

Delay Tolerant Networking  
Internet-Draft  
Obsoletes: [7242](#) (if approved)  
Intended status: Standards Track  
Expires: May 9, 2019

B. Sipos  
RKF Engineering  
M. Demmer  
UC Berkeley  
J. Ott  
Aalto University  
S. Perreault  
November 5, 2018

## **Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4 draft-ietf-dtn-tcpclv4-10**

### Abstract

This document describes a revised protocol for the TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). The protocol revision is based on implementation issues in the original TCPCL Version 3 of [RFC7242](#) and updates to the Bundle Protocol contents, encodings, and convergence layer requirements in Bundle Protocol Version 7. Specifically, the TCPCLv4 uses CBOR-encoded BPv7 bundles as its service data unit being transported and provides a reliable transport of such bundles. Several new IANA registries are defined for TCPCLv4 which define some behaviors inherited from TCPCLv3 but with updated encodings and/or semantics.

### Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on May 9, 2019.

### Copyright Notice

Copyright (c) 2018 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">3</a>
<a href="#">1.1.</a>	<a href="#">Convergence Layer Services</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Requirements Language</a>	<a href="#">6</a>
<a href="#">2.1.</a>	<a href="#">Definitions Specific to the TCPCL Protocol</a>	<a href="#">6</a>
<a href="#">3.</a>	<a href="#">General Protocol Description</a>	<a href="#">9</a>
<a href="#">3.1.</a>	<a href="#">TCPCL Session Overview</a>	<a href="#">9</a>
<a href="#">3.2.</a>	<a href="#">TCPCL States and Transitions</a>	<a href="#">11</a>
<a href="#">3.3.</a>	<a href="#">Transfer Segmentation Policies</a>	<a href="#">16</a>
<a href="#">3.4.</a>	<a href="#">Example Message Exchange</a>	<a href="#">17</a>
<a href="#">4.</a>	<a href="#">Session Establishment</a>	<a href="#">19</a>
<a href="#">4.1.</a>	<a href="#">TCP Connection</a>	<a href="#">19</a>
<a href="#">4.2.</a>	<a href="#">Contact Header</a>	<a href="#">19</a>
<a href="#">4.3.</a>	<a href="#">Contact Validation and Negotiation</a>	<a href="#">20</a>
<a href="#">4.4.</a>	<a href="#">Session Security</a>	<a href="#">21</a>
<a href="#">4.4.1.</a>	<a href="#">TLS Handshake Result</a>	<a href="#">22</a>
<a href="#">4.4.2.</a>	<a href="#">Example TLS Initiation</a>	<a href="#">22</a>
<a href="#">4.5.</a>	<a href="#">Message Type Codes</a>	<a href="#">23</a>
<a href="#">4.6.</a>	<a href="#">Session Initialization Message (SESS_INIT)</a>	<a href="#">24</a>
<a href="#">4.6.1.</a>	<a href="#">Session Extension Items</a>	<a href="#">26</a>
<a href="#">4.7.</a>	<a href="#">Session Parameter Negotiation</a>	<a href="#">27</a>
<a href="#">5.</a>	<a href="#">Established Session Operation</a>	<a href="#">28</a>
<a href="#">5.1.</a>	<a href="#">Upkeep and Status Messages</a>	<a href="#">28</a>
<a href="#">5.1.1.</a>	<a href="#">Session Upkeep (KEEPALIVE)</a>	<a href="#">28</a>
<a href="#">5.1.2.</a>	<a href="#">Message Rejection (MSG_REJECT)</a>	<a href="#">29</a>
<a href="#">5.2.</a>	<a href="#">Bundle Transfer</a>	<a href="#">30</a>
<a href="#">5.2.1.</a>	<a href="#">Bundle Transfer ID</a>	<a href="#">30</a>
<a href="#">5.2.2.</a>	<a href="#">Transfer Initialization (XFER_INIT)</a>	<a href="#">31</a>
<a href="#">5.2.3.</a>	<a href="#">Data Transmission (XFER_SEGMENT)</a>	<a href="#">34</a>
<a href="#">5.2.4.</a>	<a href="#">Data Acknowledgments (XFER_ACK)</a>	<a href="#">35</a>
<a href="#">5.2.5.</a>	<a href="#">Transfer Refusal (XFER_REFUSE)</a>	<a href="#">36</a>
<a href="#">6.</a>	<a href="#">Session Termination</a>	<a href="#">38</a>
<a href="#">6.1.</a>	<a href="#">Session Termination Message (SESS_TERM)</a>	<a href="#">38</a>
<a href="#">6.2.</a>	<a href="#">Idle Session Shutdown</a>	<a href="#">41</a>
<a href="#">7.</a>	<a href="#">Implementation Status</a>	<a href="#">41</a>
<a href="#">8.</a>	<a href="#">Security Considerations</a>	<a href="#">41</a>
<a href="#">9.</a>	<a href="#">IANA Considerations</a>	<a href="#">43</a>



<a href="#">9.1.</a>	Port Number . . . . .	<a href="#">43</a>
<a href="#">9.2.</a>	Protocol Versions . . . . .	<a href="#">43</a>
<a href="#">9.3.</a>	Session Extension Types . . . . .	<a href="#">44</a>
<a href="#">9.4.</a>	Transfer Extension Types . . . . .	<a href="#">44</a>
<a href="#">9.5.</a>	Message Types . . . . .	<a href="#">45</a>
<a href="#">9.6.</a>	XFER_REFUSE Reason Codes . . . . .	<a href="#">46</a>
<a href="#">9.7.</a>	SESS_TERM Reason Codes . . . . .	<a href="#">47</a>
<a href="#">9.8.</a>	MSG_REJECT Reason Codes . . . . .	<a href="#">48</a>
<a href="#">10.</a>	Acknowledgments . . . . .	<a href="#">49</a>
<a href="#">11.</a>	References . . . . .	<a href="#">49</a>
<a href="#">11.1.</a>	Normative References . . . . .	<a href="#">49</a>
<a href="#">11.2.</a>	Informative References . . . . .	<a href="#">49</a>
<a href="#">Appendix A.</a>	Significant changes from <a href="#">RFC7242</a> . . . . .	<a href="#">50</a>
	Authors' Addresses . . . . .	<a href="#">51</a>

## [1.](#) Introduction

This document describes the TCP-based convergence-layer protocol for Delay-Tolerant Networking. Delay-Tolerant Networking 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 Network 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) [[I-D.ietf-dtn-bpbis](#)], 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. For the remainder of this document, the abbreviation "TCPCL" without the version suffix refers to TCPCLv4.

The locations of the TCPCL and the BP 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 TCPCL as its convergence layer, both BP and TCPCL reside at the application layer of the Internet model.



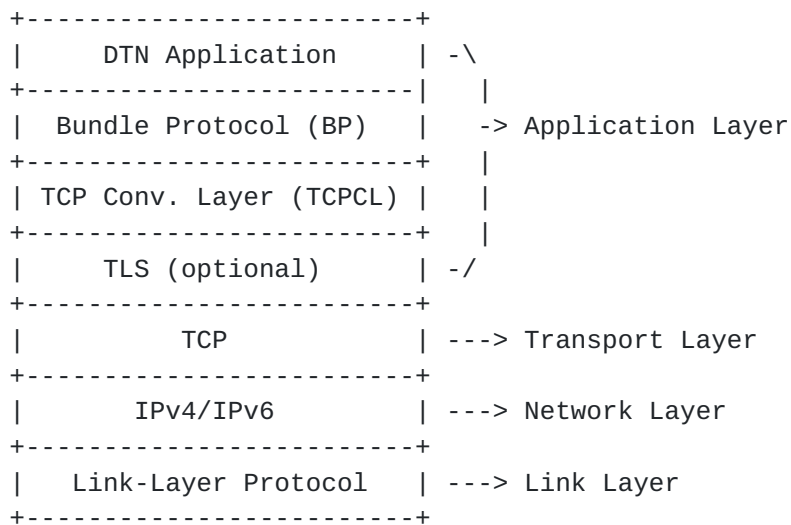


Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol above the Internet Protocol Stack

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

- o The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [[RFC5050](#)] and [[I-D.ietf-dtn-bpbis](#)]. This includes the concept of bundle fragmentation or bundle encapsulation. The TCPCL transfers bundles as opaque data blocks.
- o Mechanisms for locating or identifying other bundle entities within an internet.

### **[1.1. Convergence Layer Services](#)**

This version of the TCPCL provides the following services to support the overlaying Bundle Protocol agent. In all cases, this is not an API definition but a logical description of how the CL may interact with the BP agent. Each of these interactions may be associated with any number of additional metadata items as necessary to support the operation of the CL or BP agent.

**Attempt Session** The TCPCL allows a BP agent to pre-emptively 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 BP agent.

**Terminate Session** The TCPCL allows a BP agent to pre-emptively terminate an established TCPCL session with a peer entity. The terminate request is on a per-session basis.



**Session State Changed** The TCPCL supports indication when the session state changes. The top-level session states indicated are:

**Contact Negotiating:** A TCP connection has been made (as either 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.

**Closing:** The entity received a SESS\_TERM message and is in the closing state.

**Terminated:** The session has finished normal termination sequencing..

**Failed:** The session ended without normal termination sequencing.

**Session Idle Changed** The TCPCL supports indication when the live/idle sub-state changes. This occurs only when the top-level session state is Established. Because TCPCL transmits serially over a TCP connection, it suffers from "head of queue blocking" this indication provides information about when a session is available for immediate transfer start.

**Begin Transmission** The principal purpose of the TCPCL is to allow a BP agent to transmit bundle data over an established TCPCL session. Transmission request is on a per-session basis, the CL does not necessarily perform any per-session or inter-session queueing. Any queueing of transmissions is the obligation of the BP agent.

**Transmission Success** The TCPCL supports positive indication when a bundle has been fully transferred to a peer entity.

**Transmission Intermediate Progress** The TCPCL supports positive indication of intermediate progress of transferr to a peer entity. This intermediate progress is at the granularity of each transferred segment.

**Transmission Failure** The TCPCL supports positive indication of certain reasons for bundle transmission failure, notably when the peer entity rejects the bundle or when a TCPCL session ends before transferr success. The TCPCL itself does not have a notion of transfer timeout.





**Reception Initialized** The TCPCL supports indication to the receiver just before any transmission data is sent. This corresponds to reception of the XFER\_INIT message.

**Interrupt Reception** The TCPCL allows a BP agent 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 supports positive indication when a bundle has been fully transferred from a peer entity.

**Reception Intermediate Progress** The TCPCL supports positive indication of intermediate progress of transfer from the peer entity. This intermediate progress is at the granularity of each transferred segment. Intermediate reception indication allows a BP agent the chance to inspect bundle header contents before the entire bundle is available, and thus supports the "Reception Interruption" capability.

**Reception Failure** The TCPCL supports positive indication of 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.

## **2. Requirements Language**

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [\[RFC2119\]](#).

### **2.1. Definitions Specific to the TCPCL Protocol**

This section contains definitions specific to the TCPCL protocol.

**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. 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. The lifetime of a TCPCL session is bound to the lifetime of an underlying TCP connection. A TCPCL session is terminated when the TCP connection ends, due either to one or both entities actively terminating the TCP connection or due to network errors causing a failure of the TCP connection. 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 uni-directional user-data path within a TCPCL Session. Messages sent over a transfer stream are serialized, meaning that one set of user data must complete its transmission prior to another set of user data being transmitted over the same transfer stream. Each uni-directional 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 TCPCL is identified by a Transfer ID number which is unique only to a single direction within a single Session.

**Transfer Segment:** A subset of a transfer of user data being communicated over a transfer stream.

**Idle Session:** A TCPCL session is idle while the only messages being transmitted or received are KEEPALIVE messages.

**Live Session:** A TCPCL session is live while any messages are being transmitted or received.

**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.

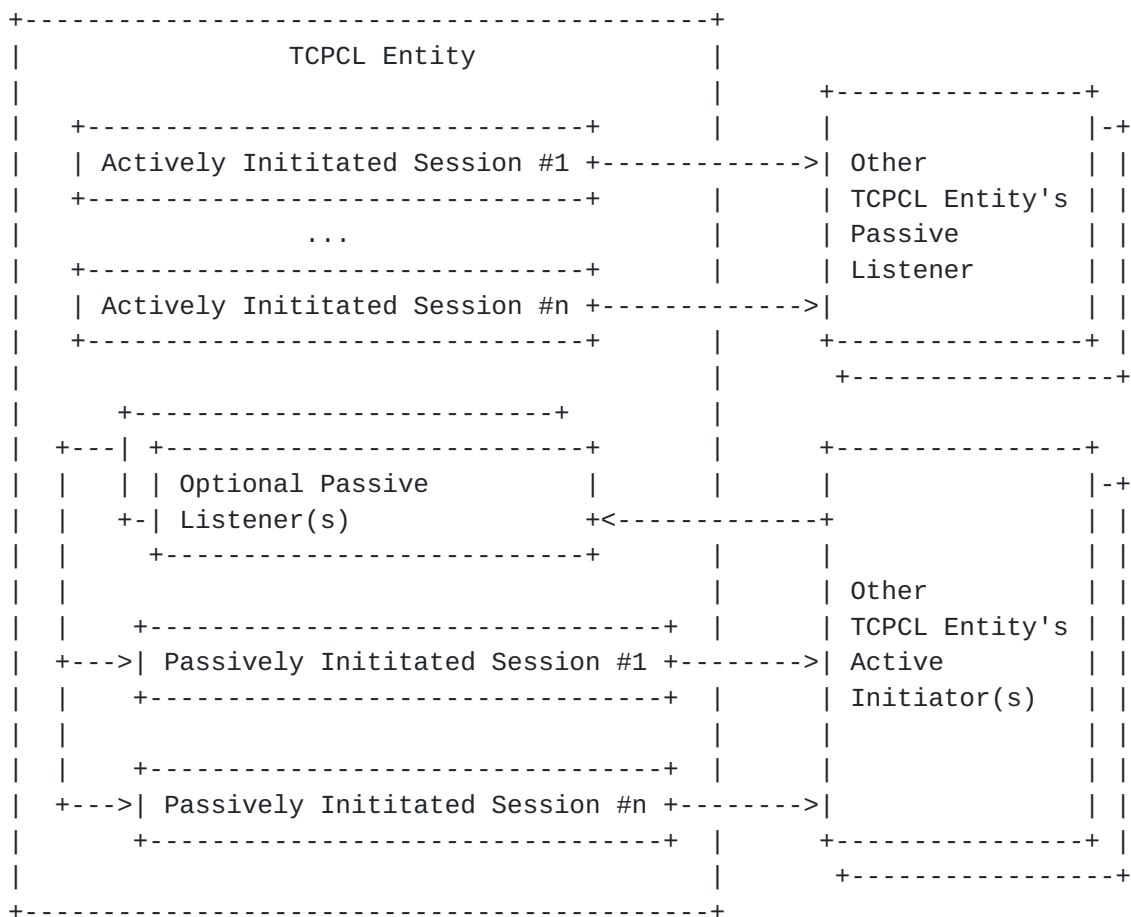


Figure 2: The relationships between TCPCL entities



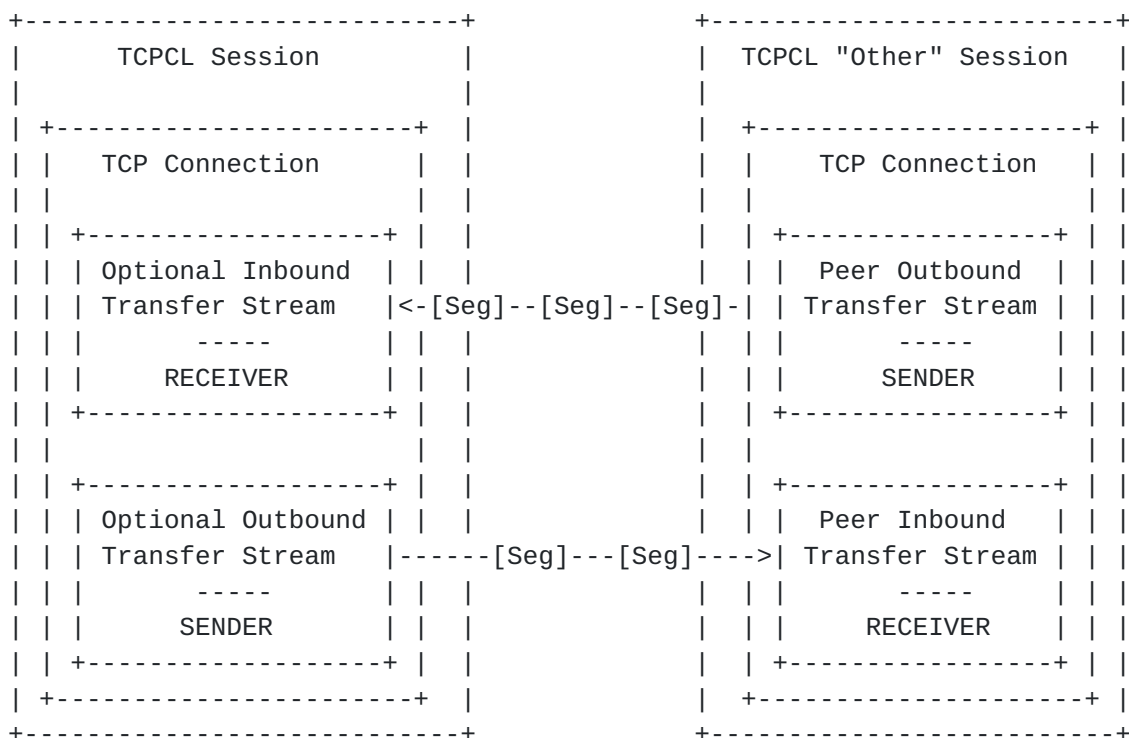


Figure 3: The relationship within a TCPCL Session of its two streams

### 3. General Protocol Description

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

#### 3.1. TCPCL Session Overview

First, one node 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 possibly initiate TLS security. 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 singleton endpoint identifier for each node (not the singleton Endpoint Identifier (EID) of any application running on the node) to denote the bundle-layer identity of each DTN node. This is used to assist in routing and forwarding messages (e.g. to prevent loops).

Once negotiated, the parameters of a TCPCL session cannot change and 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 an initialization (XFER\_INIT) message followed by one or more logical segments of data within an XFER\_SEGMENT message. Multiple bundles can be transmitted consecutively 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.

A feature of this protocol is for the receiving node to send acknowledgment (XFER\_ACK) messages as bundle data segments arrive. The rationale behind these acknowledgments is to enable the sender node to determine how much of the bundle has been received, so that in case the session is interrupted, it can perform reactive fragmentation to avoid re-sending the already transmitted part of the bundle. In addition, there is no explicit flow control on the TCPCL layer.

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 already has 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 node live-ness information during otherwise message-less time intervals.

A SESS\_TERM message is used to start the closing of a TCPCL session (see [Section 6.1](#)). During shutdown 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](#)). It is an implementation matter to determine whether or not to close a TCPCL session while there are no transfers queued or in-progress.

Once a session is established, 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 bi-directional mode of communication. Note that in the case of concurrent bidirectional transmission, acknowledgment segments MAY be interleaved with data segments.

### **[3.2.](#) TCPCL States and Transitions**

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

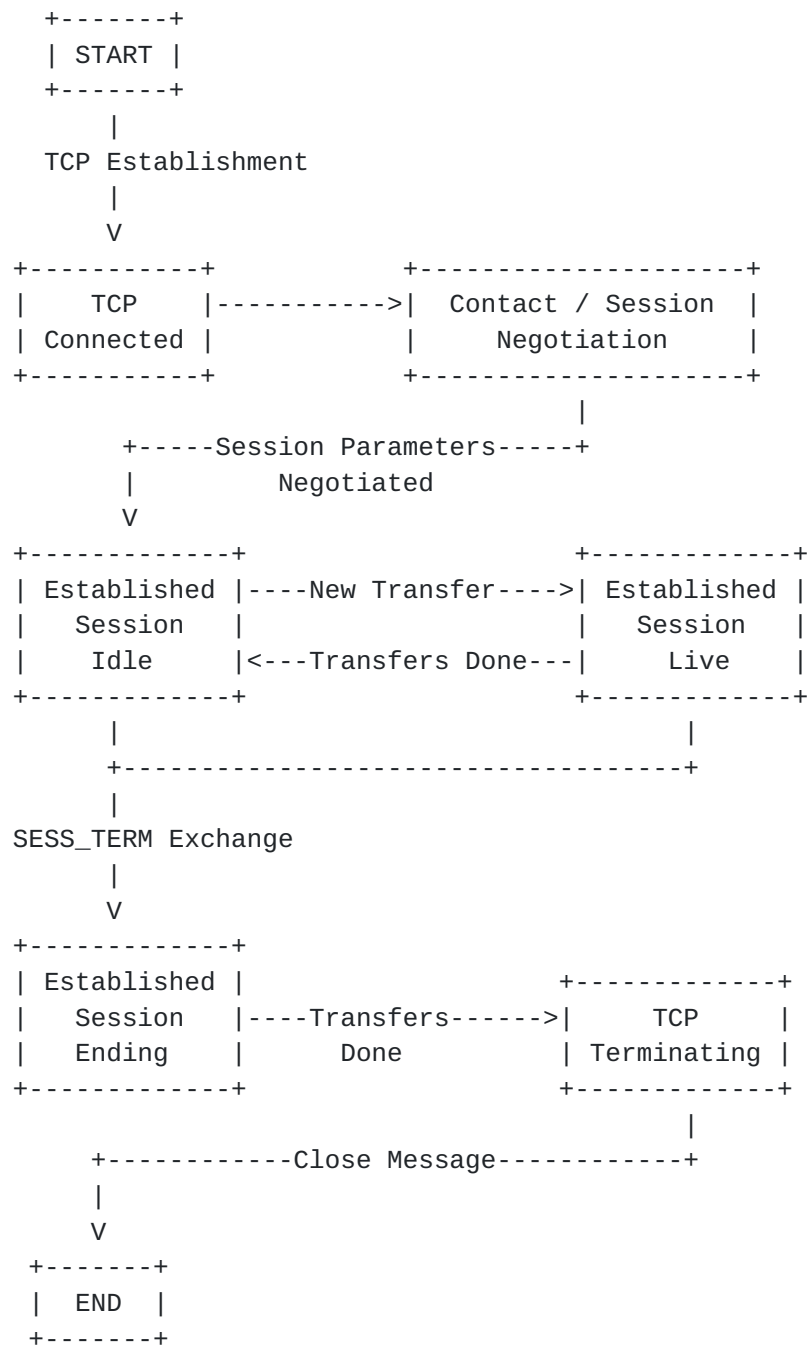


Figure 4: Top-level states of a TCPCL session

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 "Closing" means no new transfers will be allowed.

The contact negotiation sequencing is performed either as the active or passive peer, and is illustrated in Figure 5 and Figure 6 respectively which both share the data validation and analyze final states of Figure 7.

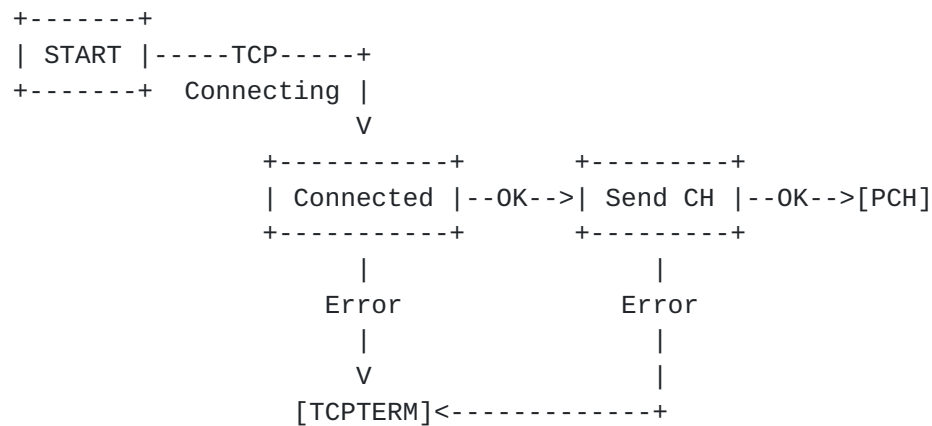


Figure 5: Contact Initiation as Active peer



Figure 6: Contact Initiation as Passive peer





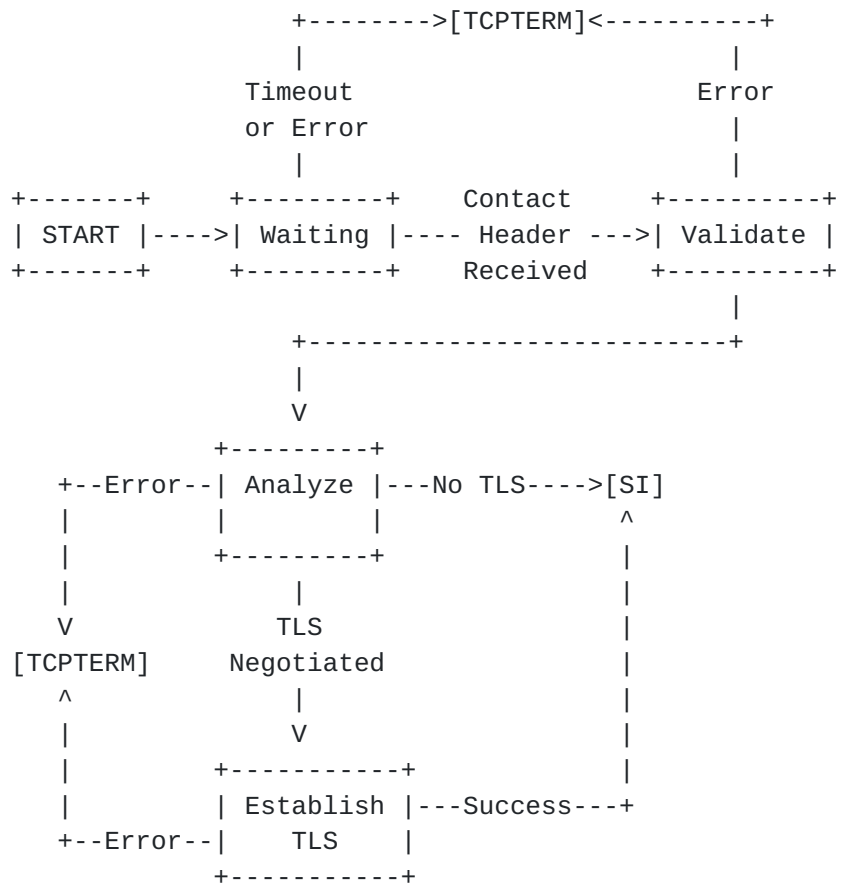


Figure 7: Processing of Contact Header (PCH)

The session negotiation sequencing is performed either as the active or passive peer, and is illustrated in Figure 8 and Figure 9 respectively which both share the data validation and analyze final states of Figure 10.

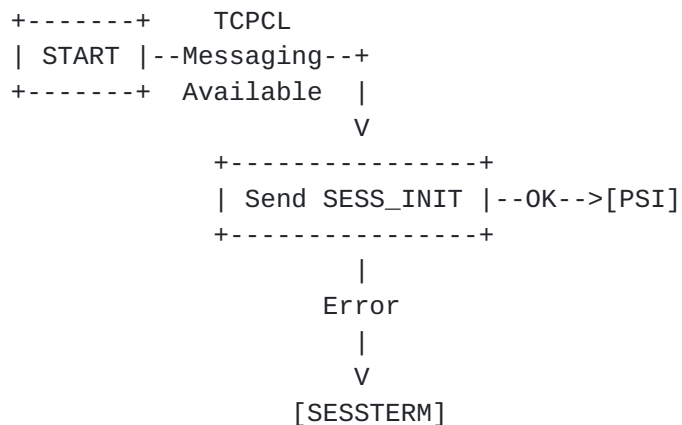


Figure 8: Session Initiation as Active peer



```

+-----+      TCPCL
| START |---Messaging-->[PSI]
+-----+      Available

```

Figure 9: Session Initiation as Passive peer

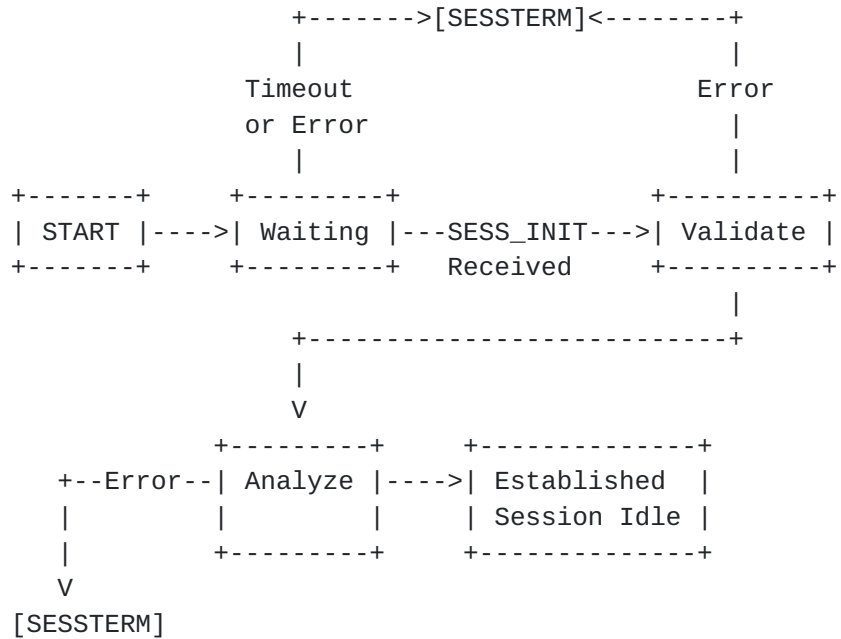


Figure 10: Processing of Session Initiation (PSI)

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 Figure 11 and Figure 12 respectively.

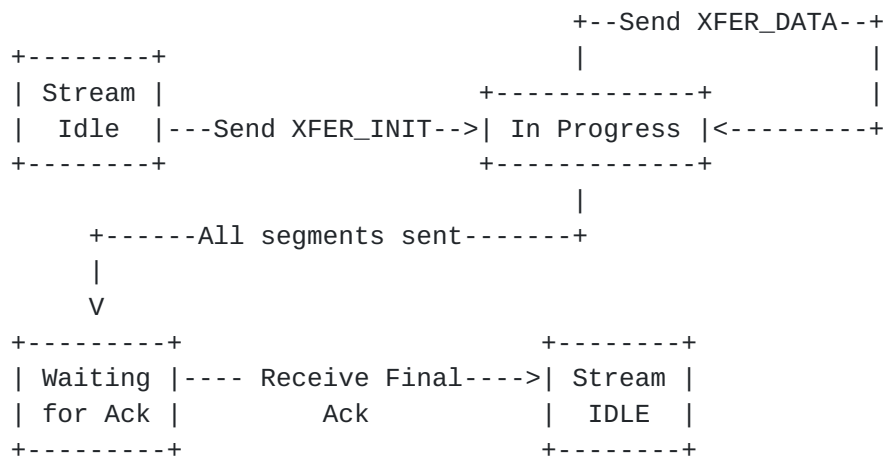


Figure 11: Transfer sender states



Notes on transfer sending:

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

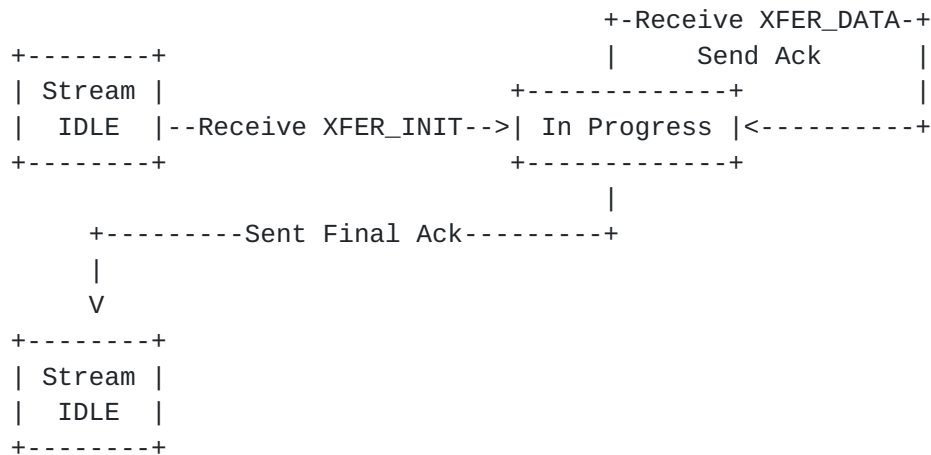


Figure 12: Transfer receiver states

### 3.3. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. A receiving node can set the Segment MRU in its contact header to determine the largest acceptable segment size, and a transmitting node 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 operating TCPCL, but guidance given here can be used to steer policy toward performance goals. It is 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 node is transmitting a single XFER\_SEGMENT message it is not able to transmit any XFER\_ACK or XFER\_REFUSE 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 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 associated with received transfers.

**Dynamic Segmentation** Even after negotiation of a Segment MRU for each receiving node, the actual transfer segmentation only needs to guarantee that any individual segment is no larger than that MRU. In a situation where network "goodput" 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 these two extremes. Different policies can be applied to each direction to/from any particular node. Additionally, future header and transfer extension types can apply further nuance to transfer policies and policy negotiation.

#### **3.4. Example Message Exchange**

The following figure 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 node MAY transmit multiple XFER\_SEGMENT messages without necessarily 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.





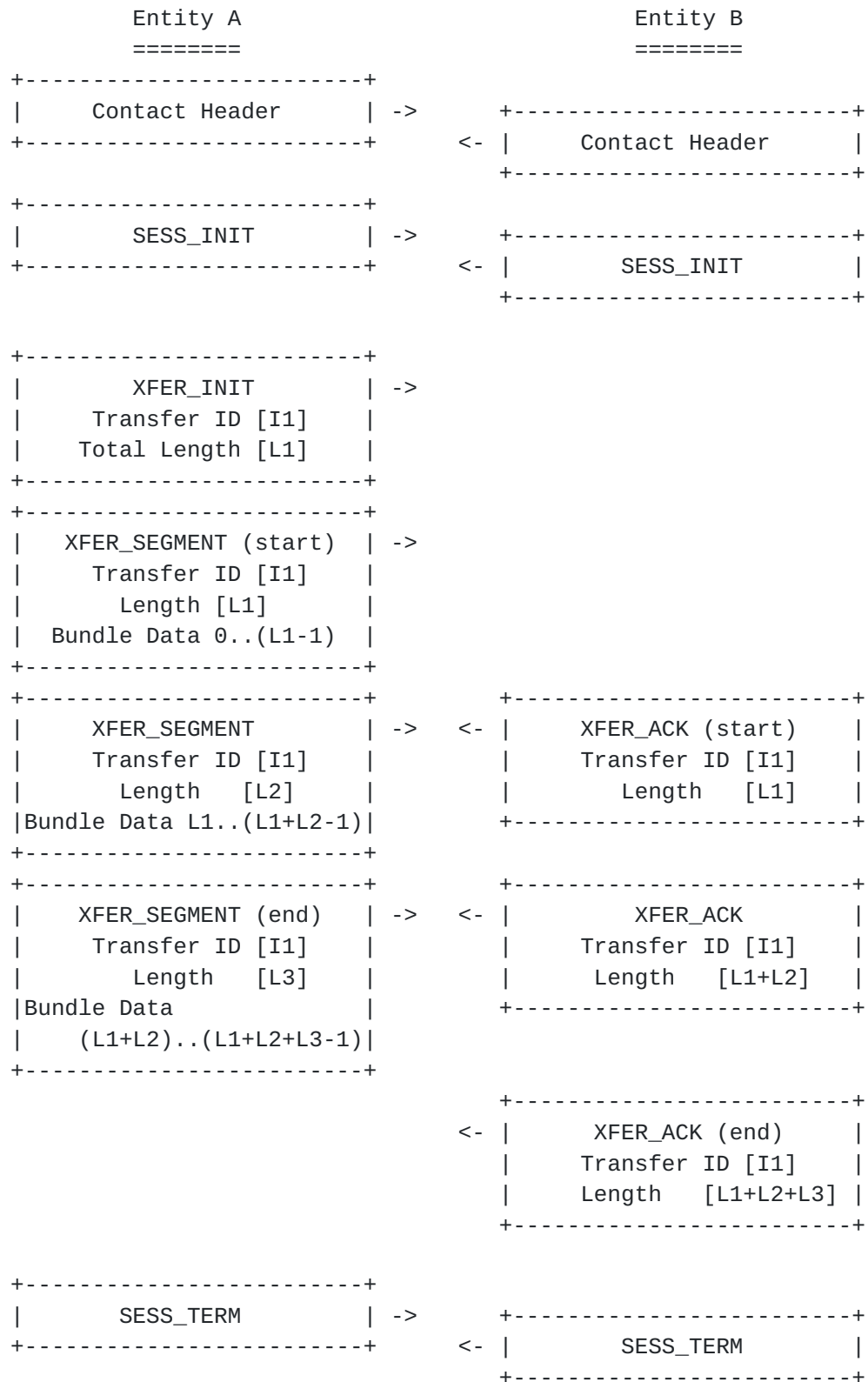


Figure 13: An example of the flow of protocol messages on a single TCP Session between two entities



## **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 MAY be opened proactively and maintained for as long as is possible given the network conditions, while other sessions MAY 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 TCP convergence layer. 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 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 re-attempt to establish the connection. If it does so, it MUST NOT overwhelm its target with repeated connection attempts. Therefore, the entity MUST retry the connection setup no earlier than some delay time from the last attempt, and it SHOULD use a (binary) exponential backoff mechanism to increase this delay in case of repeated failures.

Once a TCP connection is established, each entity MUST immediately transmit a contact header over the TCP connection. The format of the contact header is described in [Section 4.2](#).

### **4.2. Contact Header**

Once a TCP connection is established, both parties exchange a contact header. This section describes the format of the contact header and the meaning of its fields.

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:

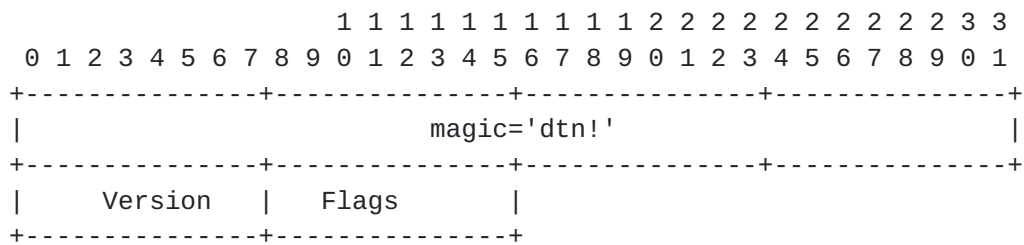


Figure 14: Contact Header Format

See [Section 4.3](#) for details on the use of each of these contact header fields. The fields of the contact header are:

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 protocol).

Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 1.

Name	Code	Description
CAN_TLS	0x01	If bit is set, indicates that the sending peer is capable of TLS security.
Reserved	others	

Table 1: Contact Header Flags

### 4.3. Contact Validation and Negotiation

Upon reception of the contact header, each node 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, an entity



MAY elect to hold an invalid connection open and idle for some time before closing it.

The first negotiation is on the TCPCL protocol version to use. The active node always sends its Contact Header first and waits for a response from the passive node. The active node can repeatedly attempt different protocol versions in descending order until the passive node accepts one with a corresponding Contact Header reply. Only upon response of a Contact Header from the passive node is the TCPCL protocol version established and parameter negotiation begun.

During contact initiation, the active TCPCL node SHALL send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If the active node receives a Contact Header with a different protocol version than the one sent earlier on the TCP connection, the TCP connection SHALL be terminated. If the active node receives a SESS\_TERM message with reason of "Version Mismatch", that node 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 node receives a contact header containing a version that is greater than the current version of the protocol that the node implements, then the node SHALL shutdown the session with a reason code of "Version mismatch". If the passive node receives a contact header with a version that is lower than the version of the protocol that the node implements, the node MAY either terminate the session (with a reason code of "Version mismatch") or the node MAY adapt its operation to conform to the older version of the protocol. The decision of version fall-back is an implementation matter.

#### **4.4. Session Security**

This version of the TCPCL supports establishing a Transport Layer Security (TLS) session within an existing TCP connection. When TLS is used within the TCPCL it affects the entire session. Once established, there is no mechanism available to downgrade a TCPCL session to non-TLS operation. If this is desired, the entire TCPCL session MUST be terminated and a new non-TLS-negotiated session established.

The use of TLS is negotiated using 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 [\[RFC5246\]](#). The parameters within each TLS negotiation are implementation dependent but any TCPCL node SHALL follow all recommended practices of [\[BCP195\]](#), or any updates or successors that





become part of [[BCP195](#)]. By convention, this protocol uses the node which initiated the underlying TCP connection 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 about sensitive data exposure are discussed in [Section 8](#).

#### **[4.4.1](#). TLS Handshake Result**

If a TLS handshake cannot negotiate a TLS session, both entities of the TCPCL session SHALL terminate the TCP connection. At this point the TCPCL session has not yet been established so there is no TCPCL session to terminate. This also avoids any potential security issues associated with further TCP communication with an untrusted peer.

After a TLS session is successfully established, the active peer SHALL send a SESS\_INIT message to begin session negotiation. This session negotiation and all subsequent messaging are secured.

#### **[4.4.2](#). Example TLS Initiation**

A summary of a typical CAN\_TLS usage is shown in the sequence in Figure 15 below.



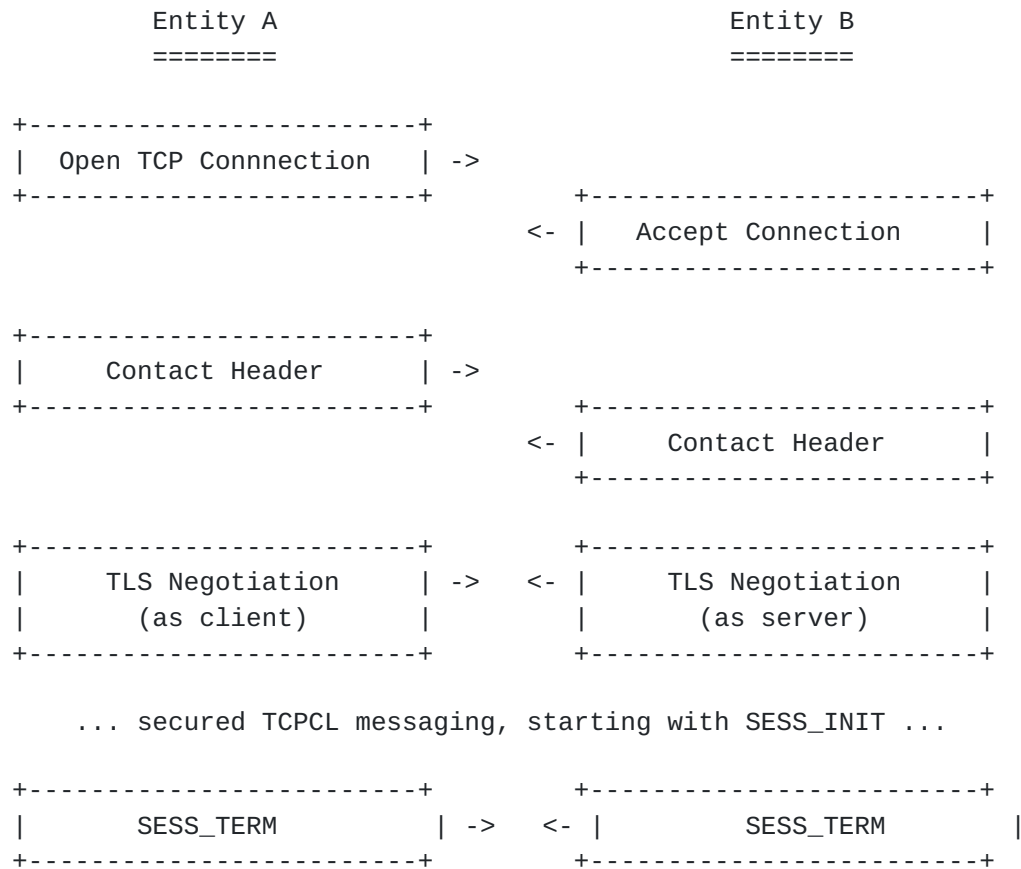


Figure 15: A simple visual example of TCPCL TLS Establishment between two entities

#### 4.5. Message Type Codes

After the initial exchange of a contact header, all messages transmitted over the session are identified by a one-octet header with the following structure:



Figure 16: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 2 below. Encoded values are listed in [Section 9.5](#).



Type	Description
SESS_INIT	Contains the session parameter inputs from one of the entities, as described in <a href="#">Section 4.6</a> .
XFER_INIT	Contains the length (in octets) of the next transfer, as described in <a href="#">Section 5.2.2</a> .
XFER_SEGMENT	Indicates the transmission of a segment of bundle data, as described in <a href="#">Section 5.2.3</a> .
XFER_ACK	Acknowledges reception of a data segment, as described in <a href="#">Section 5.2.4</a> .
XFER_REFUSE	Indicates that the transmission of the current bundle SHALL be stopped, as described in <a href="#">Section 5.2.5</a> .
KEEPALIVE	Used to keep TCPCL session active, as described in <a href="#">Section 5.1.1</a> .
SESS_TERM	Indicates that one of the entities participating in the session wishes to cleanly terminate the session, as described in <a href="#">Section 6</a> .
MSG_REJECT	Contains a TCPCL message rejection, as described in <a href="#">Section 5.1.2</a> .

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.

The format of a SESS\_INIT message is as follows in Figure 17.



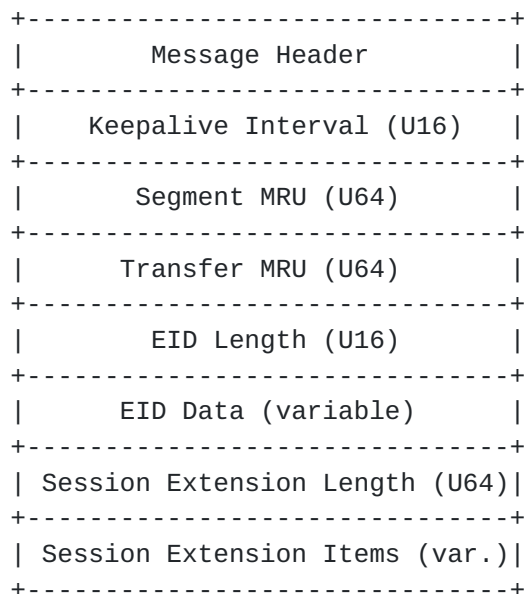


Figure 17: SESS\_INIT Format

A 16-bit unsigned integer indicating the interval, in seconds, between any subsequent messages being transmitted by the peer. The peer receiving this contact header uses this interval to determine how long to wait after any last-message transmission and a necessary subsequent KEEPALIVE message transmission.

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 relation between the two is required.

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 relation between the two is required.

Together these fields represent a variable-length text string. The EID Length is a 16-bit unsigned integer indicating the number of octets of EID Data to follow. A zero EID Length SHALL be used to indicate the lack of EID rather than a truly empty EID. This case allows an entity to avoid exposing EID information on an untrusted network. A non-zero-length EID Data SHALL contain the UTF-8 encoded EID of some singleton endpoint in which the sending





entity is a member, in the canonical format of <scheme name>:<scheme-specific part>. This EID encoding is consistent with [[I-D.ietf-dtn-bpbis](#)].

Together these fields represent protocol extension data not defined by this specification. The Session Extension Length is the total number of octets to follow which are used to encode the Session Extension Item list. The encoding of each Session Extension Item is within a consistent data container as described in [Section 4.6.1](#). 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 MAY be significant, as defined in the associated type specification(s).

#### **[4.6.1](#). 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 18. The fields of the Session Extension Item are:

**Flags:** A one-octet field containing generic bit flags about the Item, which are listed in Table 3. If a TCPCL entity receives a Session Extension Item with an unknown Item Type and the CRITICAL flag set, the entity SHALL close the TCPCL session with SESS\_TERM reason code of "Contact Failure". If the CRITICAL flag is not set, 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 allocate an IANA registry for such codes (see [Section 9.3](#)).

**Item Length:** A 32-bit unsigned integer field containing the number of Item Value octets to follow.

**Item Value:** A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specification SHOULD avoid the use of large data exchanges within the TCPCL contact header as no bundle transfers can begin until the full contact exchange and negotiation has been completed.



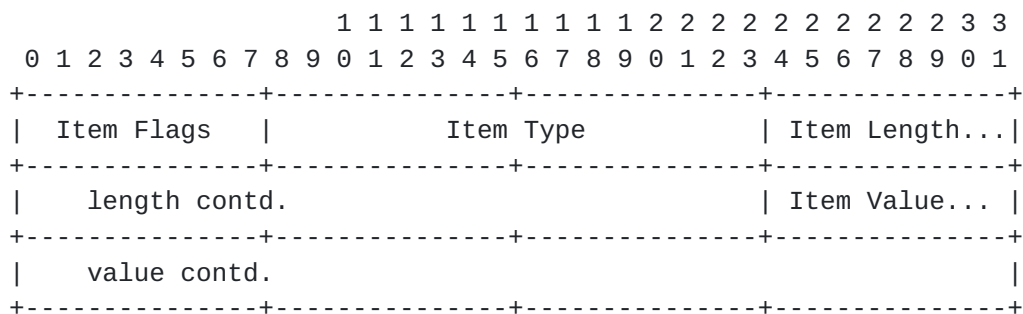


Figure 18: Session Extension Item Format

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

Table 3: Session Extension Item Flags

#### 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 contact header it sent to the peer) with the preferences of the peer node (expressed in the contact header 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 transmit unit (MTU) for whole transfers and individual segments are identical to the Transfer MRU and Segment MRU, respectively, of the received contact header. 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.

**Session Keepalive:** Negotiation of the Session Keepalive parameter is performed by taking the minimum of this two contact headers' Keepalive Interval. The Session Keepalive interval is a parameter for the behavior described in [Section 5.1.1](#).

**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 of the



negotiated value of Enable TLS is acceptable. It can be a reasonable security policy to both require or disallow the use of TLS depending upon the desired network flows. If the Enable TLS state is unacceptable, the node SHALL terminate the session with a reason code of "Contact Failure". Note that this contact failure is different than a failure of TLS handshake after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is true and acceptable then TLS negotiation feature (described in [Section 4.4](#)) begins immediately following the contact header exchange.

Once this process of parameter negotiation is completed (which includes a possible completed TLS handshake of the connection to use TLS), 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.

## **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 4.3](#), a negotiated parameter of each session is the Session Keepalive interval. If the negotiated Session Keepalive is zero (i.e. one or both contact headers contains a zero Keepalive Interval), then the keepalive feature is disabled. There is no logical minimum value for the keepalive interval, 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, 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).

Note: The Keepalive Interval SHOULD NOT be chosen too short as TCP retransmissions MAY occur in case of packet loss. Those will have to be triggered by a timeout (TCP retransmission timeout (RTT)), which



is dependent on the measured RTT for the TCP connection so that KEEPALIVE messages MAY 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 SHOULD 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 node SHOULD terminate the session by transmitting a SESS\_TERM message (as described in [Section 6.1](#)) with reason code "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 keepout interval.

#### [5.1.2.](#) Message Rejection (MSG\_REJECT)

If a TCPCL node receives a message which is unknown to it (possibly due to an unhandled protocol mismatch) or is inappropriate for the current session state (e.g. a KEEPALIVE message received after contact header negotiation has disabled that feature), there is a protocol-level message to signal this condition in the form of a MSG\_REJECT reply.

The format of a MSG\_REJECT message is as follows in Figure 19.

```

+-----+
|      Message Header      |
+-----+
|      Reason Code (U8)    |
+-----+
|  Rejected Message Header  |
+-----+

```

Figure 19: Format of MSG\_REJECT Messages

The fields of the MSG\_REJECT message are:

**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 node.
Message Unsupported	0x02	A message was received but the TCPCL node 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 in this section are directly associated with transferring a bundle between TCPCL entities.

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one node to the other. In TCPCL a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU (see [Section 4.2](#)). The choice of the length to use for segments is an implementation matter, but each segment MUST be no larger than the receiving node's maximum receive unit (MRU) (see the field "Segment MRU" of [Section 4.2](#)). The first segment for a bundle MUST set the 'START' flag, and the last one MUST set the 'end' flag in the XFER\_SEGMENT message flags.

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 TCPCL itself.

If multiple bundles are transmitted on a single TCPCL connection, they MUST be transmitted consecutively without 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 has no relation to concepts such as bundle fragmentation. Each invocation of TCPCL by the bundle protocol



agent, requesting transmission of a bundle (fragmentary or otherwise), results in the initiation of a single TCPCL transfer. Each transfer entails the sending of a XFER\_INIT message and some number of XFER\_SEGMENT and XFER\_ACK messages; all are correlated by the same Transfer ID.

Transfer IDs from each node SHALL be unique within a single TCPCL session. The initial Transfer ID from each node SHALL have value zero. Subsequent Transfer ID values SHALL be incremented from the prior Transfer ID value by one. Upon exhaustion of the entire 64-bit Transfer ID space, the sending node SHALL terminate the session with SESS\_TERM reason code "Resource Exhaustion".

For bidirectional bundle transfers, a TCPCL node SHOULD NOT rely on any relation between Transfer IDs originating from each side of the TCPCL session.

#### **[5.2.2.](#) Transfer Initialization (XFER\_INIT)**

The XFER\_INIT message contains the total length, in octets, of the bundle data in the associated transfer. The total length is formatted as a 64-bit unsigned integer.

The purpose of the XFER\_INIT message is to allow entities to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving node for the upcoming bundle data. See [Section 5.2.5](#) for details on when refusal based on XFER\_INIT content is acceptable.

The Total Bundle Length field within a XFER\_INIT message 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 XFER\_INIT message, the receiver SHOULD treat the transmitted data as invalid.

The format of the XFER\_INIT message is as follows in Figure 20.



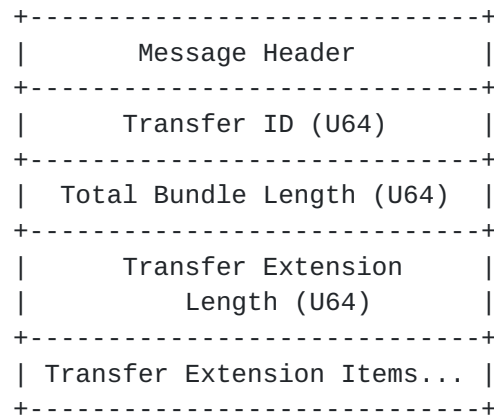


Figure 20: Format of XFER\_INIT Messages

The fields of the XFER\_INIT message are:

**Transfer ID:** A 64-bit unsigned integer identifying the transfer about to begin.

**Total Bundle Length:** A 64-bit unsigned integer indicating the size of the data-to-be-transferred.

**Transfer Extension Length and Transfer Extension Items:** Together these fields represent protocol extension data not defined by this specification. The Transfer Extension Length is the total number of octets to follow which are used to encode the Transfer Extension Item list. The encoding of each Transfer Extension Item is within a consistent data container as described in [Section 5.2.2.1](#). The full set of transfer extension items apply only to the associated single transfer. The order and multiplicity of these transfer extension items MAY be significant, as defined in the associated type specification(s).

An XFER\_INIT message SHALL be sent as the first message in a transfer sequence, before transmission of any XFER\_SEGMENT messages for the same Transfer ID. XFER\_INIT messages MUST NOT be sent unless the next XFER\_SEGMENT message has the 'START' bit set to "1" (i.e., just before the start of a new transfer).

#### [5.2.2.1](#). 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 21. The fields of the Transfer Extension Item are:

**Flags:** A one-octet field containing generic bit flags about the Item, which are listed in Table 5. If a TCPCL node receives a



Transfer Extension Item with an unknown Item Type and the CRITICAL flag set, the node SHALL refuse the transfer with an XFER\_REFUSE reason code of "Extension Failure". If the CRITICAL flag is not set, 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 allocate an IANA registry for such codes (see [Section 9.4](#)).

Item Length: A 32-bit unsigned integer field containing the number of Item Value octets to follow.

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specification SHOULD avoid the use of large data exchanges within the XFER\_INIT as the associated transfer cannot begin until the full initialization message is sent.

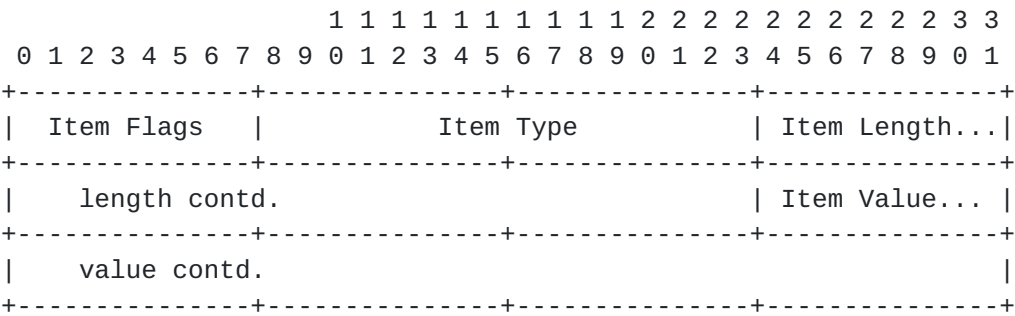


Figure 21: Transfer Extension Item Format

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

Table 5: Transfer Extension Item Flags





### 5.2.3. Data Transmission (XFER\_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER\_SEGMENT message follows in Figure 22.

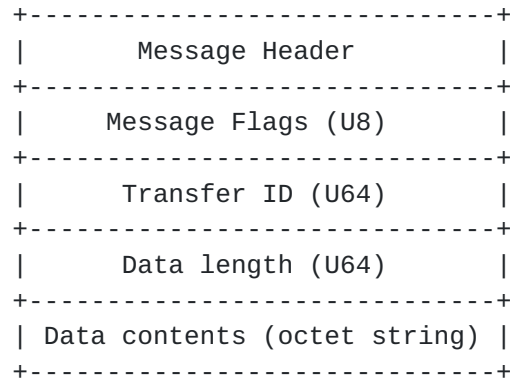


Figure 22: Format of XFER\_SEGMENT Messages

The fields of the XFER\_SEGMENT message are:

**Message Flags:** A one-octet field of single-bit flags, interpreted according to the descriptions in Table 6.

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

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

**Data contents:** The variable-length data payload of the message.

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

Table 6: XFER\_SEGMENT Flags

The flags portion of the message contains two optional values in the two low-order bits, denoted 'START' and 'END' in Table 6. The



'START' bit MUST be set to one if it precedes the transmission of the first segment of a transfer. The 'END' bit MUST be set to one when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the 'START' and 'END' bits MUST be set to one.

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

#### 5.2.4. 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 the segment. To this end, the TCPCL protocol provides feedback messaging whereby a receiving node transmits acknowledgments of reception of data segments.

The format of an XFER\_ACK message follows in Figure 23.

```

+-----+
|      Message Header      |
+-----+
|      Message Flags (U8)   |
+-----+
|      Transfer ID (U64)    |
+-----+
| Acknowledged length (U64) |
+-----+

```

Figure 23: Format of XFER\_ACK Messages

The fields of the XFER\_ACK message are:

**Message Flags:** A one-octet field of single-bit flags, interpreted according to the descriptions in Table 6.

**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 which are being acknowledged.

A receiving TCPCL node SHALL send an XFER\_ACK message in response to each received XFER\_SEGMENT message. The flags portion of the XFER\_ACK header SHALL be set to match the corresponding DATA\_SEGMENT message being acknowledged. 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 node MAY transmit multiple XFER\_SEGMENT messages without necessarily waiting for the corresponding XFER\_ACK responses. This enables pipelining of messages on a transfer stream.

For example, suppose the sending node transmits four segments of bundle data with lengths 100, 200, 500, and 1000, respectively. After receiving the first segment, the node sends an acknowledgment of length 100. After the second segment is received, the node sends an acknowledgment of length 300. The third and fourth acknowledgments are of length 800 and 1800, respectively.

#### **5.2.5. Transfer Refusal (XFER\_REFUSE)**

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

There is no required relation between the Transfer MRU of a TCPCL node (which is supposed to represent a firm limitation of what the node will accept) and 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 receiver MAY send an XFER\_REFUSE message as soon as it receives a XFER\_INIT message without waiting for the next XFER\_SEGMENT message. The sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

The format of the XFER\_REFUSE message is as follows in Figure 24.



Message Header
Reason Code (U8)
Transfer ID (U64)

Figure 24: Format of XFER\_REFUSE Messages

The fields of the XFER\_REFUSE message are:

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

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

Name	Semantics
Unknown	Reason for refusal is unknown or not specified.
Extension Failure	A failure processing the Transfer Extension Items has occurred.
Completed	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
No Resources	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.

Table 7: XFER\_REFUSE Reason Codes

The receiver MUST, for each transfer preceding the one to be refused, have either acknowledged all XFER\_SEGMENTs 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** commence transmission of any further segments of the refused bundle subsequently. 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 **MAY** cross on the wire; if this happens, subsequent segments of the bundle **SHOULD** also be refused with a `XFER_REFUSE` message.

Note: If a bundle transmission is aborted in this way, the receiver **MAY** 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' bit set to '1', indicating the start of a new transfer, and with a distinct Transfer ID value.

## **6. Session Termination**

This section describes the procedures for ending a TCPCL session.

### **6.1. Session Termination Message (`SESS_TERM`)**

To cleanly shut down a session, a `SESS_TERM` message **SHALL** be transmitted by either node at any point following complete transmission of any other message. When sent to initiate a termination, the `REPLY` bit of a `SESS_TERM` message **SHALL NOT** be set. Upon receiving a `SESS_TERM` message after not sending a `SESS_TERM` message in the same session, an entity **SHOULD** 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` bit, which is set to indicate acknowledgement.

After sending a `SESS_TERM` message, an entity **MAY** continue a possible in-progress transfer in either direction. After sending a `SESS_TERM` message, an entity **SHALL NOT** begin any new outgoing transfer (i.e. send an `XFER_INIT` message) for the remainder of the session. After receiving a `SESS_TERM` message, an entity **SHALL NOT** accept any new incoming transfer for the remainder of the session.

Instead of following a clean shutdown sequence, after transmitting a `SESS_TERM` message an entity **MAY** immediately close the associated TCP connection. When performing an unclean shutdown, a receiving node **SHOULD** acknowledge all received data segments before closing the TCP connection. When performing an unclean shutdown, a transmitting node **SHALL** treat either sending or receiving a `SESS_TERM` message (i.e. before the final acknowledgment) as a failure of the transfer. Any delay between request to terminate the TCP connection and actual closing of the connection (a "half-closed" state) **MAY** be ignored by the TCPCL node.



The format of the SESS\_TERM message is as follows in Figure 25.

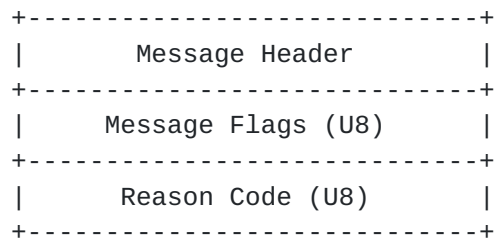


Figure 25: Format of SESS\_TERM Messages

The fields of the SESS\_TERM message are:

**Message Flags:** A one-octet field of single-bit flags, interpreted according to the descriptions in Table 8.

**Reason Code:** A one-octet refusal reason code interpreted according to the descriptions in Table 9.

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

Table 8: SESS\_TERM Flags



Name	Description
Unknown	A termination reason is not available.
Idle timeout	The session is being closed due to idleness.
Version mismatch	The node cannot conform to the specified TCPCL protocol version.
Busy	The node is too busy to handle the current session.
Contact Failure	The node cannot interpret or negotiate contact header option.
Resource Exhaustion	The node has run into some resource limit and cannot continue the session.

Table 9: SESS\_TERM Reason Codes

A session shutdown MAY occur immediately after transmission of a contact header (and prior to any further message transmit). This MAY, for example, be used to notify that the node 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.

If reception of the contact header itself somehow fails (e.g. an invalid "magic string" is received), an entity SHOULD close the TCP connection without sending a SESS\_TERM message. If the content of the Session Extension Items data disagrees with the Session Extension Length (i.e. the last Item claims to use more octets than are present in the Session Extension Length), the reception of the contact header is considered to have failed.

If a session is to be terminated before a protocol message has completed being sent, then the node MUST NOT transmit the SESS\_TERM message but still SHOULD 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 an other message. This is particularly important when large segment sizes are being transmitted; either entire XFER\_SEGMENT is sent before a SESS\_TERM message or the connection is simply terminated mid-XFER\_SEGMENT.



## **6.2. Idle Session Shutdown**

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

If there is a configured time to close idle links and if no TCPCL messages (other than KEEPALIVE messages) has been received for at least that amount of time, then either node MAY terminate the session by transmitting a SESS\_TERM message indicating the reason code of "Idle timeout" (as described in Table 9).

## **7. Implementation Status**

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to [\[RFC7942\]](#) and [\[github-dtn-bpbis-tcpcl\]](#).]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [\[RFC7942\]](#). The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

An example implementation of the this draft of TCPCLv4 has been created as a GitHub project [\[github-dtn-bpbis-tcpcl\]](#) and is intended to use as a proof-of-concept and as a possible source of interoperability testing. This example implementation uses D-Bus as the CL-BP Agent interface, so it only runs on hosts which provide the Python "dbus" library.

## **8. Security Considerations**

One security consideration for this protocol relates to the fact that entities present their endpoint identifier as part of the contact header exchange. It would be possible for an entity to fake this value and present the identity of a singleton endpoint in which the node is not a member, essentially masquerading as another DTN node. If this identifier is used outside of a TLS-secured session or without further verification as a means to determine which bundles





are transmitted over the session, then the node that has falsified its identity would be able to obtain bundles that it otherwise would not have. Therefore, an entity SHALL NOT use the EID value of an unsecured contact header to derive a peer node's identity unless it can corroborate it via other means. When TCPCL session security is mandated by a TCPCL peer, that peer SHALL transmit initial unsecured contact header values indicated in Table 10 in order. These values avoid unnecessarily leaking session parameters and will be ignored when secure contact header re-exchange occurs.

Parameter	Value
Flags	The USE_TLS flag is set.
Keepalive Interval	Zero, indicating no keepalive.
Segment MRU	Zero, indicating all segments are refused.
Transfer MRU	Zero, indicating all transfers are refused.
EID	Empty, indicating lack of EID.

Table 10: Recommended Unsecured Contact Header

TCPCL can be used to provide point-to-point transport security, but does not provide security of data-at-rest and does not guarantee end-to-end bundle security. The mechanisms defined in [[RFC6257](#)] and [[I-D.ietf-dtn-bpsec](#)] are to be used instead.

Even when using TLS to secure the TCPCL session, the actual ciphersuite negotiated between the TLS peers MAY be insecure. TLS can be used to perform authentication without data confidentiality, for example. It is up to security policies within each TCPCL node to ensure that the negotiated TLS ciphersuite meets transport security requirements. This is identical behavior to STARTTLS use in [[RFC2595](#)].

Another consideration for this protocol relates to denial-of-service attacks. An entity MAY send a large amount of data over a TCPCL session, requiring the receiving entity to handle the data, attempt to stop the flood of data by sending a XFER\_REFUSE message, or forcibly terminate the session. This burden could cause denial of service on other, well-behaving sessions. There is also nothing to prevent a malicious entity from continually establishing sessions and repeatedly trying to send copious amounts of bundle data. A listening entity MAY take countermeasures such as ignoring TCP SYN



messages, closing TCP connections as soon as they are established, waiting before sending the contact header, sending a SESS\_TERM message quickly or with a delay, etc.

## 9. IANA Considerations

In this section, registration procedures are as defined in [\[RFC8126\]](#).

Some of the registries below are created new for TCPCLv4 but share code values with TCPCLv3. This was done to disambiguate the use of these values between TCPCLv3 and TCPCLv4 while preserving the semantics of some values.

### 9.1. Port Number

Port number 4556 has been previously assigned as the default port for the TCP convergence layer in [\[RFC7242\]](#). This assignment is unchanged by protocol version 4. Each TCPCL entity identifies its TCPCL protocol version in its initial contact (see [Section 9.2](#)), so there is no ambiguity about what protocol is being used.

Parameter	Value
Service Name:	dtn-bundle
Transport Protocol(s):	TCP
Assignee:	Simon Perreault <simon@per.reau.lt>
Contact:	Simon Perreault <simon@per.reau.lt>
Description:	DTN Bundle TCP CL Protocol
Reference:	<a href="#">[RFC7242]</a>
Port Number:	4556

### 9.2. Protocol Versions

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version Numbers" and initialize it with the following table. The registration procedure is RFC Required.



Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4	TCPCLbis	This specification.
5-255	Unassigned	

### 9.3. Session Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Session Extension Types" and initialize it with the contents of Table 11. The registration procedure is RFC Required within the lower range 0x0001--0x7fff. Values in the range 0x8000--0xffff are reserved for use on private networks for functions not published to the IANA.

Code	Message Type
0x0000	Reserved
0x0001--0x7fff	Unassigned
0x8000--0xffff	Private/Experimental Use

Table 11: Session Extension Type Codes

### 9.4. Transfer Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4



Transfer Extension Types" and initialize it with the contents of Table 12. The registration procedure is RFC Required within the lower range 0x0001--0x7fff. Values in the range 0x8000--0xffff are reserved for use on private networks for functions not published to the IANA.

Code	Message Type
0x0000	Reserved
0x0001--0x7fff	Unassigned
0x8000--0xffff	Private/Experimental Use

Table 12: Transfer Extension Type Codes

### 9.5. Message Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" and initialize it with the contents of Table 13. The registration procedure is RFC Required.





Code	Message Type
0x00	Reserved
0x01	XFER_SEGMENT
0x02	XFER_ACK
0x03	XFER_REFUSE
0x04	KEEPALIVE
0x05	SESS_TERM
0x06	XFER_INIT
0x07	MSG_REJECT
0x08--0xf	Unassigned

Table 13: Message Type Codes

#### **9.6. XFER\_REFUSE Reason Codes**

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER\_REFUSE Reason Codes" and initialize it with the contents of Table 14. The registration procedure is RFC Required.



Code	Refusal Reason
0x0	Unknown
0x1	Extension Failure
0x2	Completed
0x3	No Resources
0x4	Retransmit
0x5--0x7	Unassigned
0x8--0xf	Reserved for future usage

Table 14: XFER\_REFUSE Reason Codes

### 9.7. SESS\_TERM Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 SESS\_TERM Reason Codes" and initialize it with the contents of Table 15. The registration procedure is RFC Required.



Code	Shutdown Reason
0x00	Unknown
0x01	Idle timeout
0x02	Version mismatch
0x03	Busy
0x04	Contact Failure
0x05	Resource Exhaustion
0x06--0xFF	Unassigned

Table 15: SESS\_TERM Reason Codes

#### 9.8. MSG\_REJECT Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 MSG\_REJECT Reason Codes" and initialize it with the contents of Table 16. The registration procedure is RFC Required.

Code	Rejection Reason
0x00	reserved
0x01	Message Type Unknown
0x02	Message Unsupported
0x03	Message Unexpected
0x04-0xFF	Unassigned

Table 16: REJECT Reason Codes



## **10. Acknowledgments**

This specification is based on comments on implementation of [RFC7242] provided from Scott Burleigh.

## **11. References**

### **11.1. Normative References**

- [BCP195] 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.
- [I-D.ietf-dtn-bpbis] Fall, K. and E. Birrane, "Bundle Protocol Version 7", [draft-ietf-dtn-bpbis-11](#) (work in progress), May 2018.
- [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>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [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>>.

### **11.2. Informative References**





[github-dtn-bpbis-tcpcl]

Sipos, B., "TCPCL Example Implementation",  
<<https://github.com/BSipos-RKF/dtn-bpbis-tcpcl/tree/develop>>.

[I-D.ietf-dtn-bpsec]

Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", [draft-ietf-dtn-bpsec-08](#) (work in progress), October 2018.

[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>>.

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

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

[RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", [RFC 6257](#), DOI 10.17487/RFC6257, May 2011, <<https://www.rfc-editor.org/info/rfc6257>>.

[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>>.

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

## **Appendix A.** Significant changes from [RFC7242](#)

The areas in which changes from [RFC7242](#) have been made to existing headers and messages are:

- o Split contact header into pre-TLS protocol negotiation and SESS\_INIT parameter negotiation. The contact header is now fixed-length.



- o Changed contact header content to limit number of negotiated options.
- o Added contact option to negotiate maximum segment size (per each direction).
- o Added session extension capability.
- o Added transfer extension capability.
- o Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.
- o Expanded Message Header to octet-aligned fields instead of bit-packing.
- o Added a bundle transfer identification number to all bundle-related messages (XFER\_INIT, XFER\_SEGMENT, XFER\_ACK, XFER\_REFUSE).
- o Use flags in XFER\_ACK to mirror flags from XFER\_SEGMENT.
- o Removed all uses of SDNV fields and replaced with fixed-bit-length fields.
- o Renamed SHUTDOWN to SESS\_TERM to deconflict term "shutdown".
- o Removed the notion of a re-connection delay parameter.

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

- o Added contact negotiation failure SESS\_TERM reason code.
- o Added MSG\_REJECT message to indicate an unknown or unhandled message was received.
- o Added TLS session security mechanism.
- o Added Resource Exhaustion SESS\_TERM reason code.

Authors' Addresses



Brian Sipos  
RKF Engineering Solutions, LLC  
7500 Old Georgetown Road  
Suite 1275  
Bethesda, MD 20814-6198  
United States of America

Email: BSipos@rkf-eng.com

Michael Demmer  
University of California, Berkeley  
Computer Science Division  
445 Soda Hall  
Berkeley, CA 94720-1776  
United States of America

Email: demmer@cs.berkeley.edu

Joerg Ott  
Aalto University  
Department of Communications and Networking  
PO Box 13000  
Aalto 02015  
Finland

Email: jo@netlab.tkk.fi

Simon Perreault  
Quebec, QC  
Canada

Email: simon@per.reau.lt

