID from each entity is zero, and subsequent Transfer ID values are incremented from the prior Transfer ID value by one.

Once a transfer of a bundle has commenced on a Transfer flow, i.e. on a stream or as QUIC datagrams, the entity MUST only send segments containing sequential portions of that bundle on the same Transfer flow, until it sends a segment with the END flag set to 1. No interleaving of multiple transfers from the same entity is possible within the same Transfer flow. However, simultaneous transfers can be achieved by using multiple Transfer flows in parallel. This usually happens when bundles of different priorities are sent on different streams, as by default in QUICCL.

4.5.4. Transfer Extension Items Format

Transfer Extension Items have the same format of Session Extension Items, including the CRITICAL flag. If a QUICCL entity receives a Transfer Extension Item with an unknown Item Type and the CRITICAL flag is 1, the entity SHALL refuse the transfer with a XFER_REFUSE reason code of "Extension Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Note that the TCPCLv4 Transfer length extension is no more present here, as it has become the "Bundle Length" compulsory field of the XFER_SEGMENT.

4.6. Segment Acknowledgment (XFER_ACK)

The QUICCL protocol usually acknowledges reception of data segments, as TCPCLv4 does. The notable exception is when data segments are sent by using the unreliable service. XFER_ACKs are sent back on the same stream used by the corresponding XFER_SEGMENT in the reliable service, or as Datagarams, in the notified service.

4.6.1. XFER_ACK Format

The format of a XFER_ACK message is shown in Figure 17 .

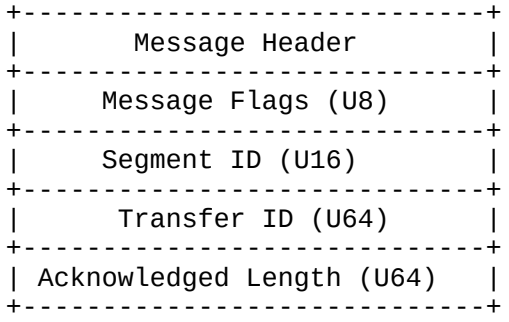

Figure 17: Format of XFER_ACK Messages

The only field not already present in TCPCLv4 is the following:

Segment ID: It contains the segment identifier of the XFER_SEGMENT being acknowledged. The insertion of this new field is necessary to support the services based on QUIC datagrams, which are sent by QUIC best effort.

The other fields of the XFER_ACK message are the same as in TCPCLv4, and are as follows:

Message Flags: A one-octet field of single-bit flags, whose meaning is the same as XFER_SEGMENT Message Flags, see Table 4. All reserved flag bits SHALL be set to 0 by the sender. All reserved flag bits SHALL be ignored by the receiver.

Transfer ID: A 64-bit unsigned integer identifying the transfer being acknowledged.

Acknowledged Length: A 64-bit unsigned integer, indicating the total number of octets in the transfer that are being acknowledged (reliable service).

4.6.2. Additional Considerations

The response to a XFER_SEGMENT reception depends on the transport service declared by the new Service ID field.

If unreliable, no XFER_ACK is sent but the Segment ID of the XFER_SEGMENT is recorded, to keep track of XFER_SEGMENTS received from the transfer start. If all segments are recieved in the due time, the bundle is reassembled and delivered to the BPA. Otherwise, an inter-segment timer expires and all segments of the incomplete transfer are dropped. The setting of the timer threshold is part of QUICCL configuration.

If the transport service is reliable or notified, a XFER_ACK MUST be issued as soon as the XFER_SEGMENT has been processed; the Message Flags SHALL be set to match the flags of the XFER_SEGMENT being acknowledged (including flags not decodable to the entity), and the same for the Segment ID. From now on, we need to distinguish between reliable and notified services.

If the service is reliable the Acknowledged Length is cumulative, i.e. it contains the sum of the Data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer (note that being sent on a QUIC stream, all XFER_SEGMENTS are received in order). The sending entity SHOULD transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream. For example, suppose the sending entity transmits four segments of bundle data with lengths 100, 200, 500, and 1000, respectively. After receiving the first segment, the entity sends an acknowledgment of length 100. After the second segment is received, the entity sends an acknowledgment of length 300. The third and fourth acknowledgments are of lengths 800 and 1800, respectively.

If the service is notified the Acknowledment Length is not cumulative, but corresponds to the amount of data delivered by the acknowledged XFER_SEGMENT. In this case the Segment ID allows the sending QUICCL peer to know which segments have been received. If all segments are confirmed in the due time, the sending BPA MUST be notified a success, otherwise a failure. In addition to the receiver side inter-segment timer, in common to the unreliable service, a second inter-segment timer is necessary at sender side, to limit the waiting time before declaring a failure to the sending BPA. The time threshold of this second timer should be set in QUICCL configuration as well.

4.7. Transfer Refusal (XFER_REFUSE)

QUICCL supports a mechanism by which a receiving entity can indicate to the sender that it does not want to receive other segments of a partiually received bundle. To do so, upon receiving a XFER_SEGMENT message, the entity MAY transmit a XFER_REFUSE message, reporting the Transfer ID of the refused bundle. The XFER_REFUSE messages must be sent back on the same stream used by the corresponding XFER_SEGMENT in the reliable service, or as Datagram, in the notified or unreliable services, as XFER_ACKs do. There is no required relationship between the Transfer MRU of a QUICCL entity (which is supposed to represent a firm limitation of what the entity will accept) and the sending of a XFER_REFUSE message. A XFER_REFUSE can be used in cases where the agent’s bundle storage is temporarily depleted or somehow constrained. A XFER_REFUSE can also be used after the bundle header or any bundle data is inspected by an agent and determined to be unacceptable.

A transfer receiver MAY send a XFER_REFUSE message as soon as it receives any XFER_SEGMENT message. The transfer sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

The QUICCL itself does not have any required behavior related to responding to a XFER_REFUSE based on its reason code; the refusal is passed up as an indication to the BPA that the transfer has been refused. If a transfer refusal has a reason code that is not decodable to the BPA, the agent SHOULD treat the refusal as having a reason code of "Unknown".

For each transfer preceding (on the same Transfer flow) the one to be refused, the reciever MUST have either acknowledged all related XFER_SEGMENT messages or refused the bundle transfer.

The bundle transfer refusal MAY be sent also before an entire data segment is received. If a sender receives a XFER_REFUSE message, the sender MUST complete the transmission of any partially sent XFER_SEGMENT message. There is no way to interrupt an individual QUICCL message partway through sending it. The sender MUST NOT subsequently commence transmission of any further segments of the refused bundle. Note, however, that this requirement does not ensure that an entity will not receive another XFER_SEGMENT for the same bundle after transmitting a XFER_REFUSE message, since messages can cross on the wire; if this happens, subsequent segments of the bundle SHALL also be refused with a XFER_REFUSE message.

Note: If a bundle transmission is aborted in this way, the receiver does not receive a segment with the END flag set to 1 for the aborted bundle. The beginning of the next bundle is identified by the START flag set to 1, indicating the start of a new transfer, and with a distinct Transfer ID value.

4.7.1. XFER_REFUSE Format

The format of the XFER_REFUSE message is shown in Figure 18.

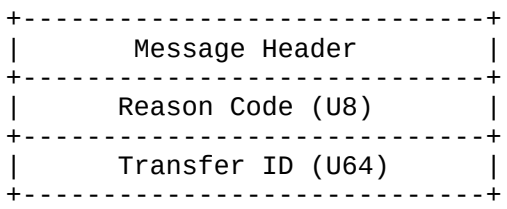

Figure 18: Format of the XFER_REFUSE message

The fields of the XFER_REFUSE message are as follows:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 5.

Transfer ID: A 64-bit unsigned integer identifying the transfer being refused.

Name	Code	Description
0x00	Unknown	The reason for refusal is unknown or is not specified
0x01	Completed	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
0x02	No resources	The receiver’s resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
0x03	Retransmit	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety
0x04	Not acceptable	Some issue with the bundle data or the transfer extension data was encountered. The sender SHOULD NOT retry the same bundle with the same extensions.
0x05	Extension failure	A failure processing the Transfer Extension Items has occurred.
0x06	Session terminating	The receiving entity is in the process of terminating the session. The sender MAY retry the same bundle at a later time in a different session.
0x07-0xEF	Unassigned	
0xF0-0xFF	Reserved for Private or Experimental Use	

Table 5

4.8. Session Upkeep (KEEPALIVE)

The protocol includes a provision for transmission of KEEPALIVE messages over the QUICCL session to help determine whether the underlying QUIC connection has been disrupted.

The Session Keepalive Interval to be used is negotiated when the session is inizialized, as detailed in Section 4.4.2. If the negotiated Session Keepalive is zero (i.e., one or both SESS_INIT messages contain a zero Keepalive Interval), then the keepalive feature is disabled. There is no logical minimum value for the Keepalive Interval (within the minimum imposed by the positive-value encoding), but when used for many sessions on an open, shared network, a short interval could lead to excessive traffic. For shared network use, a Keepalive Interval no shorter than 30 seconds is RECOMMENDED. There is no logical maximum value for the Keepalive Interval (within the maximum imposed by the fixed-size encoding), but a too long interval could result into a premature closing of the subjected QUIC connection (the actual behavior of the QUIC connection in the absence of traffic depends on QUIC implementations). Foir this reason, a Keepalive Interval no longer than 10 minutes (600 seconds) is RECOMMENDED. KEEPALIVE messages MUST be sent on stream 0.

4.9. KEEPALIVE Format

The format of a KEEPALIVE message is a one-octet Message Type code of KEEPALIVE (as described in Table 2) with no additional data. Both sides SHALL send a KEEPALIVE message whenever the negotiated interval has elapsed with no transmission of any message (KEEPALIVE or other).

If no message (KEEPALIVE or other) has been received in a session after some implementationdefined time duration, then the entity SHALL terminate the session by transmitting a SESS_TERM message (as described in Section 4.10) with a reason code of "Idle timeout". If configurable, the idle timeout duration SHOULD be no shorter than twice the Keepalive Interval. If not configurable, the idle timeout duration SHOULD be exactly twice the Keepalive Interval.

4.10. Session Termination (SESS_TERM)

To cleanly terminate a session, a SESS_TERM message SHALL be transmitted by either entity at any point following complete transmission of any other message. The REPLY flag of this SESS_TERM message SHALL be 0. Upon receiving it, the other peer SHALL respond with a SESS_TERM message with REPLY flag 1 and identical data content. SESS_TERM messages MUST be sent on stream 0.

Once a SESS_TERM message is sent, the state of that QUICCL session changes to Ending. While the session is in the Ending state,

* an entity MAY finish an in-progress transfer in either direction.

* an entity SHALL NOT begin any new outgoing transfer for the remainder of the session.

* an entity SHALL NOT accept any new incoming transfer for the remainder of the session.

If a new incoming transfer is attempted while in the Ending state, the receiving entity SHALL send a XFER_REFUSE with a reason code of "Session Terminating".

There are circumstances where an entity has an urgent need to close a QUIC connection associated with a QUICCL session, without waiting for transfers to complete but also in a way that does not force timeouts to occur. In such circumstances, after transmitting a SESS_TERM message, an entity MAY perform an unclean termination by immediately closing the associated QUIC connection. When performing an unclean termination, an entity SHOULD acknowledge all received XFER_SEGMENTs with a XFER_ACK before closing the QUIC connection. Not acknowledging received segments can result in unnecessary bundle or bundle fragment retransmissions. Any delay between a request to close the QUIC connection and the actual closing of the connection (a "half-closed" state) MAY be ignored by the QUICCL entity. If the underlying QUIC connection is closed during a transmission (in either direction), the transfer SHALL be indicated to the BPA as failed.

The QUICCL itself does not have any required behavior related to responding to a SESS_TERM based on its reason code; the termination is passed up as an indication to the BPA that the session state has changed. If a termination has a reason code that is not decodable to the BPA, the agent SHOULD treat the termination as having a reason code of "Unknown".

4.10.1. SESS_TERM Format

The format of the SESS_TERM message is shown in Figure 19.

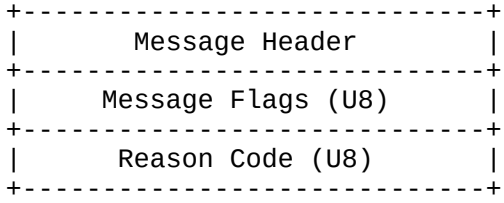

Figure 19: Format of SESS_TERM Messages

The fields of the SESS_TERM message are as follows:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 6. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 7.

Name	Code	Description
REPLY	0x01	If set, it indicates that this message is an acknowledgment of an earlier SESS_TERM message.
Reserved	Others	

Table 6

Name	Code	Description
0x00	Unknown	A termination reason is not available
0x01	Idle timeout	The session is being terminated due to idleness
0x02	Version mismatch	The entity cannot conform to the specified QUICCL version.
0x03	Busy	AThe entity is too busy to handle the current session
0x04	Init failure	The entity cannot interpret or negotiate a SESS_INIT option.
0x05	Resource exhaustion	The entity has run into some resource limit and cannot continue the session
0x06-0xEF	Unassigned	
0xF0-0xFF	Reserved for Private or Experimental Use	

Table 7

4.10.2. Additional Considerations

The protocol includes a provision for clean termination of idle sessions. Determining the length of time to wait before terminating idle sessions, if they are to be terminated at all, is an implementation and configuration matter.

If there is a configured time to terminate idle sessions and if no QUICCL messages (other than KEEPALIVE messages) have been received for at least that amount of time, then either entity MAY terminate the session by transmitting a SESS_TERM message with a reason code of "Idle timeout" (as described in Table 7).

4.11. Message Rejection (MSG_REJECT)

The purpose of this message type is to let the peer know why the current entity is not responding as expected to its messages. The MSG_REJECT messages must be sent back on the same stream used by the corresponding message, in the reliable service, or as Datagram, in the notified or unreliable services.

If a QUICCL entity receives a message type that is unknown to it (possibly due to an unhandled protocol version mismatch or an incorrectly negotiated session extension that defines a new message type), the entity SHALL send a MSG_REJECT message with a reason code of "Message Type Unknown" and close the QUIC connection. If a QUICCL entity receives a message type that is known but is inappropriate for the negotiated session parameters (possibly due to an incorrectly negotiated session extension), the entity SHALL send a MSG_REJECT message with a reason code of "Message Unsupported". If a QUICCL entity receives a message that is inappropriate for the current session state (e.g., a SESS_INIT after the session has already been established or a XFER_ACK message with an unknown Transfer ID), the entity SHALL send a MSG_REJECT message with a reason code of "Message Unexpected".

4.11.1. MSG_REJECT Format

The format of a MSG_REJECT message is shown in Figure 20 .

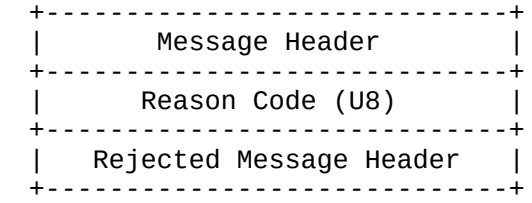

Figure 20: Format of the MSG_REJECT message

The fields of the MSG_REJECT message are as follows:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 8.

Rejected Message Header: The Rejected Message Header is a copy of the Message Header to which the MSG_REJECT message is sent as a response.

Name	Code	Description
0x01	Unknown	A message was received with a Message Type code unknown to the QUICCL entity
0x02	Unsupported	A message was received, but the QUICCL entity cannot comply with the message content
0x03	Unexpected	A message was received while the session is in a state in which the message is not expected.
0x04-0xEF	Unassigned	
0xF0-0xFF	Reserved for Private or Experimental Use	

Table 8

5. Security Considerations

This section separates security considerations into threat categories based on guidance provided in BCP 72 [RFC3552].

5.1. Threat: Passive Leak of Bundle Data

The QUICCL can be used to provide point-to-point transport security, but it does not provide security of data at rest and does not guarantee end-to-end bundle security. The bundle security mechanisms defined in [RFC9172] are to be used instead.

5.2. Threat: Denial of Service

The behaviors described in this section all amount to a potential denial of service to a quicCL entity. The denial of service could be limited to an individual QUICCL session, could affect other well-behaved sessions on an entity, or could affect all sessions on a host.

A malicious entity can trigger timeouts by continually establishing QUICCL sessions and delaying the sending of protocol-required data. The victim entity can block QUIC connections from network peers that are thought to behave incorrectly within the QUICCL.

An entity can send a large amount of data over a QUICCL session, requiring the receiving entity to handle the data. The victim entity can attempt to stop the flood of data by sending a XFER_REFUSE message or can forcibly terminate the session.

A "data dribble" attack is also possible, in which an entity presents a very small Segment MRU that causes transfers to be split among a large number of very small segments and causes the resultant segmentation overhead to overwhelm the actual bundle data segments. Similarly, an entity can present a very small Transfer MRU that will cause resources to be wasted on establishment and upkeep of a QUICCL session over which a bundle could never be transferred. The victim entity can terminate the session during parameter negotiation (Section 4.7) if the MRUs are unacceptable. An abusive entity could cause the keepalive mechanism to waste throughput within a network link that would otherwise be usable for bundle transmissions. Due to the quantization of the Keepalive Interval parameter, the smallest Session Keepalive is one second, which should be long enough to not flood the link. The victim entity can terminate the session during parameter negotiation (Section 4.7) if the Keepalive Interval is unacceptable.

Finally, an attacker or a misconfigured entity can cause issues at the QUIC connection that will cause unnecessary QUIC retransmissions or connection resets, effectively denying the use of the overlying QUICCL session.

6. IANA Considerations

Registration procedures referred to in this section (e.g., the RFC Required policy) are defined in [RFC8126].

All registries are specific for QUICCLv1, although some of these registries reuse some or all codepoints from TCPCLv4 for commonality. Differences will be highlighted.

6.1. Port Number

Within the "Service Name and Transport Protocol Port Number Registry" [IANA-PORTS], IANA is requested to assign UDP port number 4560 to QUICCL as its default port.

Parameter	Value
Service Name:	dtn-bundle
Transport Protocol(s):	UDP
Assignee:	IESG (iesg@ietf.org)
Contact:	IESG (iesg@ietf.org)
Description:	DTN Bundle QUIC CL Protocol
Reference:	This specification
Port Number:	4560

Table 9: UDP Port Number for QUICCL

6.2. Protocol Versions

IANA is requested to create a new registry, called "Bundle Protocol QUIC Convergence-Layer Version Numbers", and register the following value in it

Value	Description	Reference
1	QUICCLv1	This specification

Table 10: QUICCL Version Number

Note that QUIC makes use of TLS Application-Layer Protocol Negotiation extension (usually shortened to ALPN) [RFC7301], thus even in the presence of future QUICCL versions, there will be no ambiguity regarding which version is being used.

6.3. Session Extension Types

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA is requested to create the "Bundle Protocol QUIC Convergence-Layer Version 1 Session Extension Types" registry and populate it with the contents of Table 11. The registration procedure is Expert Review within the lower range 0x0001-0x7FFF. Values in the range 0x8000-0xFFFF are reserved for Private or Experimental Use, which are not recorded by IANA.

Specifications of new session extension types need to define the encoding of the Item Value data as well as any meaning or restriction on the number of or order of instances of the type within an extension item list. Specifications need to define how the extension functions when no instance of the new extension type is received during session negotiation.

Experts are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely esthetically displeasing or architecturally dubious).

Code	Session Extension Type
0x0000	Reserved
0x0001-0x7FFF	Unassigned
0x8000-0xFFFF	Reserved for Private or Experimental Use

Table 11: Session Extension Type Codes

6.4. Transfer Extension Types

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA is requested to create the "Bundle Protocol QUIC Convergence-Layer Version 1 Transfer Extension Types" registry and populate it with the contents of Table 12. The registration procedure is Expert Review within the lower range 0x0001-0x7FFF. Values in the range 0x8000-0xFFFF are reserved for Private or Experimental Use, which are not recorded by IANA.

Specifications of new transfer extension types need to define the encoding of the Item Value data as well as any meaning or restriction on the number of or order of instances of the type within an extension item list. Specifications need to define how the extension functions when no instance of the new extension type is received in a transfer.

Experts are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely esthetically displeasing or architecturally dubious).

Code	Transfer Extension Type
0x0000	Reserved
0x0001-0x7FFF	Unassigned
0x8000-0xFFFF	Reserved for Private or Experimental Use

Table 12: Transfer Extension Type Codes

6.5. Message Types

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA is requested to create the Bundle Protocol QUIC Convergence-Layer Version 1 Message Types" registry and populate it with the contents of Table 2. The registration procedure is RFC Required within the lower range 0x01-0xEF. Values in the range 0xF0-0xFF are reserved for Private or Experimental Use, which are not recorded by IANA.

Specifications of new message types need to define the encoding of the message data as well as the purpose and relationship of the new message to existing session/transfer state within the baseline message sequencing. The use of new message types needs to be negotiated between QUICCL entities within a session (using the session extension mechanism) so that the receiving entity can properly decode all message types used in the session.

Experts are encouraged to favor new session/transfer extension types over new message types. QUICCL messages are not self-delimiting, so care must be taken in introducing new message types. If an entity receives an unknown message type, the only thing that can be done is to send a MSG_REJECT and close the QUIC connection; not even a clean termination can be done at that point.

6.6. XFER_REFUSE Reason Codes

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA is requested to create the Bundle Protocol QUIC Convergence-Layer Version 1 XFER_REFUSE Reason Codes" registry and populate it with the contents of Table 5. The registration procedure is Specification Required within the lower range 0x00-0xEF. Values in the range 0xF0-0xFF are reserved for Private or Experimental Use, which are not recorded by IANA.

Specifications of new XFER_REFUSE reason codes need to define the meaning of the reason and disambiguate it from preexisting reasons. Each refusal reason needs to be usable by the receiving BPA to make retransmission or rerouting decisions.

Experts are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely esthetically displeasing or architecturally dubious).

6.7. SESS_TERM Reason Codes

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA is requested to create the "Bundle Protocol QUIC Convergence-Layer Version 1 SESS_TERM Reason Codes" registry and populate it with the contents of Table 7. The registration procedure is Specification Required within the lower range 0x00-0xEF. Values in the range 0xF0-0xFF are reserved for Private or Experimental Use, which are not recorded by IANA.

Specifications of new SESS_TERM reason codes need to define the meaning of the reason and disambiguate it from preexisting reasons. Each termination reason needs to be usable by the receiving BPA to make reconnection decisions.

Experts are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely esthetically displeasing or architecturally dubious).

6.8. MSG_REJECT Reason Codes

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA is requested to create the "Bundle Protocol QUIC Convergence-Layer Version 1 MSG_REJECT Reason Codes" registry and populate it with the contents of Table 8. The registration procedure is Specification Required within the lower range 0x01-0xEF. Values in the range 0xF0-0xFF are reserved for Private or Experimental Use, which are not recorded by IANA.

Specifications of MSG_REJECT reason codes need to define the meaning of the reason and disambiguate it from preexisting reasons. Each rejection reason needs to be usable by the receiving QUICCL entity to make message sequencing and/or session termination decisions.

Experts are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely esthetically displeasing or architecturally dubious).

7.

References

7.1. Normative References

[IANA-BUNDLE]
IANA, "Bundle Protocol", <https://www.iana.org/assignments/bundle/>.

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

[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/info/rfc2119>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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/info/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/info/rfc9000>.

[RFC9171] Burleigh, S., Fall, K., and E. Birrane, III, "Bundle Protocol Version 7", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

[RFC9172] Birrane, III, E. and K. McKeever, "Bundle Protocol Security (BPSec)", RFC 9172, DOI 10.17487/RFC9172, January 2022, <https://www.rfc-editor.org/info/rfc9172>.

[RFC9174] Sipos, B., Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol Version 4", RFC 9174, DOI 10.17487/RFC9174, January 2022, <https://www.rfc-editor.org/info/rfc9174>.

[RFC9221] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/info/rfc9221>.

7.2. Informative References

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <https://www.rfc-editor.org/info/rfc5050>.

[RFC5326] Ramadas, M., Burleigh, S., and S. Farrell, "Licklider Transmission Protocol - Specification", RFC 5326, DOI 10.17487/RFC5326, September 2008, <https://www.rfc-editor.org/info/rfc5326>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC9308] K端hlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", RFC 9308, DOI 10.17487/RFC9308, September 2022, <https://www.rfc-editor.org/info/rfc9308>.

[I-D.burleigh-dtn-ecos] Burleigh, S. and F. Templin, "Bundle Protocol Extended Class of Service (ECOS)", Work in Progress, Internet-Draft, draft-burleigh-dtn-ecos-00, 5 May 2021, <https://datatracker.ietf.org/doc/html/draft-burleigh-dtn-ecos-00>.

[Bisacchi_2022]
Bisacchi, A., Caini, C., and T. de Cola, "Multicolor Licklider Transmission Protocol: An LTP Version for Future Interplanetary Links", IEEE Transactions on Aerospace and Electronic Systems, vol. 58, no. 5, pp. 3859-3869, DOI 10.1109/TAES.2022.3176847, October 2022, <https://doi.org/10.1109/TAES.2022.3176847>.

[Burleigh_2007]
Burleigh, S., "Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol", IEEE Consumer Commun. and Networking Conference, 2007, pp. 222-226, DOI 10.1109/CCNC.2007.51, January 2007, <https://doi.org/10.1109/CCNC.2007.51>.

[Caini_2024]
Caini, C. and L. Persampieri, "Design and Features of Unibo-BP, the Unibo Implementation of the DTN Bundle Protocol", IEEE Journal of Radio Frequency Identification, vol. 8, pp. 458-467, DOI 10.1109/JRFID.2024.3358012, January 2024, <https://doi.org/10.1109/JRFID.2024.3358012>.

Appendix A. Appendix A. Significant Changes from RFC 9174.

The areas in which changes from [RFC9174] have been made to existing headers and messages are as follows:

* Removed any reference to TLS security, as this is now included in QUIC.

* Added the unreliable and notified services to the reliable service; the former two are based on QUIC datagrams, the latter on QUIC streams.

* Added priority support in the reliable service, based on QUIC streams.

* Removed contact headers (no more necessary because TLS is integrated in QUIC).

* Added inter-segment timers at sender and receiver side, to support the notified and the unreliable services.

* Message type codes redefined to follow the order in which they usually appear during a session.

* Added the field Datagram MRU to the SESS_INIT message. It is the same as the Segment MRU, but it applies to services based on QUIC datagrams.

* Added the Segment ID, the Total Segments and the Service Mode fields to the XFER_SEGMENT structure to support the unreliable and notified services.

* The Total length Transfer Extension has become the Bundle Length (compulsory) field of the XFER_SEGMENT, to simplify the protocol.

* Added the field Segment ID to the XFER_ACK structure, necessary to support the notified service.

* Use of UDP port 4560 requested to IANA for QUICCL listening.

* Defined a few new IANA registries.

* Renamed (TCPCL) STREAMS to (QUICCL) FLOWS to reserve the term "stream" to QUIC connections.

Appendix B. Acknowledgments

This work is largely inspired by [RFC9174] (TCPCLv4), which was an effort of the Delay-Tolerant Networking Research Group.

Authors' Addresses

Carlo Caini (editor)
ARCES, University of Bologna
via Toffano 2/2
40125 Bologna
Italy
Email: carlo.caini@unibo.it
URI: https://www.unibo.it/sitoweb/carlo.caini/en

Tomaso de Cola
German Aerospace Center, DLR
Munchener Str. 20
82234 Wessling
Germany
Email: tomaso.decola@dlr.de

Mattia Moffa
ARCES, University of Bologna
via Toffano 2/2
40125 Bologna
Italy
Email: mattia.moffa@studio.unibo.it