
Stream: Internet Engineering Task Force (IETF)
RFC: [9174](#)
Category: Standards Track
Published: January 2022
ISSN: 2070-1721
Authors: B. Sipos M. Demmer J. Ott S. Perreault
RKF Engineering *Technical University of Munich* *LogMeIn*

RFC 9174

Delay-Tolerant Networking TCP Convergence-Layer Protocol Version 4

Abstract

This document describes a TCP convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). This version of the TCPCL protocol resolves implementation issues in the earlier TCPCL version 3 as defined in RFC 7242 and provides updates to the Bundle Protocol (BP) contents, encodings, and convergence-layer requirements in BP version 7 (BPv7). Specifically, TCPCLv4 uses BPv7 bundles encoded by the Concise Binary Object Representation (CBOR) as its service data unit being transported and provides a reliable transport of such bundles. This TCPCL version also includes security and extensibility mechanisms.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <https://www.rfc-editor.org/info/rfc9174>.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions

with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction
 - 1.1. Scope
2. Requirements Language
 - 2.1. Definitions Specific to the TCPCL Protocol
3. General Protocol Description
 - 3.1. Convergence-Layer Services
 - 3.2. TCPCL Session Overview
 - 3.3. TCPCL States and Transitions
 - 3.4. PKIX Environments and CA Policy
 - 3.5. Session-Keeping Policies
 - 3.6. Transfer Segmentation Policies
 - 3.7. Example Message Exchange
4. Session Establishment
 - 4.1. TCP Connection
 - 4.2. Contact Header
 - 4.3. Contact Validation and Negotiation
 - 4.4. Session Security
 - 4.4.1. Entity Identification
 - 4.4.2. Certificate Profile for the TCPCL
 - 4.4.3. TLS Handshake
 - 4.4.4. TLS Authentication
 - 4.4.5. Policy Recommendations
 - 4.4.6. Example TLS Initiation
 - 4.5. Message Header
 - 4.6. Session Initialization Message (SESS_INIT)
 - 4.7. Session Parameter Negotiation

- 4.8. Session Extension Items
- 5. Established Session Operation
 - 5.1. Upkeep and Status Messages
 - 5.1.1. Session Upkeep (KEEPALIVE)
 - 5.1.2. Message Rejection (MSG_REJECT)
 - 5.2. Bundle Transfer
 - 5.2.1. Bundle Transfer ID
 - 5.2.2. Data Transmission (XFER_SEGMENT)
 - 5.2.3. Data Acknowledgments (XFER_ACK)
 - 5.2.4. Transfer Refusal (XFER_REFUSE)
 - 5.2.5. Transfer Extension Items
- 6. Session Termination
 - 6.1. Session Termination Message (SESS_TERM)
 - 6.2. Idle Session Termination
- 7. Security Considerations
 - 7.1. Threat: Passive Leak of Node Data
 - 7.2. Threat: Passive Leak of Bundle Data
 - 7.3. Threat: TCPCL Version Downgrade
 - 7.4. Threat: Transport Security Stripping
 - 7.5. Threat: Weak TLS Configurations
 - 7.6. Threat: Untrusted End-Entity Certificate
 - 7.7. Threat: Certificate Validation Vulnerabilities
 - 7.8. Threat: Symmetric Key Limits
 - 7.9. Threat: BP Node Impersonation
 - 7.10. Threat: Denial of Service
 - 7.11. Mandatory-to-Implement TLS
 - 7.12. Alternate Uses of TLS
 - 7.12.1. TLS without Authentication
 - 7.12.2. Non-certificate TLS Use
 - 7.13. Predictability of Transfer IDs

- 8. IANA Considerations
 - 8.1. Port Number
 - 8.2. Protocol Versions
 - 8.3. Session Extension Types
 - 8.4. Transfer Extension Types
 - 8.5. Message Types
 - 8.6. XFER_REFUSE Reason Codes
 - 8.7. SESS_TERM Reason Codes
 - 8.8. MSG_REJECT Reason Codes
 - 8.9. Object Identifier for PKIX Module Identifier
 - 8.10. Object Identifier for PKIX Other Name Forms
 - 8.11. Object Identifier for PKIX Extended Key Usage
 - 9. References
 - 9.1. Normative References
 - 9.2. Informative References
- [Appendix A. Significant Changes from RFC 7242](#)
- [Appendix B. ASN.1 Module](#)
- [Appendix C. Example of the BundleEID Other Name Form](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

1. Introduction

This document describes the TCP convergence-layer protocol for Delay-Tolerant Networking (DTN). DTN is an end-to-end architecture providing communications in and/or through highly stressed environments, including those with intermittent connectivity, long and/or variable delays, and high bit error rates. More detailed descriptions of the rationale and capabilities of these networks can be found in "[Delay-Tolerant Networking Architecture](#)" [RFC4838].

An important goal of the DTN architecture is to accommodate a wide range of networking technologies and environments. The protocol used for DTN communications is the Bundle Protocol version 7 (BPv7) [RFC9171], an application-layer protocol that is used to construct a store-and-forward overlay network. BPv7 requires the services of a "convergence-layer adapter"

(CLA) to send and receive bundles using the service of some "native" link, network, or Internet protocol. This document describes one such convergence-layer adapter that uses the well-known Transmission Control Protocol (TCP). This convergence layer is referred to as TCP Convergence Layer version 4 (TCPCLv4). For the remainder of this document,

- the abbreviation "BP" without the version suffix refers to BPv7.
- the abbreviation "TCPCL" without the version suffix refers to TCPCLv4.

The locations of the TCPCL and the Bundle Protocol in the Internet model protocol stack (described in [RFC1122]) are shown in Figure 1. In particular, when BP is using TCP as its bearer with the TCPCL as its convergence layer, both BP and the TCPCL reside at the application layer of the Internet model.

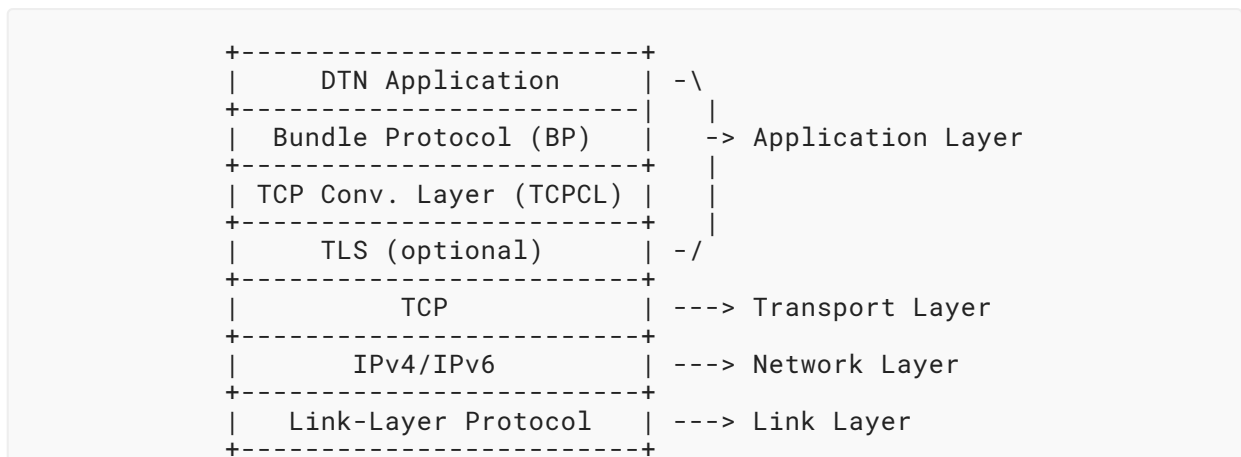


Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol above the Internet Protocol Stack

1.1. Scope

This document describes the format of the protocol data units passed between entities participating in TCPCL communications. This document does not address:

- The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [RFC9171]. This includes the concept of bundle fragmentation or bundle encapsulation. The TCPCL transfers bundles as opaque data blocks.
- Mechanisms for locating or identifying other bundle entities (peers) within a network or across an internet. The mapping of a node ID to a potential convergence layer (CL) protocol and network address is left to implementation and configuration of the BP Agent (BPA) and its various potential routing strategies, as is the mapping of a DNS name and/or address to a choice of an end-entity certificate to authenticate a node to its peers.
- Logic for routing bundles along a path toward a bundle's endpoint. This CL protocol is involved only in transporting bundles between adjacent entities in a routing sequence.
- Policies or mechanisms for issuing Public Key Infrastructure Using X.509 (PKIX) certificates; provisioning, deploying, or accessing certificates and private keys; deploying or accessing

certificate revocation lists (CRLs); or configuring security parameters on an individual entity or across a network.

- Uses of TLS that are not based on PKIX certificate authentication (see [Section 7.12.2](#)) or in which authentication of both entities is not possible (see [Section 7.12.1](#)).

Any TCPCL implementation requires a BPA to perform those above-listed functions in order to perform end-to-end bundle delivery.

2. Requirements Language

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

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions specific to the TCPCL protocol.

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

TCPCL Entity: This is the notional TCPCL application that initiates TCPCL 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 TCPCL sessions. The relationship between a TCPCL entity and TCPCL sessions is defined as follows:

- A TCPCL entity **MAY** actively initiate any number of TCPCL sessions and should do so whenever the entity is the initial transmitter of information to another entity in the network.
- A TCPCL entity **MAY** support zero or more passive listening elements that listen for connection requests from other TCPCL entities operating on other entities in the network.
- A TCPCL entity **MAY** passively initiate any number of TCPCL 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 TCPCL 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 "TCPCL" refers to a TCPCL entity.

TCP Connection: The term "connection" in this specification exclusively refers to a TCP connection and any and all behaviors, sessions, and other states associated with that TCP connection.

TCPCL Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two TCPCL entities. A TCPCL session operates within a single underlying TCP connection, and the lifetime of a TCPCL session is bound to the lifetime of that TCP connection. A TCPCL session is terminated when the TCP connection ends, due to either (1) one or both entities actively closing the TCP connection or (2) network errors causing a failure of the TCP connection. Within a single TCPCL session, there are two possible transfer streams: one in each direction, with one stream from each entity being the outbound stream and the other being the inbound stream (see [Figure 3](#)). From the perspective of a TCPCL session, the two transfer streams do not logically interact with each other. The streams do operate over the same TCP connection and between the same BPAs, so there are logical relationships at those layers (message and bundle interleaving, respectively). For the remainder of this document, the term "session" without the prefix "TCPCL" refers to a TCPCL session.

Session Parameters: These are a set of values used to affect the operation of the TCPCL for a given session. The manner in which these parameters are conveyed to the bundle entity and thereby to the TCPCL is implementation dependent. However, the mechanism by which two entities exchange and negotiate the values to be used for a given session is described in [Section 4.3](#).

Transfer Stream: A transfer stream is a unidirectional user-data path within a TCPCL session. Transfers sent over a transfer stream are serialized, meaning that one transfer must complete its transmission prior to another transfer being started over the same transfer stream. At the stream layer, there is no logical relationship between transfers in that stream; it's only within the BPA that transfers are fully decoded as bundles. Each unidirectional stream has a single sender entity and a single receiver entity.

Transfer: This refers to the procedures and mechanisms for conveyance of an individual bundle from one node to another. Each transfer within the TCPCL 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 is a subset of a transfer of user data being communicated over a transfer stream.

Idle Session: A TCPCL session is idle while there is no transmission in progress in either direction. While idle, the only messages being transmitted or received are KEEPALIVE messages.

Live Session: A TCPCL session is live while there is a transmission in progress in either direction.

Reason Codes: The TCPCL uses numeric codes to encode specific reasons for individual failure/error message types.

The relationship between connections, sessions, and streams is shown in [Figure 3](#).

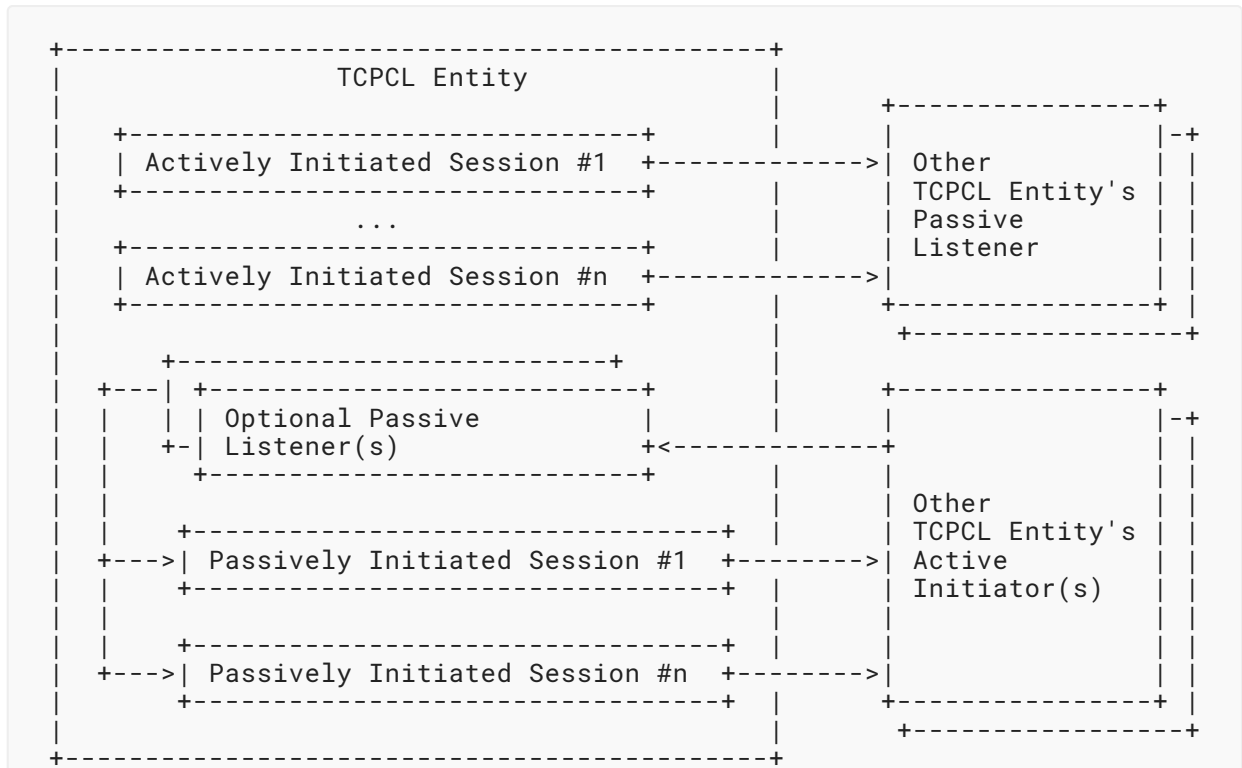


Figure 2: The Relationships between TCPCL Entities

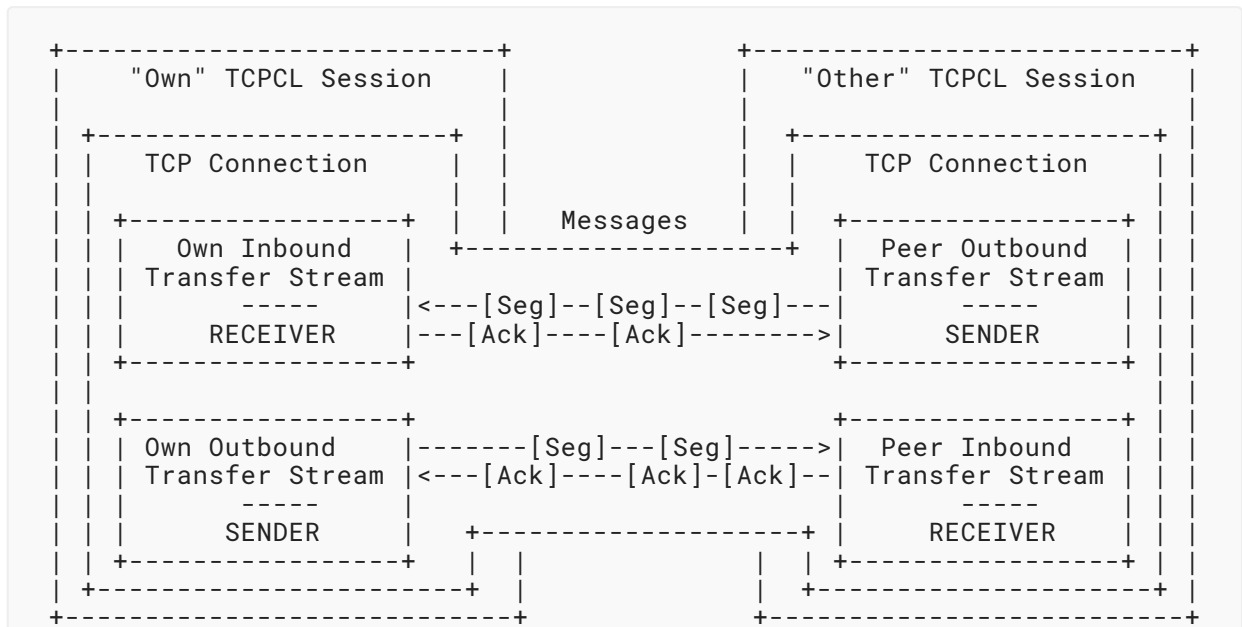


Figure 3: The Relationship within a TCPCL Session of its Two Streams

3. General Protocol Description

The service of this protocol is the transmission of DTN bundles via TCP. This document specifies the encapsulation of bundles, procedures for TCP setup and teardown, and a set of messages and entity requirements. The general operation of the protocol is as follows.

3.1. Convergence-Layer Services

This version of the TCPCL protocol provides the following services to support the overlaying BPA. In all cases, this is not an API definition but a logical description of how the CL can interact with the BPA. 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.

Attempt Session: The TCPCL allows a BPA to preemptively attempt to establish a TCPCL session with a peer entity. Each session attempt can send a different set of session negotiation parameters as directed by the BPA.

Terminate Session: The TCPCL allows a BPA to preemptively terminate an established TCPCL session with a peer entity. The terminate request is done on a per-session basis.

Session State Changed: The TCPCL entity indicates to the BPA when the session state changes. The top-level session states indicated are as follows:

Connecting: A TCP connection is being established. This state only applies to the active entity.

Contact Negotiating: A TCP connection has been made (as either the active or passive entity), and contact negotiation has begun.

Session Negotiating: Contact negotiation has been completed (including possible TLS use), and session negotiation has begun.

Established: The session has been fully established and is ready for its first transfer. When the session is established, the peer node ID (along with an indication of whether or not it was authenticated) and the negotiated session parameters (see [Section 4.7](#)) are also communicated to the BPA.

Ending: The entity sent a SESS_TERM message and is in the Ending state.

Terminated: The session has finished normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed: The TCPCL entity indicates 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 either transfer stream; the session transitions from Live to Idle at the end of a transfer when the other transfer stream does not have an ongoing transfer. Because the TCPCL transmits serially over a TCP connection, it suffers from "head-of-queue blocking", so a transfer in either direction can block an immediate start of a new transfer in the session.

Begin Transmission: The principal purpose of the TCPCL is to allow a BPA to transmit bundle data over an established TCPCL session. Transmission requests are done on a per-session basis, and the CL does not necessarily perform any per-session or inter-session queueing. Any queueing of transmissions is the obligation of the BPA.

Transmission Success: The TCPCL entity indicates to the BPA when a bundle has been fully transferred to a peer entity.

Transmission Intermediate Progress: The TCPCL entity indicates to the BPA the intermediate progress of a transfer to a peer entity. This intermediate progress is at the granularity of each transferred segment.

Transmission Failure: The TCPCL entity indicates to the BPA certain reasons for bundle transmission failure, notably when the peer entity rejects the bundle or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

Reception Initialized: The TCPCL entity indicates this status to the receiving BPA just before any transmission data is sent. This corresponds to reception of the XFER_SEGMENT message with the START flag set to 1.

Interrupt Reception: The TCPCL entity allows a BPA to interrupt an individual transfer before it has fully completed (successfully or not). Interruption can occur any time after the reception is initialized.

Reception Success: The TCPCL entity indicates to the BPA when a bundle has been fully transferred from a peer entity.

Reception Intermediate Progress: The TCPCL entity indicates to the BPA 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: The TCPCL entity indicates to the BPA certain reasons for reception failure, notably when the local entity rejects an attempted transfer for some local policy reason or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

3.2. TCPCL Session Overview

First, one entity establishes a TCPCL session to the other by initiating a TCP connection in accordance with [RFC0793]. After setup of the TCP connection is complete, an initial Contact Header is exchanged in both directions to establish a shared TCPCL version and negotiate the use of TLS security (as described in Section 4). Once contact negotiation is complete, TCPCL messaging is available and the session negotiation is used to set parameters of the TCPCL session. One of these parameters is a node ID; each TCPCL entity is acting on behalf of a BPA having a node ID. This is used to assist in routing and forwarding messages by the BPA and is part of the authentication capability provided by TLS.

Once negotiated, the parameters of a TCPCL session cannot change; if there is a desire by either peer to transfer data under different parameters, then a new session must be established. This makes CL logic simpler but relies on the assumption that establishing a TCP connection is lightweight enough that TCP connection overhead is negligible compared to TCPCL data sizes.

Once the TCPCL session is established and configured in this way, bundles can be transferred in either direction. Each transfer is performed by segmenting the transfer data into one or more XFER_SEGMENT messages. Multiple bundles can be transmitted consecutively in a single direction on a single TCPCL connection. Segments from different bundles are never interleaved. Bundle interleaving can be accomplished by fragmentation at the BP layer or by establishing multiple TCPCL sessions between the same peers. There is no fundamental limit on the number of TCPCL sessions that a single entity can establish, beyond the limit imposed by the number of available (ephemeral) TCP ports of the active entity.

One feature of this protocol is that the receiving entity can send acknowledgment (XFER_ACK) messages as bundle data segments arrive. The rationale behind these acknowledgments is to enable the transmitting entity to determine how much of the bundle has been received, so that if the session is interrupted, it can perform reactive fragmentation to avoid resending the already-transmitted part of the bundle. In addition, there is no explicit flow control on the TCPCL.

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

Note: This enables a cross-layer optimization in that it allows a receiver that detects that it has already received a certain bundle to interrupt transmission as early as possible and thus save transmission capacity for other bundles.

For sessions that are idle, a KEEPALIVE message is sent at a negotiated interval. This is used to convey entity liveness information during otherwise messageless time intervals.

A SESS_TERM message is used to initiate the ending of a TCPCL session (see Section 6.1). 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 (see Section 4.3). Regardless of the reason, session termination is initiated by one of the entities and the other

entity responds to it, as illustrated by Figures 13 and 14 in the next subsection. Even when there are no transfers queued or in progress, the session termination procedure allows each entity to distinguish between a clean end to a session and the TCP connection being closed because of some underlying network issue.

Once a session is established, the TCPCL is a symmetric protocol between the peers. Both sides can start sending data segments in a session, and one side's bundle transfer does not have to complete before the other side can start sending data segments on its own. Hence, the protocol allows for a bidirectional mode of communication. Note that in the case of concurrent bidirectional transmission, acknowledgment segments **MAY** be interleaved with data segments.

3.3. TCPCL States and Transitions

The states of a normal TCPCL session (i.e., without session failures) are indicated in Figure 4.

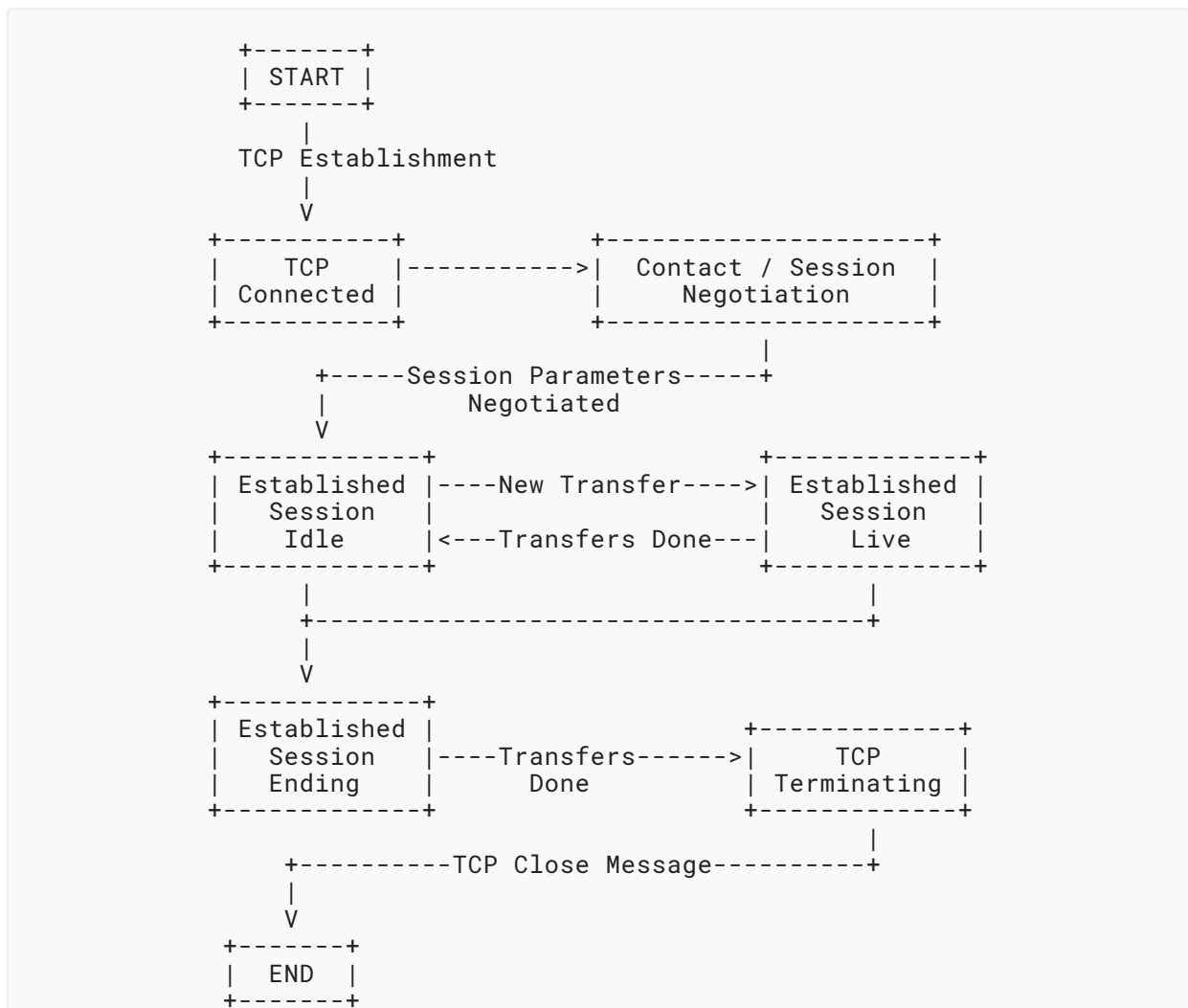


Figure 4: Top-Level States of a TCPCL Session

Notes on established session states:

- Session "Live" means transmitting or receiving over a transfer stream.
- Session "Idle" means no transmission/reception over a transfer stream.
- Session "Ending" means no new transfers will be allowed.

Contact negotiation involves exchanging a Contact Header ("CH" in Figures 5, 6, and 7) in both directions and deriving a negotiated state from the two headers. The contact negotiation sequencing is performed as either the active or passive entity and is illustrated in Figures 5 and 6, respectively, which both share the data validation and negotiation of the Processing of Contact Header ("[PCH]") activity (Figure 7) and the "[TCPCLOSE]" activity, which indicates TCP connection close. Successful negotiation results in one of the Session Initiation ("[SI]") activities being performed, as shown further below. To avoid data loss, a Session Termination ("[ST]") exchange allows cleanly finishing transfers before a session is ended.

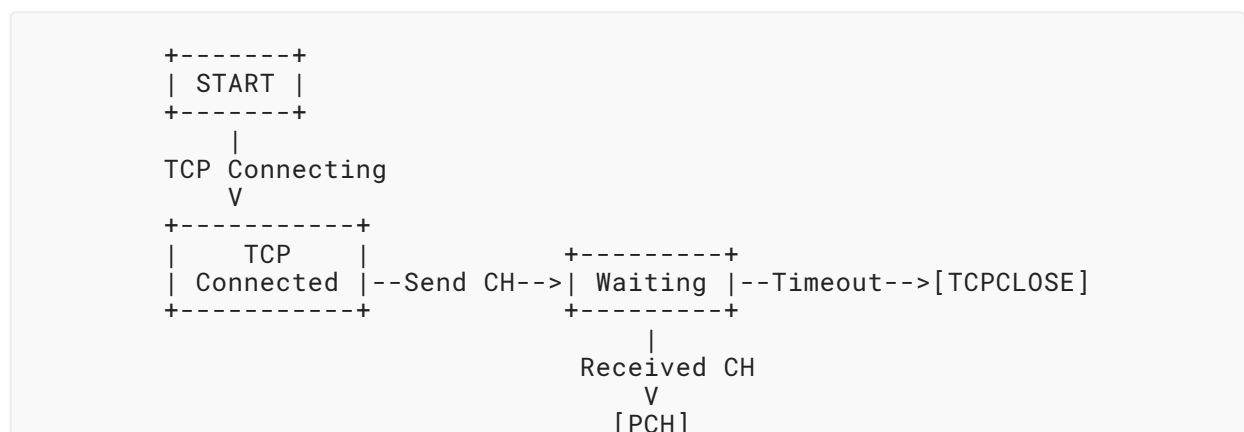


Figure 5: Contact Initiation as Active Entity

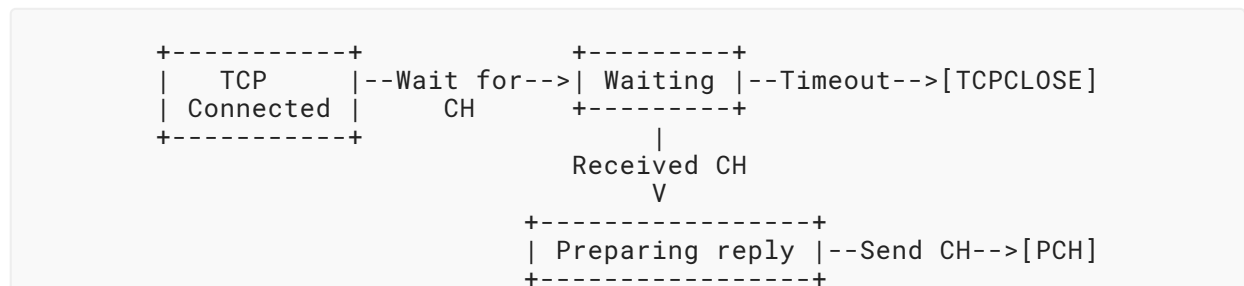


Figure 6: Contact Initiation as Passive Entity

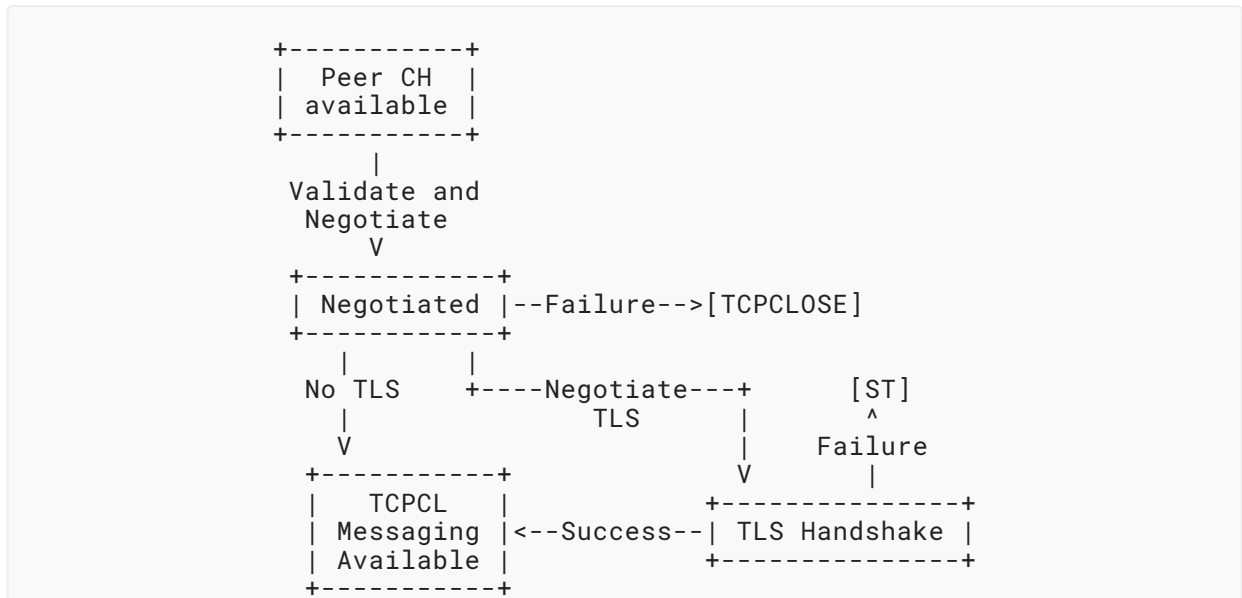


Figure 7: Processing of Contact Header [PCH]

Session negotiation involves exchanging a session initialization (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 Figures 8 and 9, respectively (where "[PSI]" means "Processing of Session Initiation"), which both share the data validation and negotiation shown in Figure 10. The validation here includes certificate validation and authentication when TLS is used for the session.

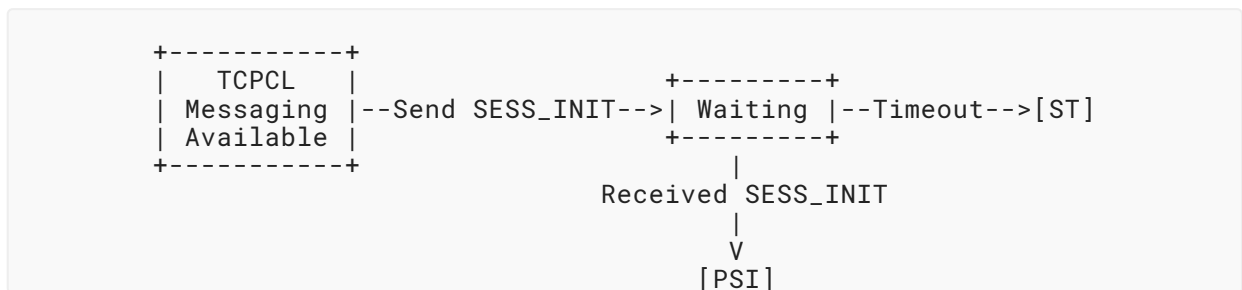


Figure 8: Session Initiation [SI] as Active Entity

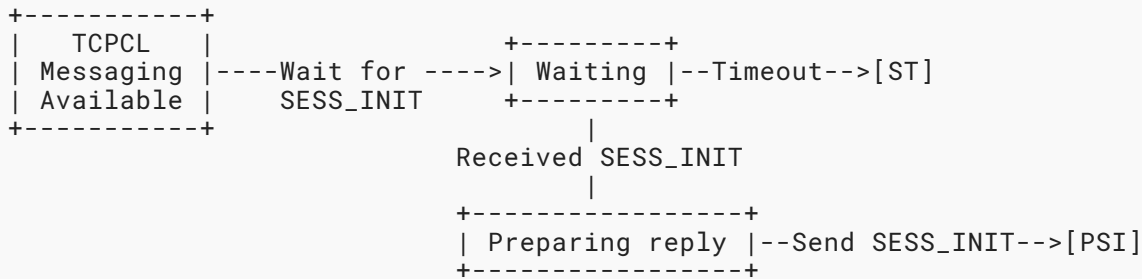


Figure 9: Session Initiation [SI] as Passive Entity

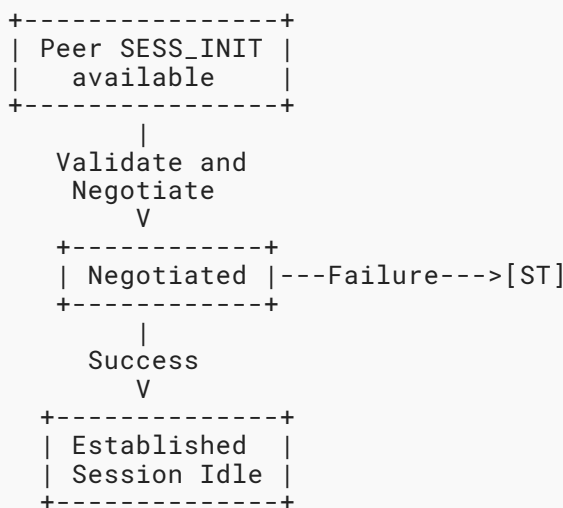


Figure 10: Processing of Session Initiation [PSI]

Transfers can occur after a session is established and it's not in the Ending state. Each transfer occurs within a single logical transfer stream between a sender and a receiver, as illustrated in Figures 11 and 12, respectively.

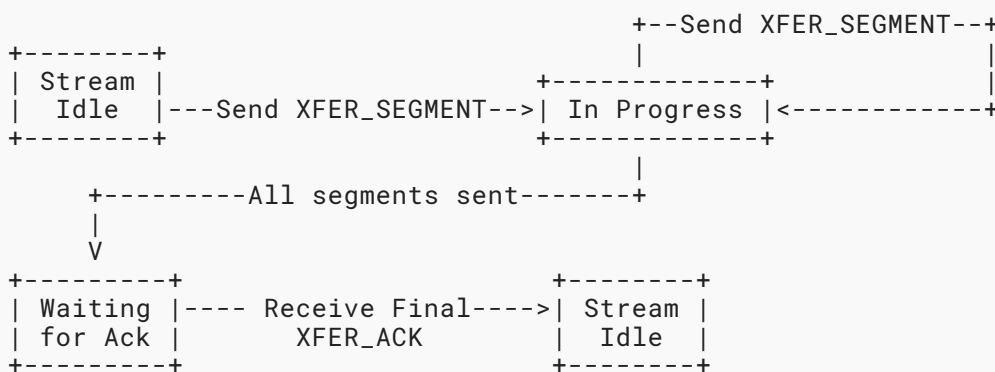


Figure 11: Transfer Sender States

Note on transfer sending: Pipelining of transfers can occur when the sending entity begins a new transfer while in the "Waiting for Ack" state.

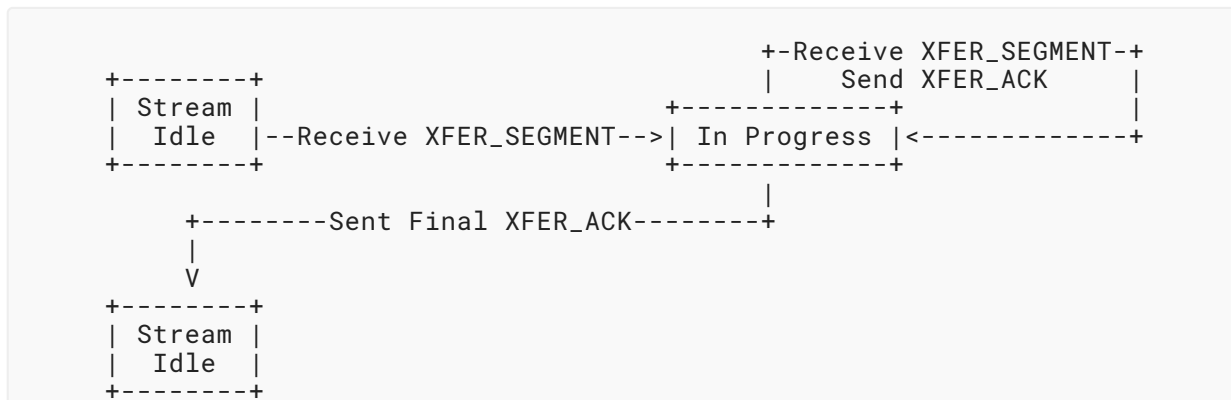


Figure 12: Transfer Receiver States

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 13: Session Termination [ST] from the Initiator

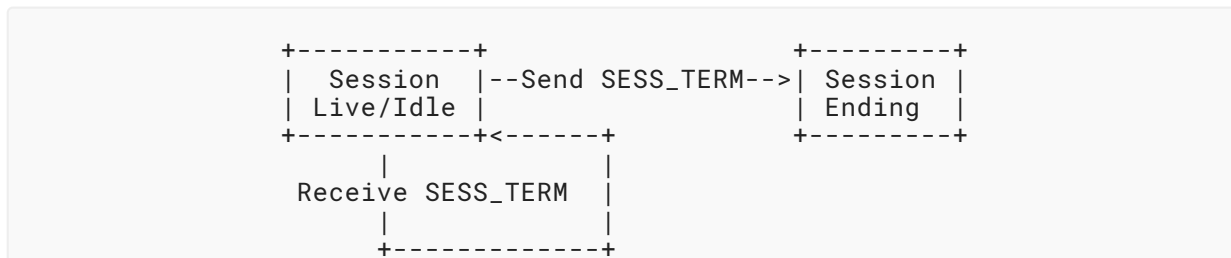


Figure 14: Session Termination [ST] from the Responder

3.4. PKIX Environments and CA Policy

This specification defines requirements regarding how to use PKIX certificates issued by a Certificate Authority (CA) but does not define any mechanisms for how those certificates come to be. The requirements regarding TCPCL certificate use are broad, to support two quite different PKIX environments:

DTN-Aware CAs: In the ideal case, the CA or CAs issuing certificates for TCPCL entities are aware of the end use of the certificate, have a mechanism for verifying ownership of a node ID, and are issuing certificates directly for that node ID. In this environment, the ability to authenticate a peer entity node ID directly avoids the need to authenticate a network name or address and then implicitly trust the node ID of the peer. The TCPCL authenticates the node ID whenever possible; this is preferred over lower-level PKIX identities.

DTN-Ignorant CAs: It is expected that Internet-scale "public" CAs will continue to focus on DNS names as the preferred PKIX identifier. There are large infrastructures already in place for managing network-level authentication and protocols to manage identity verification in those environments [RFC8555]. The TCPCL allows for this type of environment by authenticating a lower-level identifier for a peer and requiring the entity to trust that the node ID given by the peer (during session initialization) is valid. This situation is not ideal, as it allows the vulnerabilities described in Section 7.9, but it still provides some amount of mutual authentication to take place for a TCPCL session.

Even within a single TCPCL session, each entity may operate within different PKI environments and with different identifier limitations. The requirements related to identifiers in a PKIX certificate are provided in Section 4.4.1.

It is important for interoperability that a TCPCL entity have its own security policy tailored to accommodate the peers with which it is expected to operate. Some security policy recommendations are given in Section 4.4.5, but these are meant as a starting point for tailoring. A strict TLS security policy is appropriate for a private network with a single shared CA. Operation on the Internet (such as inter-site BP gateways) could trade more lax TCPCL security with the use of encrypted bundle encapsulation [DTN-BIBECT] to ensure strong bundle security.

By using the Server Name Indication (SNI) DNS name (see Section 4.4.3), a single passive entity can act as a convergence layer for multiple BPAs with distinct node IDs. When this "virtual host" behavior is used, the DNS name is used as the indication of which BP node the active entity is attempting to communicate with. A virtual host CL entity can be authenticated by a certificate containing all of the DNS names and/or node IDs being hosted or by several certificates each authenticating a single DNS name and/or node ID, using the SNI value from the peer to select which certificate to use. The logic for mapping an SNI DNS name to an end-entity certificate is an implementation matter and can involve correlating a DNS name with a node ID or other certificate attributes.

3.5. Session-Keeping Policies

This specification defines requirements regarding how to initiate, sustain, and terminate a TCPCL 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, but guidance given here can be used to steer policy toward performance goals.

Persistent Session: This policy preemptively establishes a single session to known entities in the network and keeps the session active using KEEPALIVES. Benefits of this policy include reducing the total amount of TCP data that needs to be exchanged for a set of transfers

(assuming that the KEEPALIVE size is significantly smaller than the transfer size) and allowing the session state to indicate peer connectivity. Drawbacks include wasted network resources when a session is mostly idle or when network connectivity is inconsistent (which requires that failed sessions be reestablished), and potential queueing issues when multiple transfers are requested simultaneously. This policy assumes that there is agreement between pairs of entities as to which of the peers will initiate sessions; if there is no such agreement, there is potential for duplicate sessions to be established between peers.

Ephemeral Sessions: This policy only establishes a session when an outgoing transfer needs to be sent. Benefits of this policy include not wasting network resources on sessions that are idle for long periods of time and avoiding potential queueing issues as can be seen when using a single persistent session. Drawbacks include the TCP and TLS overhead of establishing a new session for each transfer. This policy assumes that each entity can function in a passive role to listen for session requests from any peer that needs to send a transfer; when that is not the case, the polling behavior discussed below needs to happen. This policy can be augmented to keep the session established as long as any transfers are queued.

Active-Only Polling Sessions: When naming and/or addressing of one entity is variable (i.e., a dynamically assigned IP address or domain name) or when firewall or routing rules prevent incoming TCP connections, that entity can only function in the active role. In these cases, sessions also need to be established when an incoming transfer is expected from a peer or based on a periodic schedule. This polling behavior causes inefficiencies compared to as-needed ephemeral sessions.

Many other policies can be established in a TCPCL network between the two extremes of single persistent sessions and only ephemeral sessions. Different policies can be applied to each peer entity and to each bundle as it needs to be transferred (e.g., for quality of service). Additionally, future session extension types can apply further nuance to session policies and policy negotiation.

3.6. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. A receiving entity can set the Segment Maximum Receive Unit (MRU) in its SESS_INIT message to determine the largest acceptable segment size, and a transmitting entity can segment a transfer into any sizes smaller than the receiver's Segment MRU. It is a network administration matter to determine an appropriate segmentation policy for entities using the TCPCL protocol, but guidance given here can be used to steer policy toward performance goals. Administrators are also advised to consider the Segment MRU in relation to chunking/packetization performed by TLS, TCP, and any intermediate network-layer nodes.

Minimum Overhead: For a simple network expected to exchange relatively small bundles, the Segment MRU can be set to be identical to the Transfer MRU, which indicates that all transfers can be sent with a single data segment (i.e., no actual segmentation). If the network is closed and all transmitters are known to follow a single-segment transfer policy, then receivers can avoid the necessity of segment reassembly. Because this CL operates over a TCP stream, which

suffers from a form of head-of-queue blocking between messages, while one entity is transmitting a single XFER_SEGMENT message it is not able to transmit any XFER_ACK or XFER_REFUSE messages for any associated received transfers.

Predictable Message Sizing: In situations where the maximum message size is desired to be well controlled, the Segment MRU can be set to the largest acceptable size (the message size less the XFER_SEGMENT header size) and transmitters can always segment a transfer into maximum-size chunks no larger than the Segment MRU. This guarantees that any single XFER_SEGMENT will not monopolize the TCP stream for too long, which would prevent outgoing XFER_ACK and XFER_REFUSE messages associated with received transfers.

Dynamic Segmentation: Even after negotiation of a Segment MRU for each receiving entity, the actual transfer segmentation only needs to guarantee that any individual segment is no larger than that MRU. In a situation where TCP throughput is dynamic, the transfer segmentation size can also be dynamic in order to control message transmission duration.

Many other policies can be established in a TCPCL network between the two extremes of minimum overhead (large MRU, single segment) and predictable message sizing (small MRU, highly segmented). Different policies can be applied to each transfer stream to and from any particular entity. Additionally, future session extension and transfer extension types can apply further nuance to transfer policies and policy negotiation.

3.7. Example Message Exchange

Figure 15 depicts the protocol exchange for a simple session, showing the session establishment and the transmission of a single bundle split into three data segments (of lengths "L1", "L2", and "L3") from Entity A to Entity B.

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 a transfer 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 is not allowed.

No errors or rejections are shown in this example.

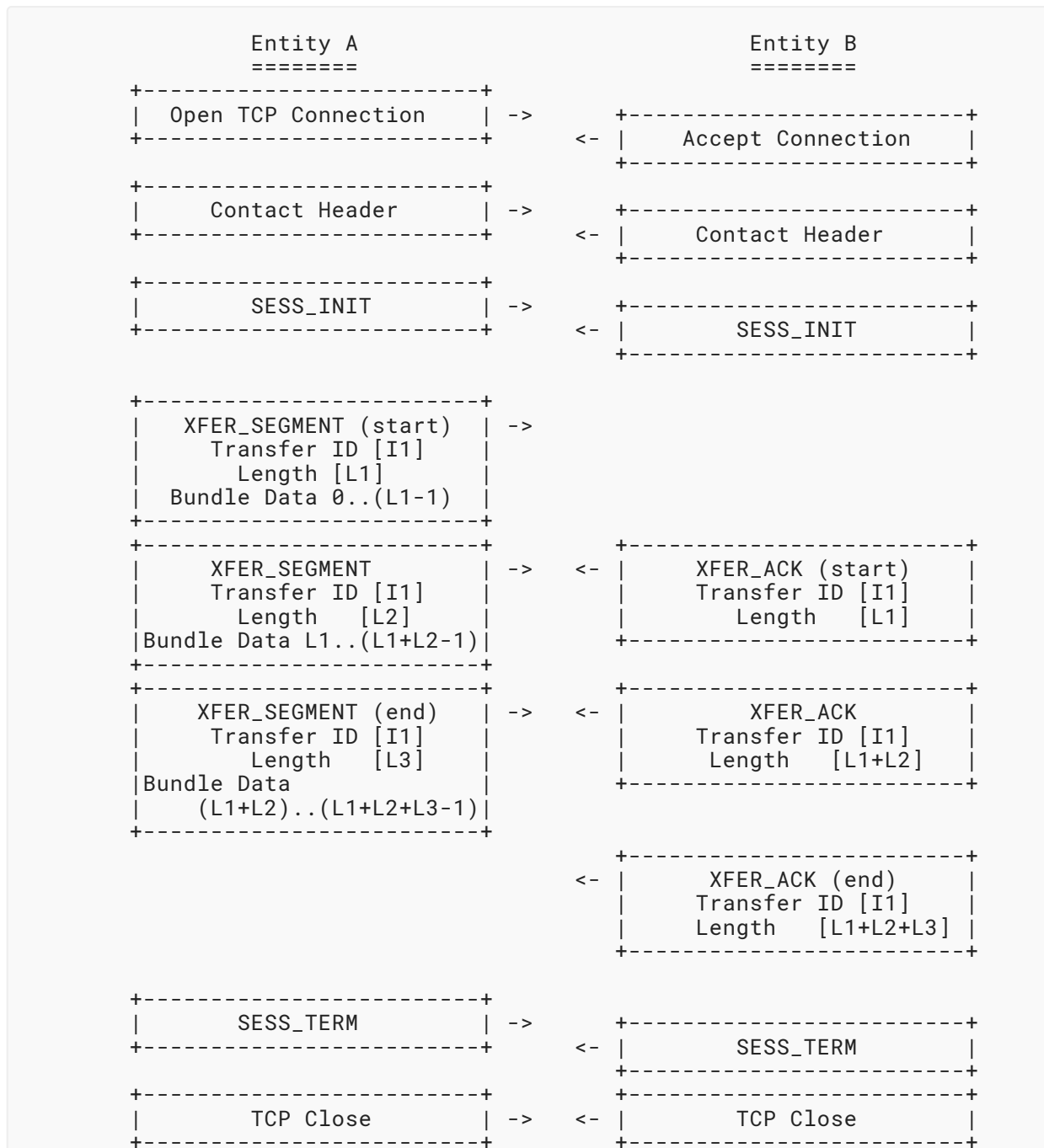


Figure 15: An Example of the Flow of Protocol Messages on a Single TCP Session between Two Entities

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session **MUST** first be established between communicating entities. It is up to the implementation to decide how and when session setup is triggered. For example, some sessions can be opened proactively and maintained for as

long as is possible given the network conditions, while other sessions will be opened only when there is a bundle that is queued for transmission and the routing algorithm selects a certain next-hop node.

4.1. TCP Connection

To establish a TCPCL session, an entity **MUST** first establish a TCP connection with the intended peer entity, typically by using the services provided by the operating system. Destination port number 4556 has been assigned by IANA as the registered port number for the TCPCL; see [Section 8.1](#). Other destination port numbers **MAY** be used per local configuration. Determining a peer's destination port number (if different from the registered TCPCL port number) is left up to the implementation. Any source port number **MAY** be used for TCPCL sessions. Typically, an operating system assigned number in the TCP Ephemeral range (49152-65535) is used.

If the entity is unable to establish a TCP 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.

Once a TCP connection is established, the active entity **SHALL** immediately transmit its Contact Header. The passive entity **SHALL** wait for the active entity's Contact Header. Upon reception of a Contact Header, the passive entity **SHALL** transmit its Contact Header. If either entity does not receive a Contact Header after some implementation-defined time duration after the TCP connection is established, the waiting entity **SHALL** close the TCP connection. Entities **SHOULD** choose a Contact Header reception timeout interval no longer than one minute (60 seconds). The ordering of the Contact Header exchange allows the passive entity to avoid allocating resources to a potential TCPCL session until after a valid Contact Header has been received from the active entity. This ordering also allows the passive peer to adapt to alternate TCPCL protocol versions.

The format of the Contact Header is described in [Section 4.2](#). Because the TCPCL protocol version in use is part of the initial Contact Header, entities using TCPCL version 4 can coexist on a network with entities using earlier TCPCL versions (with some negotiation needed for interoperation, as described in [Section 4.3](#)).

Within this specification, when an entity is said to "close" a TCP connection the entity **SHALL** use the TCP FIN mechanism and not the RST mechanism. However, either mechanism, when received, will cause a TCP connection to become closed.

4.2. Contact Header

This section describes the format of the Contact Header and the meaning of its fields.

If the entity is configured to enable the exchange of messages according to TLS 1.3 [RFC8446] or any successors that are compatible with that TLS ClientHello, the CAN_TLS flag within its Contact Header **SHALL** be set to 1. The **RECOMMENDED** policy is to enable TLS for all sessions, even if security policy does not allow or require authentication. This follows the "opportunistic security" model specified in [RFC7435], though an active attacker could interfere with the exchange in such cases (see Section 7.4).

Upon receipt of the Contact Header, both entities perform the validation and negotiation procedures defined in Section 4.3. After receiving the Contact Header from the other entity, either entity **MAY** refuse the session by sending a SESS_TERM message with an appropriate reason code.

The format for the Contact Header is as follows:

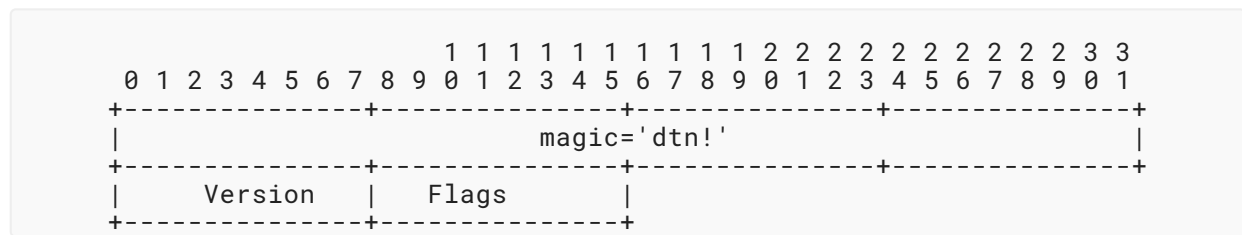


Figure 16: Contact Header Format

See Section 4.3 for details on the use of each of these Contact Header fields.

The fields of the Contact Header are as follows:

magic: A four-octet field that always contains the octet sequence 0x64 0x74 0x6E 0x21, i.e., the text string "dtn!" in US-ASCII (and UTF-8).

Version: A one-octet field value containing the value 4 (current version of the TCPCL protocol).

Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 1. All reserved header flag bits **SHALL** be set to 0 by the sender. All reserved header flag bits **SHALL** be ignored by the receiver.

| Name | Code | Description |
|----------|--------|--|
| CAN_TLS | 0x01 | If this bit is set, it indicates that the sending peer has enabled TLS security. |
| Reserved | others | |

Table 1: Contact Header Flags

4.3. Contact Validation and Negotiation

Upon reception of the Contact Header, each entity follows the following procedures to ensure the validity of the TCPCL session and to negotiate values for the session parameters.

If the "magic string" is not present or is not valid, the connection **MUST** be terminated. The intent of the magic string is to provide some protection against an inadvertent TCP connection by a different protocol than the one described in this document. To prevent a flood of repeated connections from a misconfigured application, a passive entity **MAY** deny new TCP connections from a specific peer address for a period of time after one or more connections fail to provide a decodable Contact Header.

The first negotiation attempts to determine which TCPCL protocol version to use. The active entity always sends its Contact Header first and waits for a response from the passive entity. During contact initiation, the active TCPCL entity **SHALL** send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If the active entity receives a Contact Header with a lower protocol version than the one sent earlier on the TCP connection, the TCP connection **SHALL** be closed. If the active entity receives a SESS_TERM message with a reason code of "Version mismatch", that entity **MAY** attempt further TCPCL sessions with the peer using earlier protocol version numbers in decreasing order. Managing multi-TCPCL-session state such as this is an implementation matter.

If the passive entity receives a Contact Header containing a version that is not a version of the TCPCL protocol that the entity implements, then the entity **SHALL** send its Contact Header and immediately terminate the session with a reason code of "Version mismatch". If the passive entity receives a Contact Header with a version that is lower than the latest version of the protocol that the entity implements, the entity **MAY** either terminate the session (with a reason code of "Version mismatch") or adapt its operation to conform to the older version of the protocol. The decision of version fallback is an implementation matter.

The negotiated contact parameters defined by this specification are described in the following paragraphs.

TCPCL Version: Both Contact Headers of a successful contact negotiation have identical TCPCL version numbers as described above. Only upon response of a Contact Header from the passive entity is the TCPCL protocol version established and session negotiation begun.

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two Contact Headers' CAN_TLS flags. A local security policy is then applied to determine whether the negotiated value of Enable TLS is acceptable. A reasonable security policy would require or disallow the use of TLS, depending upon the desired network flows. The **RECOMMENDED** policy is to require TLS for all sessions, even if security policy does not allow or require authentication. Because this state is negotiated over an unsecured medium, there is a risk of TLS Stripping as described in [Section 7.4](#).

If the Enable TLS state is unacceptable, the entity **SHALL** terminate the session with a reason code of "Contact Failure". Note that this "Contact Failure" reason is different than a failure of a TLS handshake or TLS authentication after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is "true" and acceptable, then the TLS negotiation feature described in [Section 4.4](#) begins immediately following the Contact Header exchange.

4.4. Session Security

This version of the TCPCL protocol supports establishing a TLS session within an existing TCP connection. When TLS is used within the TCPCL, it affects the entire session. Once TLS is established, there is no mechanism available to downgrade the TCPCL session to non-TLS operation.

Once established, the lifetime of a TLS connection **SHALL** be bound to the lifetime of the underlying TCP connection. Immediately prior to actively ending a TLS connection after TCPCL session termination, the peer that sent the original (non-reply) SESS_TERM message **SHOULD** follow the closure alert procedure provided in [RFC8446] to cleanly terminate the TLS connection. Because each TCPCL message is either fixed length or self-indicates its length, the lack of a TLS closure alert will not cause data truncation or corruption.

Subsequent TCPCL session attempts to the same passive entity **MAY** attempt to use the TLS session resumption feature. There is no guarantee that the passive entity will accept the request to resume a TLS session, and the active entity cannot assume any resumption outcome.

4.4.1. Entity Identification

The TCPCL uses TLS for certificate exchange in both directions to identify each entity and to allow each entity to authenticate its peer. Each certificate can potentially identify multiple entities, and there is no problem using such a certificate as long as the identifiers are sufficient to meet authentication policy (as described in later sections) for the entity that presents it.

Because the PKIX environment of each TCPCL entity is likely not controlled by the certificate end users (see Section 3.4), the TCPCL defines a prioritized list of what a certificate can identify regarding a TCPCL entity:

Node ID: The ideal certificate identity is the node ID of the entity using the NODE-ID, as defined below. When the node ID is identified, there is no need for any lower-level identification to be present (though it can still be present, and if so it is also validated).

DNS Name: If CA policy forbids a certificate to contain an arbitrary NODE-ID but allows a DNS-ID to be identified, then one or more stable DNS names can be identified in the certificate. The use of wildcard DNS-IDs is discouraged due to the complex rules for matching and dependence on implementation support for wildcard matching (see Section 6.4.3 of [RFC6125]).

Network Address: If no stable DNS name is available but a stable network address is available and CA policy allows a certificate to contain an IPADDR-ID (as defined below), then one or more network addresses can be identified in the certificate.

This specification defines a NODE-ID of a certificate as being the subjectAltName entry of type otherName with a name form of BundleEID (see Section 4.4.2.1) and a value limited to a node ID. An entity **SHALL** ignore any entry of type otherName with a name form of BundleEID and a value that is some URI other than a node ID. The NODE-ID is similar to the URI-ID as defined in [RFC6125] but is restricted to a node ID rather than a URI with a qualified-name authority part.

Unless specified otherwise by the definition of the URI scheme being authenticated, URI matching of a NODE-ID **SHALL** use the URI comparison logic provided in [RFC3986] and scheme-based normalization of those schemes specified in [RFC9171]. A URI scheme can refine this "exact match" logic with rules regarding how node IDs within that scheme are to be compared with the certificate-authenticated NODE-ID.

This specification reuses the DNS-ID definition in Section 1.8 of [RFC6125], which is the subjectAltName entry of type dNSName whose value is encoded according to [RFC5280].

This specification defines an IPADDR-ID of a certificate as being the subjectAltName entry of type iPAddress whose value is encoded according to [RFC5280].

4.4.2. Certificate Profile for the TCPCL

All end-entity certificates used by a TCPCL entity **SHALL** conform to [RFC5280], or any updates or successors to that profile. When an end-entity certificate is supplied, the full certification chain **SHOULD** be included unless security policy indicates that is unnecessary. An entity **SHOULD** omit the root CA certificate (the last item of the chain) when sending a certification chain, as the recipient already has the root CA to anchor its validation.

The TCPCL requires version 3 certificates due to the extensions used by this profile. TCPCL entities **SHALL** reject as invalid version 1 and version 2 end-entity certificates.

TCPCL entities **SHALL** accept certificates that contain an empty Subject field or contain a Subject without a Common Name. Identity information in end-entity certificates is contained entirely in the subjectAltName extension as defined in Section 4.4.1 and discussed in the paragraphs below.

All end-entity and CA certificates used for the TCPCL **SHOULD** contain both a subject key identifier and an authority key identifier extension in accordance with [RFC5280]. TCPCL entities **SHOULD NOT** rely on either a subject key identifier or an authority key identifier being present in any received certificate. Including key identifiers simplifies the work of an entity that needs to assemble a certification chain.

Unless prohibited by CA policy, a TCPCL end-entity certificate **SHALL** contain a NODE-ID that authenticates the node ID of the peer. When assigned one or more stable DNS names, a TCPCL end-entity certificate **SHOULD** contain a DNS-ID that authenticates those (fully qualified) names. When assigned one or more stable network addresses, a TCPCL end-entity certificate **MAY** contain an IPADDR-ID that authenticates those addresses.

When allowed by CA policy, a Bundle Protocol Security (BPsec; see [RFC9172]) end-entity certificate **SHOULD** contain a PKIX Extended Key Usage (EKU) extension in accordance with Section 4.2.1.12 of [RFC5280]. When the PKIX EKU extension is present, it **SHOULD** contain the key purpose `id-kp-bundleSecurity` (see Section 4.4.2.1). Although not specifically required by the TCPCL, some networks or TLS implementations assume that `id-kp-clientAuth` and `id-kp-serverAuth` need to be used for the client side and the server side of TLS authentication, respectively. For interoperability, a TCPCL end-entity certificate **MAY** contain an EKU with both `id-kp-clientAuth` and `id-kp-serverAuth` values.

When allowed by CA policy, a TCPCL end-entity certificate **SHOULD** contain a PKIX key usage extension in accordance with [Section 4.2.1.3](#) of [\[RFC5280\]](#). The PKIX key usage bit that is consistent with TCPCL security using TLS 1.3 is digitalSignature. The specific algorithms used during the TLS handshake will determine which of those key uses are exercised. Earlier versions of TLS can mandate the use of the keyEncipherment bit or the keyAgreement bit.

When allowed by CA policy, a TCPCL end-entity certificate **SHOULD** contain an Online Certificate Status Protocol (OCSP) URI within an authority information access extension in accordance with [Section 4.2.2.1](#) of [\[RFC5280\]](#).

4.4.2.1. PKIX OID Allocations

This document defines a PKIX Other Name Form identifier, `id-on-bundleEID`, in [Appendix B](#); this identifier can be used as the `type-id` in a `subjectAltName` entry of type `otherName`. The `BundleEID` value associated with the `otherName` type-id `id-on-bundleEID` **SHALL** be a URI, encoded as an IA5String, with a scheme that is present in the IANA "Bundle Protocol URI Scheme Types" registry [[IANA-BUNDLE](#)]. Although this Other Name Form allows any endpoint ID to be present, the `NODE-ID` defined in [Section 4.4.1](#) limits its use to contain only a node ID.

This document defines a PKIX EKU key purpose, `id-kp-bundleSecurity`, in [Appendix B](#); this purpose can be used to restrict a certificate's use. The `id-kp-bundleSecurity` purpose can be combined with other purposes in the same certificate.

4.4.3. TLS Handshake

The use of TLS is negotiated via the Contact Header, as described in [Section 4.3](#). After negotiating an Enable TLS parameter of "true", and before any other TCPCL messages are sent within the session, the session entities **SHALL** begin a TLS handshake in accordance with [\[RFC8446\]](#). By convention, this protocol uses the entity that initiated the underlying TCP connection (the active peer) as the "client" role of the TLS handshake request.

The TLS handshake, if it occurs, is considered to be part of the contact negotiation before the TCPCL session itself is established. Specifics regarding exposure of sensitive data are discussed in [Section 7](#).

The parameters within each TLS negotiation are implementation dependent but any TCPCL entity **SHALL** follow all recommended practices specified in [BCP 195](#) [[RFC7525](#)], or any updates or successors that become part of BCP 195. Within each TLS handshake, the following requirements apply (using the rough order in which they occur):

ClientHello: When a resolved DNS name was used to establish the TCP connection, the TLS ClientHello **SHOULD** include a "server_name" extension in accordance with [\[RFC6066\]](#). When present, the `server_name` extension **SHALL** contain a "HostName" value taken from the DNS name (of the passive entity) that was resolved.

Note: The "HostName" in the `server_name` extension is the network name for the passive entity, not the node ID of that entity.

Server Certificate: The passive entity **SHALL** supply a certificate within the TLS handshake to allow authentication of its side of the session. The supplied end-entity certificate **SHALL** conform to the profile described in [Section 4.4.2](#). The passive entity **MAY** use the SNI DNS name to choose an appropriate server-side certificate that authenticates that DNS name.

Certificate Request: During the TLS handshake, the passive entity **SHALL** request a client-side certificate.

Client Certificate: The active entity **SHALL** supply a certificate chain within the TLS handshake to allow authentication of its side of the session. The supplied end-entity certificate **SHALL** conform to the profile described in [Section 4.4.2](#).

If a TLS handshake cannot negotiate a TLS connection, both entities of the TCPCL session **SHALL** close the TCP connection. At this point, the TCPCL session has not yet been established, so there is no TCPCL session to terminate.

After a TLS connection is successfully established, the active entity **SHALL** send a `SESS_INIT` message to begin session negotiation. This session negotiation and all subsequent messaging are secured.

4.4.4. TLS Authentication

Using PKIX certificates exchanged during the TLS handshake, each of the entities can authenticate a peer node ID directly or authenticate the peer DNS name or network address. The logic for handling certificates and certificate data is separated into the following phases:

1. Validating the certification path from the end-entity certificate up to a trusted root CA.
2. Validating the EKU and other properties of the end-entity certificate.
3. Authenticating identities from a valid end-entity certificate.
4. Applying security policy to the result of each identity type authentication.

The result of validating a peer identity (see [Section 4.4.1](#)) against one or more types of certificate claims is one of the following:

Absent: Indicating that no such claims are present in the certificate and the identity cannot be authenticated.

Success: Indicating that one or more such claims are present and at least one matches the peer identity value.

Failure: Indicating that one or more such claims are present and none match the peer identity.

4.4.4.1. Certificate Path and Purpose Validation

For any peer end-entity certificate received during the TLS handshake, the entity **SHALL** perform the certification path validation described in [\[RFC5280\]](#) up to one of the entity's trusted CA certificates. If enabled by local policy, the entity **SHALL** perform an OCSP check of each certificate providing OCSP authority information in accordance with [\[RFC6960\]](#). If certificate validation fails or if security policy disallows a certificate for any reason, the entity **SHALL** fail the TLS handshake

with a "bad_certificate" alert. Leaving out part of the certification chain can cause the entity to fail to validate a certificate if the certificates that were left out are unknown to the entity (see [Section 7.6](#)).

For the end-entity peer certificate received during the TLS handshake, the entity **SHALL** apply security policy to the key usage extension (if present) and EKU extension (if present) in accordance with Sections [4.2.1.12](#) and [4.2.1.3](#) of [RFC5280], respectively, and with the profile discussed in [Section 4.4.2](#) of this document.

4.4.4.2. Network-Level Authentication

Either during or immediately after the TLS handshake, each entity, if required by security policy, **SHALL** validate the following certificate identifiers together in accordance with [Section 6](#) of [RFC6125]:

- If the active entity resolved a DNS name (of the passive entity) in order to initiate the TCP connection, that DNS name **SHALL** be used as a DNS-ID reference identifier.
- The IP address of the other side of the TCP connection **SHALL** be used as an IPADDR-ID reference identifier.

If the network-level identifier's authentication result is Failure or if the result is Absent and security policy requires an authenticated network-level identifier, the entity **SHALL** terminate the session (with a reason code of "Contact Failure").

4.4.4.3. Node ID Authentication

Immediately before session parameter negotiation, each entity, if required by security policy, **SHALL** validate the certificate NODE-ID in accordance with [Section 6](#) of [RFC6125] using the node ID of the peer's SESS_INIT message as the NODE-ID reference identifier. If the NODE-ID validation result is Failure or if the result is Absent and security policy requires an authenticated node ID, the entity **SHALL** terminate the session (with a reason code of "Contact Failure").

4.4.5. Policy Recommendations

A **RECOMMENDED** security policy encompasses the following:

- enabling the use of OCSP checking during the TLS handshake.
- instructing that, if an EKU extension is present, the extension needs to contain `id-kp-bundleSecurity` ([Section 4.4.2.1](#)) to be usable with TCPCL security.
- requiring a validated node ID ([Section 4.4.4.3](#)) and ignoring any network-level identifier ([Section 4.4.4.2](#)).

This policy relies on and informs the certificate requirements provided in [Section 4.4.3](#). This policy assumes that a DTN-aware CA (see [Section 3.4](#)) will only issue a certificate for a node ID when it has verified that the private key holder actually controls the bundle node; this is needed to avoid the threat identified in [Section 7.9](#). This policy requires that a certificate contain a NODE-ID and allows the certificate to also contain network-level identifiers. A tailored policy on a more controlled network could relax the requirement on node ID validation and allow just network-level identifiers to authenticate a peer.

4.4.6. Example TLS Initiation

A summary of a typical TLS initiation is shown in the sequence in [Figure 17](#) below. In this example, the active peer terminates the session, but termination can be initiated from either peer.

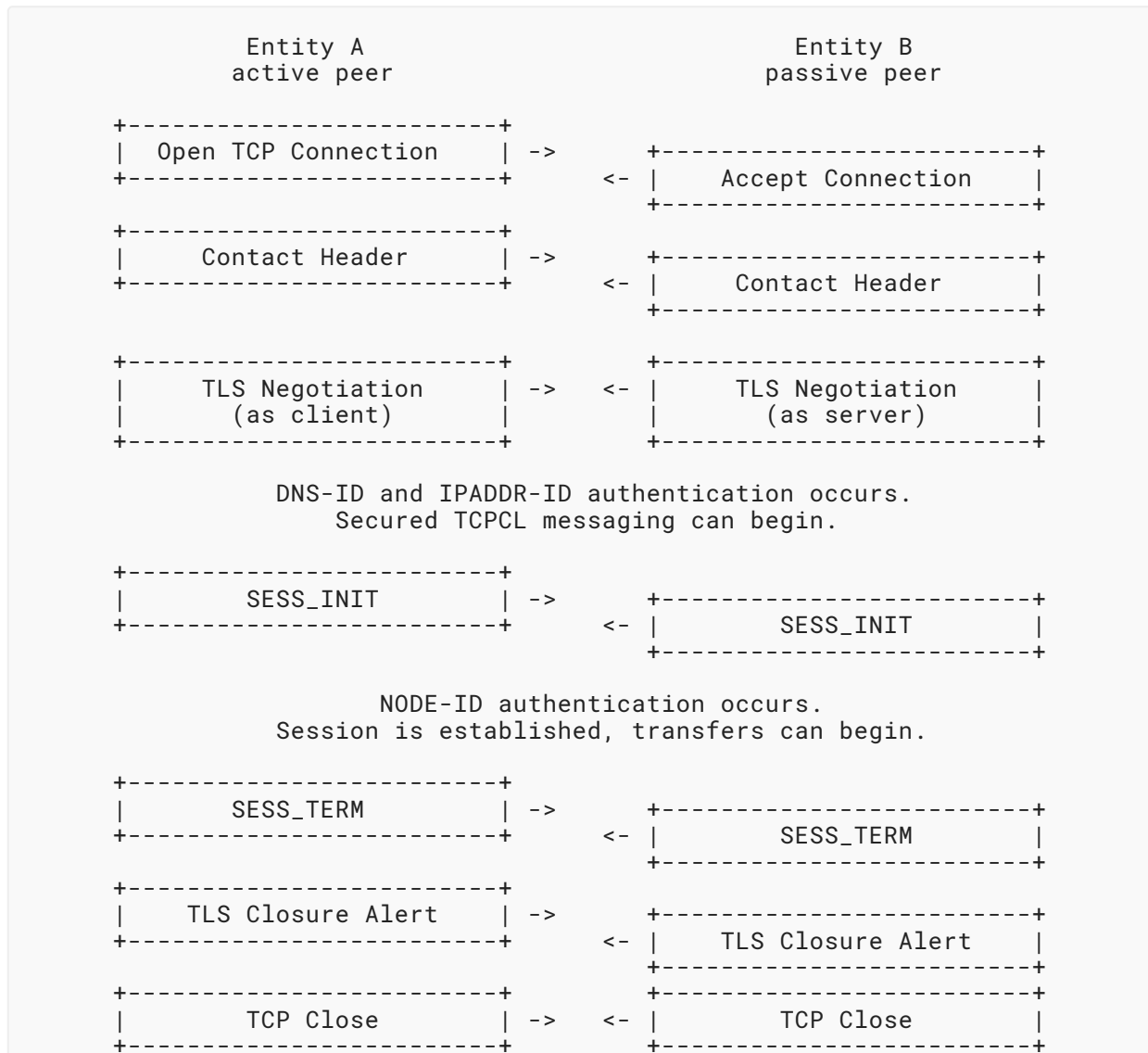


Figure 17: A Simple Visual Example of TCPCL TLS Establishment between Two Entities

4.5. Message Header

After the initial exchange of a Contact Header and (if TLS is negotiated to be used) the TLS handshake, all messages transmitted over the session are identified by a one-octet header with the following structure:

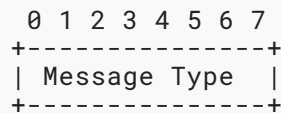


Figure 18: Format of the Message Header

The Message Header contains the following field:

Message Type: Indicates the type of the message as per [Table 2](#) below. Encoded values are listed in [Section 8.5](#).

| Name | Code | Description |
|--------------|------|--|
| SESS_INIT | 0x07 | Contains the session parameter inputs from one of the entities, as described in Section 4.6 . |
| SESS_TERM | 0x05 | Indicates that one of the entities participating in the session wishes to cleanly terminate the session, as described in Section 6.1 . |
| XFER_SEGMENT | 0x01 | Indicates the transmission of a segment of bundle data, as described in Section 5.2.2 . |
| XFER_ACK | 0x02 | Acknowledges reception of a data segment, as described in Section 5.2.3 . |
| XFER_REFUSE | 0x03 | Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.2.4 . |
| KEEPALIVE | 0x04 | Used to keep the TCPCL session active, as described in Section 5.1.1 . |
| MSG_REJECT | 0x06 | Contains a TCPCL message rejection, as described in Section 5.1.2 . |

Table 2: TCPCL Message Types

4.6. Session Initialization Message (SESS_INIT)

Before a session is established and ready to transfer bundles, the session parameters are negotiated between the connected entities. The SESS_INIT message is used to convey the per-entity parameters, which are used together to negotiate the per-session parameters as described in [Section 4.7](#).

The format of a SESS_INIT message is shown in [Figure 19](#).

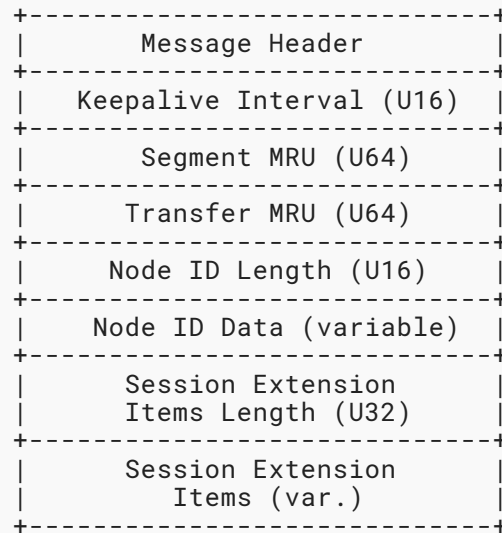


Figure 19: SESS_INIT Format

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 4.7](#).

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.

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 can 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 [\[IANA-BUNDLE\]](#). The node ID itself can be authenticated as described in [Section 4.4.4](#).

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 4.8](#). The full set of Session Extension Items apply for the duration of the TCPCL 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.

If an entity receives a peer node ID that is not authenticated (by the procedure described in [Section 4.4.4.3](#)), that node ID **SHOULD NOT** be used by a BPA for any discovery or routing functions. Trusting an unauthenticated node ID can lead to the threat described in [Section 7.9](#).

When the active entity initiates a TCPCL 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 TCPCL session, the session **MAY** be allowed to be established. If allowed, such a session **SHALL** be associated with the node ID provided in the SESS_INIT message rather than any intended value.

4.7. Session Parameter Negotiation

An entity calculates the parameters for a TCPCL 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 and Segment MTU: The Maximum Transmission Unit (MTU) for whole transfers and individual segments is identical to the Transfer MRU and Segment MRU, respectively, of the received SESS_INIT message. A transmitting peer can send individual segments with any size smaller than the Segment MTU, depending on local policy, dynamic network conditions, etc. Determining the size of each transmitted segment is an implementation matter. If either the Transfer MRU or Segment MRU is unacceptable, the entity **SHALL** terminate the session with a reason code of "Contact 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 5.1.1](#). If the Session Keepalive Interval is unacceptable, the entity **SHALL** terminate the session with a reason code of "Contact 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 TCPCL session **MUST** be terminated and a new session established.

4.8. Session Extension Items

Each of the Session Extension Items **SHALL** be encoded in an identical Type-Length-Value (TLV) container form as indicated in [Figure 20](#).

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 TCPCL entity receives a Session Extension Item with an unknown Item Type and the CRITICAL flag set to 1, the entity **SHALL** terminate the TCPCL session with a SESS_TERM reason code of "Contact 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.

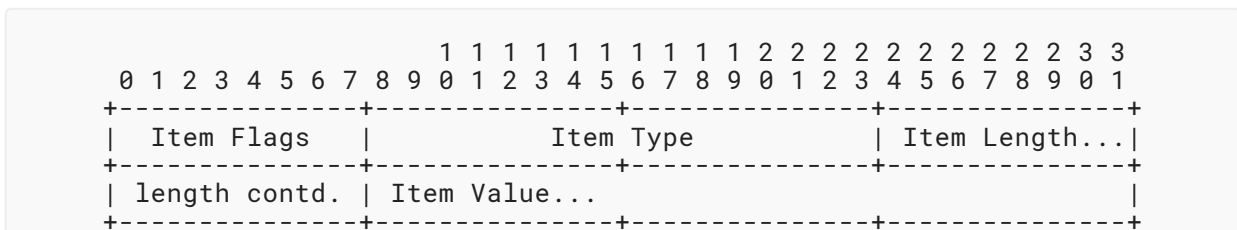


Figure 20: Session Extension Item Format

| Name | Code | Description |
|----------|--------|--|
| CRITICAL | 0x01 | If this bit is set, it indicates that the receiving peer must handle the extension item. |
| Reserved | others | |

Table 3: Session Extension Item Flags

5. Established Session Operation

This section describes the protocol operation for the duration of an established session, including the mechanism for transmitting bundles over the session.

5.1. Upkeep and Status Messages

5.1.1. Session Upkeep (KEEPALIVE)

The protocol includes a provision for transmission of KEEPALIVE messages over the TCPCL session to help determine if the underlying TCP connection has been disrupted.

As described in [Section 4.7](#), a negotiated parameter of each session is the Session Keepalive Interval. 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, entities **SHOULD** choose a Keepalive Interval no shorter than 30 seconds. There is no logical maximum value for the Keepalive Interval (within the maximum imposed by the fixed-size encoding), but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens of minutes. Entities **SHOULD** choose a Keepalive Interval no longer than 10 minutes (600 seconds).

The chosen Keepalive Interval **SHOULD NOT** be too short, as TCP retransmissions may occur in the case of packet loss. Those will have to be triggered by a timeout (TCP retransmission timeout (RTO)), which is dependent on the measured RTT for the TCP connection so that KEEPALIVE messages can experience noticeable latency.

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 implementation-defined time duration, then the entity **SHALL** terminate the session by transmitting a SESS_TERM message (as described in [Section 6.1](#)) 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.

5.1.2. Message Rejection (MSG_REJECT)

This message type is not expected to be seen in a well-functioning session. Its purpose is to aid in troubleshooting bad entity behavior by allowing the peer to observe why an entity is not responding as expected to its messages.

If a TCPCL 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 TCP connection. If a TCPCL 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 TCPCL 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".

The format of a MSG_REJECT message is shown in [Figure 21](#).

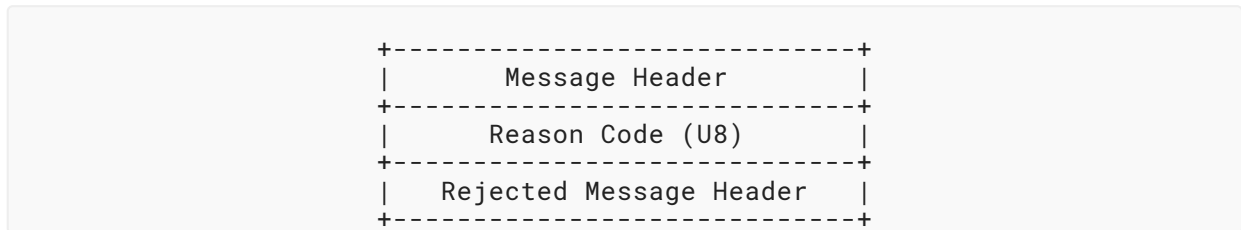


Figure 21: Format of MSG_REJECT Messages

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 4](#).

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 |
|----------------------|------|--|
| Message Type Unknown | 0x01 | A message was received with a Message Type code unknown to the TCPCL entity. |
| Message Unsupported | 0x02 | A message was received, but the TCPCL entity cannot comply with the message contents. |
| Message Unexpected | 0x03 | A message was received while the session is in a state in which the message is not expected. |

Table 4: MSG_REJECT Reason Codes

5.2. Bundle Transfer

All of the messages discussed in this section are directly associated with transferring a bundle between TCPCL entities.

A single TCPCL transfer results in the exchange of a bundle (handled by the convergence layer as opaque data) between two entities. In the TCPCL, a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU, which is defined in [Section](#)

4.6. 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.

A single transfer (and, by extension, a single segment) **SHALL NOT** contain data of more than a single bundle. This requirement is imposed on the agent using the TCPCL, rather than on the TCPCL itself.

If multiple bundles are transmitted on a single TCPCL connection, they **MUST** be transmitted consecutively, without the interleaving of segments from multiple bundles.

5.2.1. Bundle Transfer ID

Each of the bundle transfer messages contains a Transfer ID, which is used to correlate messages (from both sides of a transfer) for each bundle. A Transfer ID does not attempt to address uniqueness of the bundle data itself and is not related to such concepts as bundle fragmentation. Each invocation of the TCPCL by the BPA, requesting transmission of a bundle (fragmentary or otherwise), results in the initiation of a single TCPCL 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 TCPCL 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 TCPCL entity **SHOULD NOT** rely on any relationship between Transfer IDs originating from each side of the TCPCL session.

Although there is not a strict requirement for initial Transfer ID values or the ordering of Transfer IDs (see [Section 7.13](#)), 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.

5.2.2. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message is shown in [Figure 22](#).

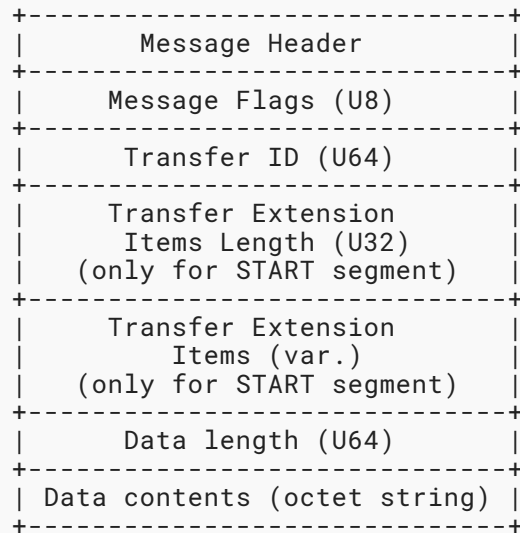


Figure 22: Format of XFER_SEGMENT Messages

The fields of the XFER_SEGMENT message are as follows:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in [Table 5](#). 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 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 5.2.5](#). 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 data payload of the message.

| Name | Code | Description |
|------|------|---|
| END | 0x01 | If this bit is set, it indicates that this is the last segment of the transfer. |

| Name | Code | Description |
|----------|--------|--|
| START | 0x02 | If this bit is set, it indicates that this is the first segment of the transfer. |
| Reserved | others | |

Table 5: XFER_SEGMENT Flags

The flags portion of the message contains two flag values in the two low-order bits, denoted START and END in Table 5. 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.

Once a transfer of a bundle has commenced, the entity **MUST** only send segments containing sequential portions of that bundle until it sends a segment with the END flag set to 1. No interleaving of multiple transfers from the same entity is possible within a single TCPCL session. Simultaneous transfers between two entities **MAY** be achieved using multiple TCPCL sessions.

5.2.3. Data Acknowledgments (XFER_ACK)

Although the TCP transport provides reliable transfer of data between transport peers, the typical BSD sockets interface provides no means to inform a sending application of when the receiving application has processed some amount of transmitted data. Thus, after transmitting some data, the TCPCL needs an additional mechanism to determine whether the receiving agent has successfully received and fully processed the segment. To this end, the TCPCL protocol provides feedback messaging whereby a receiving entity transmits acknowledgments of reception of data segments.

The format of a XFER_ACK message is shown in Figure 23.

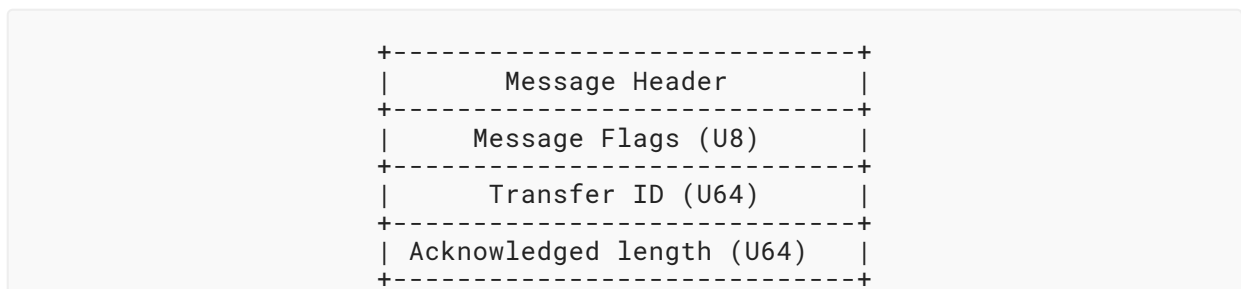


Figure 23: Format of XFER_ACK Messages

The fields of the XFER_ACK message are as follows:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. 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 acknowledged.

Acknowledged length: A 64-bit unsigned integer indicating the total number of octets in the transfer that are being acknowledged.

A receiving TCPCL entity **SHALL** send a XFER_ACK message in response to each received XFER_SEGMENT message after the segment has been fully processed. The flags portion of the XFER_ACK header **SHALL** be set to match the corresponding XFER_SEGMENT message being acknowledged (including flags not decodable to the entity). The acknowledged length of each XFER_ACK contains the sum of the Data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer. 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.

5.2.4. Transfer Refusal (XFER_REFUSE)

The TCPCL supports a mechanism by which a receiving entity can indicate to the sender that it does not want to receive the corresponding bundle. To do so, upon receiving a XFER_SEGMENT message, the entity **MAY** transmit a XFER_REFUSE message. As data segments and acknowledgments can cross on the wire, the bundle that is being refused **SHALL** be identified by the Transfer ID of the refusal.

There is no required relationship between the Transfer MRU of a TCPCL 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 TCPCL 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".

The format of the XFER_REFUSE message is shown in [Figure 24](#).



Figure 24: Format of XFER_REFUSE Messages

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 6](#).

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

| Name | Code | Description |
|---------------------|------|--|
| Unknown | 0x00 | The reason for refusal is unknown or is not specified. |
| Completed | 0x01 | The receiver already has the complete bundle. The sender MAY consider the bundle as completely received. |
| No Resources | 0x02 | The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying. |
| Retransmit | 0x03 | The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety. |
| Not Acceptable | 0x04 | 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. |
| Extension Failure | 0x05 | A failure processing the Transfer Extension Items has occurred. |
| Session Terminating | 0x06 | 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. |

Table 6: XFER_REFUSE Reason Codes

The receiver **MUST**, for each transfer preceding the one to be refused, have either acknowledged all XFER_SEGMENT messages or refused the bundle transfer.

The bundle transfer refusal **MAY** be sent 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 TCPCL 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.

5.2.5. Transfer Extension Items

Each of the Transfer Extension Items **SHALL** be encoded in an identical Type-Length-Value (TLV) container form as indicated in [Figure 25](#).

The fields of the Transfer Extension Item are as follows:

Item Flags: A one-octet field containing generic bit flags related to the Item, which are listed in [Table 7](#). 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 TCPCL 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.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification creates an IANA registry for such codes (see [Section 8.4](#)).

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 the associated transfer cannot begin until the full extension data is sent.

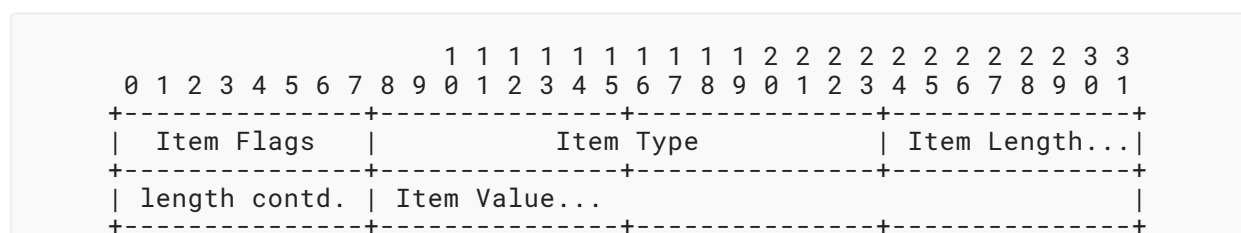


Figure 25: Transfer Extension Item Format

| Name | Code | Description |
|----------|--------|--|
| CRITICAL | 0x01 | If this bit is set, it indicates that the receiving peer must handle the extension item. |
| Reserved | others | |

Table 7: Transfer Extension Item Flags

5.2.5.1. Transfer Length Extension

The purpose of the Transfer Length Extension is to allow entities to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving entity for the upcoming bundle data.

Multiple Transfer Length Extension Items **SHALL NOT** occur within the same transfer. The lack of a Transfer Length Extension Item in any transfer **SHALL NOT** imply anything regarding the potential length of the transfer. The Transfer Length Extension **SHALL** use the IANA-assigned code point from [Section 8.4](#).

If a transfer occupies exactly one segment (i.e., both the START flag and the END flag are 1), the Transfer Length Extension **SHOULD NOT** be present. The extension does not provide any additional information for single-segment transfers.

The format of the Transfer Length Extension data is shown in [Figure 26](#).

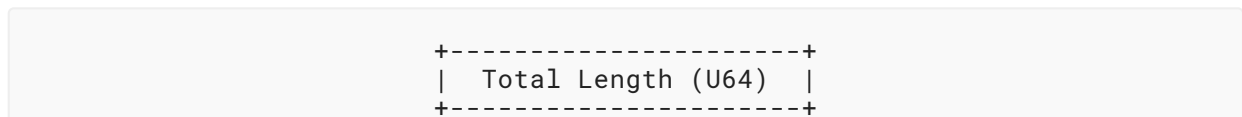


Figure 26: Format of Transfer Length Extension Data

The Transfer Length Extension data contains the following field:

Total Length: A 64-bit unsigned integer indicating the size of the data to be transferred. The Total Length field **SHALL** be treated as authoritative by the receiver. If, for whatever reason, the actual total length of bundle data received differs from the value indicated by the Total Length value, the receiver **SHALL** treat the transmitted data as invalid and send a XFER_REFUSE with a reason code of "Not Acceptable".

6. Session Termination

This section describes the procedures for terminating a TCPCL session. The purpose of terminating a session is to allow transfers to complete before the TCP connection is closed but not allow any new transfers to start. A session state change is necessary for this to happen, because transfers can be in progress in either direction (transfer stream) within a session.

Waiting for a transfer to complete in one direction does not control or influence the possibility of a transfer in the other direction. Either peer of a session can terminate an established session at any time.

6.1. Session Termination Message (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. When sent to initiate a termination, the REPLY flag of a SESS_TERM message **SHALL** be 0. Upon receiving a SESS_TERM message after not sending a SESS_TERM message in the same session, an entity **SHALL** send an acknowledging SESS_TERM message. When sent to acknowledge a termination, a SESS_TERM message **SHALL** have identical data content from the message being acknowledged except for the REPLY flag, which is set to 1 to indicate acknowledgment.

Once a SESS_TERM message is sent, the state of that TCPCL 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 TCP connection associated with a TCPCL session, without waiting for transfers to complete but also in a way that doesn't force timeouts to occur -- for example, due to impending shutdown of the underlying data-link layer. Instead of following a clean termination sequence, after transmitting a SESS_TERM message, an entity **MAY** perform an unclean termination by immediately closing the associated TCP connection. When performing an unclean termination, an entity **SHOULD** acknowledge all received XFER_SEGMENTS with a XFER_ACK before closing the TCP connection. Not acknowledging received segments can result in unnecessary bundle or bundle fragment retransmissions. Any delay between a request to close the TCP connection and the actual closing of the connection (a "half-closed" state) **MAY** be ignored by the TCPCL entity. If the underlying TCP connection is closed during a transmission (in either transfer stream), the transfer **SHALL** be indicated to the BPA as failed (see the transmission failure and reception failure indications defined in [Section 3.1](#)).

The TCPCL 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".

The format of the SESS_TERM message is shown in [Figure 27](#).

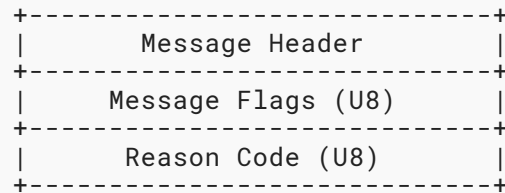


Figure 27: Format of SESS_TERM Messages

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 8](#). 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 9](#).

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

Table 8: SESS_TERM Flags

| Name | Code | Description |
|---------------------|------|--|
| Unknown | 0x00 | A termination reason is not available. |
| Idle timeout | 0x01 | The session is being terminated due to idleness. |
| Version mismatch | 0x02 | The entity cannot conform to the specified TCPCL protocol version. |
| Busy | 0x03 | The entity is too busy to handle the current session. |
| Contact Failure | 0x04 | The entity cannot interpret or negotiate a Contact Header or SESS_INIT option. |
| Resource Exhaustion | 0x05 | The entity has run into some resource limit and cannot continue the session. |

Table 9: SESS_TERM Reason Codes

The earliest a TCPCL session termination **MAY** occur is immediately after transmission of a Contact Header (and prior to any further message transmissions). This can, for example, be used as a notification that the entity is currently not able or willing to communicate. However, an entity **MUST** always send the Contact Header to its peer before sending a SESS_TERM message.

Termination of the TCP connection **MAY** occur prior to receiving the Contact Header as discussed in [Section 4.1](#). If reception of the Contact Header itself somehow fails (e.g., an invalid magic string is received), an entity **SHALL** close the TCP connection without sending a SESS_TERM message.

If a session is to be terminated before the sending of a protocol message has completed, then the entity **MUST NOT** transmit the SESS_TERM message but still **SHALL** close the TCP connection. Each TCPCL message is contiguous in the octet stream and has no ability to be cut short and/or preempted by another message. This is particularly important when large segment sizes are being transmitted; either the entire XFER_SEGMENT is sent before a SESS_TERM message or the connection is simply terminated mid-XFER_SEGMENT.

6.2. Idle Session Termination

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 TCPCL 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 9](#)).

7. Security Considerations

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

7.1. Threat: Passive Leak of Node Data

When used without TLS security, the TCPCL exposes the node ID and other configuration data to passive eavesdroppers. This occurs even when no transfers occur within a TCPCL session. This can be avoided by always using TLS, even if authentication is not available (see [Section 7.12](#)).

7.2. Threat: Passive Leak of Bundle Data

The TCPCL 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.

When used without TLS security, the TCPCL exposes all bundle data to passive eavesdroppers. This can be avoided by always using TLS, even if authentication is not available (see [Section 7.12](#)).

7.3. Threat: TCPCL Version Downgrade

When a TCPCL entity supports multiple versions of the protocol, it is possible for a malicious or misconfigured peer to use an older version of the TCPCL protocol that does not support transport security. An on-path attacker can also manipulate a Contact Header to present a lower protocol version than desired.

It is up to security policies within each TCPCL entity to ensure that the negotiated TCPCL version meets transport security requirements.

7.4. Threat: Transport Security Stripping

When security policy allows non-TLS sessions, the TCPCL does not protect against active network attackers. It is possible for an on-path attacker to set the CAN_TLS flag to 0 on either side of the Contact Header exchange, which will cause the negotiation discussed in [Section 4.3](#) to disable TLS. This leads to the "SSL Stripping" attack described in [\[RFC7457\]](#).

The purpose of the CAN_TLS flag is to allow the use of the TCPCL on entities that simply do not have a TLS implementation available. When TLS is available on an entity, it is strongly encouraged that the security policy disallow non-TLS sessions. This requires that the TLS handshake occur, regardless of the policy-driven parameters of the handshake and policy-driven handling of the handshake outcome.

One mechanism to mitigate the possibility of TLS Stripping is the use of DNS-based Authentication of Named Entities (DANE) [\[RFC6698\]](#) toward the passive peer. This mechanism relies on DNS and is unidirectional, so it doesn't help with applying policy toward the active peer, but it can be useful in an environment using opportunistic security. The configuration and use of DANE are outside of the scope of this document.

The negotiated use of TLS is identical in behavior to the use of STARTTLS as described in [\[RFC2595\]](#), [\[RFC4511\]](#), and others.

7.5. Threat: Weak TLS Configurations

Even when using TLS to secure the TCPCL session, the actual cipher suite negotiated between the TLS peers can be insecure. Recommendations for using cipher suites are included in [BCP 195](#) [\[RFC7525\]](#). It is up to security policies within each TCPCL entity to ensure that the negotiated TLS cipher suite meets transport security requirements.

7.6. Threat: Untrusted End-Entity Certificate

The authentication method discussed in [Section 4.4.4](#) uses end-entity certificates chained to a trusted root CA. During a TLS handshake, either entity can send a certificate set that does not contain the full chain, possibly excluding intermediate or root CAs. In an environment where peers are known to already contain needed root and intermediate CAs, there is no need to include those CAs, but this carries the risk of an entity not actually having one of the needed CAs.

7.7. Threat: Certificate Validation Vulnerabilities

Even when TLS itself is operating properly, an attacker can attempt to exploit vulnerabilities within certificate check algorithms or configuration to establish a secure TCPCL session using an invalid certificate. A BPA treats the peer node ID within a TCPCL session as authoritative, and exploitation via an invalid certificate could lead to bundle data leaking and/or denial of service to the node ID being impersonated.

There are many reasons, as described in [RFC5280] and [RFC6125], why a certificate can fail to validate, including using the certificate outside of its valid time interval, using purposes for which it was not authorized, or using it after it has been revoked by its CA. Validating a certificate is a complex task and can require network connectivity outside of the primary TCPCL network path(s) if a mechanism such as OCSP [RFC6960] is used by the CA. The configuration and use of particular certificate validation methods are outside of the scope of this document.

7.8. Threat: Symmetric Key Limits

Even with a secure block cipher and securely established session keys, there are limits to the amount of plaintext that can be safely encrypted with a given set of keys, as described in [AEAD-LIMITS]. When permitted by the negotiated TLS version (see [RFC8446]), it is advisable to take advantage of session key updates to avoid those limits.

7.9. Threat: BP Node Impersonation

The certificates exchanged by TLS enable authentication of the peer DNS name and node ID, but it is possible that either a peer does not provide a valid certificate or the certificate does not validate either the DNS-ID/IPADDR-ID or NODE-ID of the peer (see Section 3.4). Having a CA-validated certificate does not alone guarantee the identity of the network host or BP node from which the certificate is provided; additional validation procedures as provided in Section 4.4.4 bind the DNS-ID/IPADDR-ID or NODE-ID based on the contents of the certificate.

The DNS-ID/IPADDR-ID validation is a weaker form of authentication, because even if a peer is operating on an authenticated network DNS name or IP address it can provide an invalid node ID and cause bundles to be "leaked" to an invalid node. Especially in DTN environments, network names and addresses of nodes can be time-variable, so binding a certificate to a node ID results in a more stable identity.

NODE-ID validation ensures that the peer to which a bundle is transferred is in fact the node that the BPA expects it to be. In circumstances where certificates can only be issued to DNS names, node ID validation is not possible, but it could be reasonable to assume that a trusted host is not going to present an invalid node ID. Determining when a DNS-ID/IPADDR-ID authentication can be trusted to validate a node ID is also a policy matter outside of the scope of this document.

One mitigation regarding arbitrary entities with valid PKIX certificates impersonating arbitrary node IDs is the use of the PKIX EKU key purpose `id-kp-bundleSecurity` ([Section 4.4.2.1](#)). When this EKU is present in the certificate, it represents a stronger assertion that the private key holder should in fact be trusted to operate as a bundle node.

7.10. Threat: Denial of Service

The behaviors described in this section all amount to a potential denial of service to a TCPCL entity. The denial of service could be limited to an individual TCPCL 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 TCPCL sessions and delaying the sending of protocol-required data. The victim entity can block TCP connections from network peers that are thought to behave incorrectly within the TCPCL.

An entity can send a large amount of data over a TCPCL 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 TCPCL 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 TCP connection that will cause unnecessary TCP retransmissions or connection resets, effectively denying the use of the overlying TCPCL session.

7.11. Mandatory-to-Implement TLS

Following IETF best current practice, TLS is mandatory to implement for all TCPCL implementations but TLS is optional to use for a given TCPCL session. The policy recommendations in [Sections 4.2](#) and [4.3](#) both enable TLS and require TLS, but entities are permitted to disable and not require TLS based on local configuration. The configuration to enable or require TLS for an entity or a session is outside of the scope of this document. The configuration to disable TLS is different from the threat of TLS Stripping as described in [Section 7.4](#).

7.12. Alternate Uses of TLS

This specification makes use of PKIX certificate validation and authentication within TLS. There are alternate uses of TLS that are not necessarily incompatible with the security goals of this specification but that are outside of the scope of this document. The following subsections give examples of alternate TLS uses.

7.12.1. TLS without Authentication

In environments where PKI is available but there are restrictions on the issuance of certificates (including the contents of certificates), it may be possible to make use of TLS in a way that authenticates only the passive entity of a TCPCL session or that does not authenticate either entity. Using TLS in a way that does not successfully authenticate some claim of both peer entities of a TCPCL session is outside of the scope of this document but does have properties similar to the opportunistic security model [[RFC7435](#)].

7.12.2. Non-certificate TLS Use

In environments where PKI is unavailable, alternate uses of TLS that do not require certificates such as pre-shared key (PSK) authentication [[RFC5489](#)] and the use of raw public keys [[RFC7250](#)] are available and can be used to ensure confidentiality within the TCPCL. Using non-PKI node authentication methods is outside of the scope of this document.

7.13. Predictability of Transfer IDs

The only requirement on Transfer IDs is that they be unique within each session from the sending peer only. The trivial algorithm of the first transfer starting at zero and later transfers incrementing by one causes absolutely predictable Transfer IDs. Even when a TCPCL session is not TLS secured and there is an on-path attacker causing denial of service with XFER_REFUSE messages, it is not possible to preemptively refuse a transfer, so there is no benefit in having unpredictable Transfer IDs within a session.

8. IANA Considerations

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

Some of the registries have been defined as version specific for TCPCLv4, and these registries reuse some or all codepoints from TCPCLv3. This was done to disambiguate the use of these codepoints between TCPCLv3 and TCPCLv4 while preserving the semantics of some of the codepoints.

8.1. Port Number

Within the "Service Name and Transport Protocol Port Number Registry" [[IANA-PORTS](#)], TCP port number 4556 had previously been assigned as the default port for the TCPCL; see [[RFC7242](#)]. This assignment is unchanged by TCPCL version 4, but the assignment reference has been updated to

point to this specification. Each TCPCL entity identifies its TCPCL protocol version in its initial contact (see Sections 3.2 and 8.2), so there is no ambiguity regarding what protocol is being used. The related assignments for UDP and DCCP port 4556 (both registered by [RFC7122]) are unchanged.

| Parameter | Value |
|------------------------|----------------------------|
| Service Name: | dtn-bundle |
| Transport Protocol(s): | TCP |
| Assignee: | IESG (iesg@ietf.org) |
| Contact: | IESG (iesg@ietf.org) |
| Description: | DTN Bundle TCP CL Protocol |
| Reference: | This specification |
| Port Number: | 4556 |

Table 10: TCP Port Number for the TCPCL

8.2. Protocol Versions

IANA has registered the following value in the "Bundle Protocol TCP Convergence-Layer Version Numbers" registry [RFC7242].

| Value | Description | Reference |
|-------|-------------|--------------------|
| 4 | TCPCLv4 | This specification |

Table 11: New TCPCL Version Number

8.3. Session Extension Types

Under the "Bundle Protocol" registry [IANA-BUNDLE], IANA has created the "Bundle Protocol TCP Convergence-Layer Version 4 Session Extension Types" registry and populated 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 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 12: Session Extension Type Codes

8.4. Transfer Extension Types

Under the "Bundle Protocol" registry [[IANA-BUNDLE](#)], IANA has created the "Bundle Protocol TCP Convergence-Layer Version 4 Transfer Extension Types" registry and populated it with the contents of [Table 13](#). 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 | Transfer Length Extension |
| 0x0002-0x7FFF | Unassigned |
| 0x8000-0xFFFF | Reserved for Private or Experimental Use |

Table 13: Transfer Extension Type Codes

8.5. Message Types

Under the "Bundle Protocol" registry [[IANA-BUNDLE](#)], IANA has created the "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" registry and populated it with the contents of [Table 14](#). 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 TCPCL 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. TCPCL 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 TCP connection; not even a clean termination can be done at that point.

| Code | Message Type |
|-----------|--|
| 0x00 | Reserved |
| 0x01 | XFER_SEGMENT |
| 0x02 | XFER_ACK |
| 0x03 | XFER_REFUSE |
| 0x04 | KEEPALIVE |
| 0x05 | SESS_TERM |
| 0x06 | MSG_REJECT |
| 0x07 | SESS_INIT |
| 0x08-0xEF | Unassigned |
| 0xF0-0xFF | Reserved for Private or Experimental Use |

Table 14: Message Type Codes

8.6. XFER_REFUSE Reason Codes

Under the "Bundle Protocol" registry [[IANA-BUNDLE](#)], IANA has created the "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" registry and populated it with the contents of [Table 15](#). 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).

| Code | Refusal Reason |
|-----------|--|
| 0x00 | Unknown |
| 0x01 | Completed |
| 0x02 | No Resources |
| 0x03 | Retransmit |
| 0x04 | Not Acceptable |
| 0x05 | Extension Failure |
| 0x06 | Session Terminating |
| 0x07-0xEF | Unassigned |
| 0xF0-0xFF | Reserved for Private or Experimental Use |

Table 15: XFER_REFUSE Reason Codes

8.7. SESS_TERM Reason Codes

Under the "Bundle Protocol" registry [[IANA-BUNDLE](#)], IANA has created the "Bundle Protocol TCP Convergence-Layer Version 4 SESS_TERM Reason Codes" registry and populated it with the contents of [Table 16](#). 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).

| Code | Termination Reason |
|------|--------------------|
| 0x00 | Unknown |
| 0x01 | Idle timeout |
| 0x02 | Version mismatch |
| 0x03 | Busy |

| Code | Termination Reason |
|-----------|--|
| 0x04 | Contact Failure |
| 0x05 | Resource Exhaustion |
| 0x06-0xEF | Unassigned |
| 0xF0-0xFF | Reserved for Private or Experimental Use |

Table 16: *SESS_TERM Reason Codes*

8.8. MSG_REJECT Reason Codes

Under the "Bundle Protocol" registry [[IANA-BUNDLE](#)], IANA has created the "Bundle Protocol TCP Convergence-Layer Version 4 MSG_REJECT Reason Codes" registry and populated it with the contents of [Table 17](#). 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 new 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 TCPCL 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).

| Code | Rejection Reason |
|-----------|--|
| 0x00 | Reserved |
| 0x01 | Message Type Unknown |
| 0x02 | Message Unsupported |
| 0x03 | Message Unexpected |
| 0x04-0xEF | Unassigned |
| 0xF0-0xFF | Reserved for Private or Experimental Use |

Table 17: *MSG_REJECT Reason Codes*

8.9. Object Identifier for PKIX Module Identifier

IANA has registered the following in the "SMI Security for PKIX Module Identifier" registry [[IANA-SMI](#)] for identifying the module described in [Appendix B](#).

| Decimal | Description | References |
|---------|-------------------------|--------------------|
| 103 | id-mod-dtn-tcpclv4-2021 | This specification |

Table 18: New SMI Security Module

8.10. Object Identifier for PKIX Other Name Forms

IANA has registered the following in the "SMI Security for PKIX Other Name Forms" registry [IANA-SMI] for identifying bundle endpoint IDs:

| Decimal | Description | References |
|---------|-----------------|--------------------|
| 11 | id-on-bundleEID | This specification |

Table 19: New PKIX Other Name Form

The formal structure of the associated Other Name Form is provided in [Appendix B](#). The use of this OID is defined in Sections [4.4.1](#) and [4.4.2](#).

8.11. Object Identifier for PKIX Extended Key Usage

IANA has registered the following in the "SMI Security for PKIX Extended Key Purpose" registry [IANA-SMI] for securing BP bundles.

| Decimal | Description | References |
|---------|----------------------|--------------------|
| 35 | id-kp-bundleSecurity | This specification |

Table 20: New PKIX Extended Key Purpose

The formal definition of this EKU is provided in [Appendix B](#). The use of this OID is defined in [Section 4.4.2](#).

9. References

9.1. 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/>>.
- [IANA-SMI] IANA, "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", <<https://www.iana.org/assignments/smi-numbers/>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.

-
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [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>>.
- [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>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <<https://www.rfc-editor.org/info/rfc6960>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [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>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [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>>.
-

- [X.680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <<https://www.itu.int/rec/T-REC-X.680-202102-I/en>>.

9.2. Informative References

- [AEAD-LIMITS] Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", August 2017, <<https://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf>>.
- [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <<https://www.rfc-editor.org/info/rfc2595>>.
- [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>>.
- [RFC4511] Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI 10.17487/RFC4511, June 2006, <<https://www.rfc-editor.org/info/rfc4511>>.
- [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>>.
- [RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, DOI 10.17487/RFC5489, March 2009, <<https://www.rfc-editor.org/info/rfc5489>>.
- [RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <<https://www.rfc-editor.org/info/rfc5912>>.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<https://www.rfc-editor.org/info/rfc6698>>.
- [RFC7122] Kruse, H., Jero, S., and S. Ostermann, "Datagram Convergence Layers for the Delay- and Disruption-Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP)", RFC 7122, DOI 10.17487/RFC7122, March 2014, <<https://www.rfc-editor.org/info/rfc7122>>.
- [RFC7242] Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol", RFC 7242, DOI 10.17487/RFC7242, June 2014, <<https://www.rfc-editor.org/info/rfc7242>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.

- [RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <<https://www.rfc-editor.org/info/rfc7435>>.
- [RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <<https://www.rfc-editor.org/info/rfc7457>>.
- [RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <<https://www.rfc-editor.org/info/rfc8555>>.
- [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>>.
- [DTN-BIBECT] Burleigh, S., "Bundle-in-Bundle Encapsulation", Work in Progress, Internet-Draft, draft-ietf-dtn-bibect-03, 18 February 2020, <<https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bibect-03>>.

Appendix A. Significant Changes from RFC 7242

The areas in which changes from [RFC7242] have been made to existing headers and messages are as follows:

- Split Contact Header into pre-TLS protocol negotiation and SESS_INIT parameter negotiation. The Contact Header is now fixed length.
- Changed Contact Header content to limit number of negotiated options.
- Added session option to negotiate maximum segment size (per each direction).
- Renamed "endpoint ID" to "node ID" to conform with BPv7 terminology.
- Added session extension capability.
- Added transfer extension capability. Moved transfer total length into an extension item.
- Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.
- Pointed out that segments of all new IANA registries are reserved for private/experimental use.
- Expanded Message Header to octet-aligned fields instead of bit-packing.
- Added a bundle transfer identification number to all bundle-related messages (XFER_SEGMENT, XFER_ACK, XFER_REFUSE).
- Added flags in XFER_ACK to mirror flags from XFER_SEGMENT.
- Removed all uses of Self-Delimiting Numeric Value (SDNV) fields and replaced with fixed-bit-length (network byte order) fields.
- Renamed SHUTDOWN to SESS_TERM to deconflict term "shutdown" related to TCP connections.
- Removed the notion of a reconnection delay parameter.

The areas in which extensions from [\[RFC7242\]](#) have been made as new messages and codes are as follows:

- Added MSG_REJECT message to indicate that an unknown or unhandled message was received.
- Added TLS connection security mechanism.
- Added "Not Acceptable", "Extension Failure", and "Session Terminating" XFER_REFUSE reason codes.
- Added "Contact Failure" (contact negotiation failure) and "Resource Exhaustion" SESS_TERM reason codes.

Appendix B. ASN.1 Module

The following ASN.1 module formally specifies the BundleEID structure, its Other Name Form, and the bundleSecurity EKU, using ASN.1 syntax per [\[X.680\]](#). This specification uses the ASN.1 definitions from [\[RFC5912\]](#) with the 2002 ASN.1 notation used in that document.

```

<CODE BEGINS>
DTN-TCPCLv4-2021
  { iso(1) identified-organization(3) dod(6)
    internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-dtn-tcpclv4-2021(103) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
  OTHER-NAME
  FROM PKIX1Implicit-2009 -- [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-implicit-02(59) }

  id-pkix
  FROM PKIX1Explicit-2009 -- [RFC5912]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-explicit-02(51) } ;

id-kp OBJECT IDENTIFIER ::= { id-pkix 3 }
id-on OBJECT IDENTIFIER ::= { id-pkix 8 }

DTNOtherNames OTHER-NAME ::= { on-bundleEID, ... }

-- The otherName definition for BundleEID
on-bundleEID OTHER-NAME ::= {
  BundleEID IDENTIFIED BY { id-on-bundleEID }
}

id-on-bundleEID OBJECT IDENTIFIER ::= { id-on 11 }

-- Same encoding as GeneralName of uniformResourceIdentifier
BundleEID ::= IA5String

-- The Extended Key Usage key for bundle security
id-kp-bundleSecurity OBJECT IDENTIFIER ::= { id-kp 35 }

END

<CODE ENDS>

```

Appendix C. Example of the BundleEID Other Name Form

This non-normative example demonstrates an otherName with a name form of BundleEID to encode the node ID "dtn://example/".

The hexadecimal form of the DER encoding of the otherName is as follows:

```
a01c06082b0601050507080ba010160e64746e3a2f2f6578616d706c652f
```

And the text decoding in [Figure 28](#) is an output of Peter Gutmann's "dumpasn1" program.

```
0 28: [0] {
2  8:  OBJECT IDENTIFIER '1 3 6 1 5 5 7 8 11'
12 16:  [0] {
14 14:    IA5String 'dtn://example/'
      :    }
      :  }
```

Figure 28: Visualized Decoding of the on-bundleEID

Acknowledgments

This specification is based on comments regarding the implementation of [\[RFC7242\]](#) as provided by Scott Burleigh.

The ASN.1 module and its Other Name Form are based on a recommendation provided by Russ Housley.

Authors' Addresses

Brian Sipos

RKF Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
United States of America
Email: brian.sipos+ietf@gmail.com

Michael Demmer

Email: demmer@gmail.com

Jörg Ott

Technical University of Munich
Department of Informatics
Chair of Connected Mobility
Boltzmannstrasse 3
DE-85748 Garching
Germany
Email: ott@in.tum.de

Simon Perreault

LogMeIn

410 boulevard Charest Est

Suite 250

Quebec QC G1K 8G3

Canada

Email: simon.perreault@logmein.com