Delay Tolerant Networking
Internet-Draft
Obsoletes: 7242 (if approved)
Intended status: Standards Track
Expires: August 1, 2018

B. Sipos
RKF Engineering
M. Demmer
UC Berkeley
J. Ott
Aalto University
S. Perreault

January 28, 2018

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-06

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 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

Copyright Notice

Table of Contents

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.

```
         +-------------------------+
         |     DTN Application     | -\
         +-------------------------|   |
         |  Bundle Protocol (BP)   |   -> Application Layer
         +-------------------------+   |
         | TCP Conv. Layer (TCPCL) |   |
         +-------------------------+   |
         |     TLS (optional)      | -/
         +-------------------------+
         |          TCP            | ---> Transport Layer
         +-------------------------+
         |       IPv4/IPv6         | ---> Network Layer
         +-------------------------+
         |   Link-Layer Protocol   | ---> Link Layer
         +-------------------------+
```

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:

- 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.

- Mechanisms for locating or identifying other bundle nodes within an internet.

1.1. Convergence Layer Services

This version of the TCPCL provides the following services to support the overlaying Bundle Protocol agent:

Attempt Session The TCPCL allows a BP agent to pre-emptively attempt to establish a TCPCL session with a peer node. Each session attempt can send a different set of contact header parameters as directed by the BP agent.

Session Started The TCPCL supports indication when a new TCP connection has been started (as either client or server) before the TCPCL handshake has begun.

Session Established The TCPCL supports indication when a new session has been fully established and is ready for its first transfer.

Session Shutdown The TCPCL supports indication when an established session has been ended by normal exchange of SHUTDOWN messages with all transfers completed.

Session Failed The TCPCL supports indication when a session fails, either during contact negotiation, TLS negotiation, or after establishement for any reason other than normal shutdown.

Transmission Availability Because TCPCL transmits serially over a TCP connection, it suffers from "head of queue blocking" and supports indication of when an established session is live-but-idle (i.e. available for immediate transfer start) or live-and-not-idle.

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

Transmission Intermediate Progress The TCPCL supports positive indication of intermediate progress of transferr to a peer node. 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 node rejects the bundle or when a TCPCL session ends before transferr success. The TCPCL itself does not have a notion of transfer timeout.

Reception Interruption The TCPCL allows a BP agent to interrupt an individual transfer before it has fully completed (successfully or not).

Reception Success The TCPCL supports positive indication when a bundle has been fully transferred from a peer node.

Reception Intermediate Progress The TCPCL supports positive indication of intermediate progress of transfer from the peer node. 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 node 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 Node: This term refers to either side of a negotiating or in-service TCPCL Session. For most TCPCL behavior, the two nodes are symmetric and there is no protocol distinction between them. Some specific behavior, particularly during negotiation, distinguishes between the connecting node and the connected-to node. For the remainder of this document, the term "node" without the prefix "TCPCL" refers to a TCPCL node.

TCP Connection: This term refers to a transport connection using TCP as the transport protocol.

TCPCL Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two bundle nodes. 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 nodes 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 node and thereby to the TCPCL is implementation dependent. However, the mechanism by which two bundle nodes exchange and negotiate the values to be used for a given session is described in Section 4.3.

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.

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.

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 set parameters of the TCPCL session and exchange 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 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 SHUTDOWN 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 SHUTDOWN 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.

TCPCL is a symmetric protocol between the peers of a session. 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. 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 Node A to Node 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 channel. 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.

```
                 Node A                              Node B
                 ======                              ======
       +-------------------------+         +-------------------------+
       |     Contact Header      | ->   <- |     Contact Header      |
       +-------------------------+         +-------------------------+

       +-------------------------+
       |        XFER_INIT        | ->
       |     Transfer ID [I1]    |
       |    Total Length [L1]    |
       +-------------------------+
       +-------------------------+
       |   XFER_SEGMENT (start)  | ->
       |     Transfer ID [I1]    |
       |       Length [L1]       |
       |  Bundle Data 0..(L1-1)  |
       +-------------------------+
       +-------------------------+         +-------------------------+
       |     XFER_SEGMENT        | ->   <- |     XFER_ACK (start)    |
       |     Transfer ID [I1]    |         |     Transfer ID [I1]    |
       |       Length   [L2]     |         |        Length   [L1]    |
       |Bundle Data L1..(L1+L2-1)|         +-------------------------+
       +-------------------------+
       +-------------------------+         +-------------------------+
       |    XFER_SEGMENT (end)   | ->   <- |         XFER_ACK        |
       |     Transfer ID [I1]    |         |     Transfer ID [I1]    |
       |        Length   [L3]    |         |      Length   [L1+L2]   |
       |Bundle Data              |         +-------------------------+
       |    (L1+L2)..(L1+L2+L3-1)|
       +-------------------------+

                                           +-------------------------+
                                        <- |      XFER_ACK (end)     |
                                           |     Transfer ID [I1]    |
                                           |     Length   [L1+L2+L3] |
                                           +-------------------------+

       +-------------------------+         +-------------------------+
       |       SHUTDOWN          | ->   <- |         SHUTDOWN        |
       +-------------------------+         +-------------------------+
```

Figure 2: An Example of the Flow of Protocol Messages on a Single TCP Session between Two Nodes (A and B)

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating nodes. 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, a node MUST first establish a TCP connection with the intended peer node, 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 node is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. A node 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 node 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. In case a SHUTDOWN message specifying a reconnection delay is received, that delay is used as the initial delay. The default initial re-attempt delay SHOULD be no shorter than 1 second and SHOULD be configurable since it will be application and network type dependent.

Once a TCP connection is established, each node 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 nodes perform the validation and negotiation procedures defined in Section 4.3. After receiving the contact header from the other node, either node MAY refuse the session by sending a SHUTDOWN message with an appropriate reason code.

The format for the Contact Header is as follows:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|                          magic=’dtn!’                         |
+---------------+---------------+---------------+---------------+
|     Version   |   Flags       |      Keepalive Interval       |
+---------------+---------------+---------------+---------------+
|                          Segment MRU...                       |
+---------------+---------------+---------------+---------------+
|                          contd.                               |
+---------------+---------------+---------------+---------------+
|                         Transfer MRU...                       |
+---------------+---------------+---------------+---------------+
|                          contd.                               |
+---------------+---------------+---------------+---------------+
|          EID Length           |             EID Data...       |
+---------------+---------------+---------------+---------------+
|                        EID Data contd.                        |
+---------------+---------------+---------------+---------------+
|                  Header Extension Length...                   |
+---------------+---------------+---------------+---------------+
|                          contd.                               |
+---------------+---------------+---------------+---------------+
|                   Header Extension Items...                   |
+---------------+---------------+---------------+---------------+
```

Figure 3: 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.

Keepalive Interval: 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.

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 nodes of a single session MAY have different Segment MRUs, and no relation 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 nodes of a single session MAY have different Transfer MRUs, and no relation between the two is required.

EID Length and EID Data: 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 a node 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 node is a member, in the canonical format of <scheme name>:<scheme-specific part>. This EID encoding is consistent with [I-D.ietf-dtn-bpbis].

Header Extension Length and Header Extension Items: Together these fields represent protocol extension data not defined by this specification. The Header Extension Length is the total number of octets to follow which are used to encode the Header Extension Item list. The encoding of each Header Extension Item is within a consistent data container as described in Section 4.2.1.

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

4.2.1. Header Extension Items

Each of the Header Extension Items SHALL be encoded in an identical Type-Length-Value (TLV) container form as indicated in Figure 4. The fields of the Header Extension Item are:

Flags: A one-octet field containing generic bit flags about the Item, which are listed in Table 2. If a TCPCL node receives a Header Extension Item with an unknown Item Type and the CRITICAL flag set, the node SHALL close the TCPCL session with SHUTDOWN reason code of "Contact Failure". If the CRITICAL flag is not set, a node 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 8.3).

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.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|  Item Flags   |           Item Type           | Item Length...|
+---------------+---------------+---------------+---------------+
|    length contd.                              | Item Value... |
+---------------+---------------+---------------+---------------+
|    value contd.                                               |
+---------------+---------------+---------------+---------------+
```

Figure 4: Header 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 2: Header Extension Item Flags

4.3. Validation and Parameter 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, a node MAY elect to hold an invalid connection open and idle for some time before closing it.

A connecting TCPCL node SHALL send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If a connecting node receives a SHUTDOWN 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 a 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 a 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.

A node 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 idententical to the Transfer MRU and Segment MRU, respectively, of the recevied 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 5.2.1.

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 negotated value of Enable TLS is acceptable. If not, the node SHALL shutdown the session with a reason code of "Contact Failure". Note that this contact failure is different than a "TLS Failure" 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.

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 shutdown and a new non-TLS-negotiated session established.

The use of TLS is negotated 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 nodes SHALL begin a TLS handshake in accordance with [RFC5246]. The parameters within each TLS negotiation are implementation dependent but any TCPCL node SHOULD follow all recommended best practices of [RFC7525]. 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 7.

4.4.1. TLS Handshake Result

If a TLS handshake cannot negotiate a TLS session, both nodes of the TCPCL session SHALL start a TCPCL shutdown with reason "TLS Failure".

After a TLS session is successfully established, both TCPCL nodes SHALL re-exchange TCPCL Contact Header messages. Any information cached from the prior Contact Header exchange SHALL be discarded. This re-exchange avoids a "man-in-the-middle" attack in identical fashion to [RFC2595]. Each re-exchange header CAN_TLS flag SHALL be identical to the original header CAN_TLS flag from the same node. The CAN_TLS logic (TLS negotiation) SHALL NOT apply during header re-exchange. This reinforces the fact that there is no TLS downgrade mechanism.

4.4.2. Example TLS Initiation

A summary of a typical CAN_TLS usage is shown in the sequence in Figure 5 below.

```
                 Node A                              Node B
                 ======                              ======

       +-------------------------+
       |  Open TCP Connnection   | ->
       +-------------------------+
                                           +-------------------------+
                                        <- |   Accept Connection     |
                                           +-------------------------+

       +-------------------------+         +-------------------------+
       |     Contact Header      | ->   <- |     Contact Header      |
       +-------------------------+         +-------------------------+

       +-------------------------+         +-------------------------+
       |     TLS Negotiation     | ->   <- |     TLS Negotiation     |
       |       (as client)       |         |       (as server)       |
       +-------------------------+         +-------------------------+

       +-------------------------+         +-------------------------+
       |     Contact Header      | ->   <- |     Contact Header      |
       +-------------------------+         +-------------------------+

                       ... secured TCPCL messaging ...

       +-------------------------+         +-------------------------+
       |       SHUTDOWN          | ->   <- |         SHUTDOWN        |
       +-------------------------+         +-------------------------+
```

Figure 5: A simple visual example of TCPCL TLS Establishment between two nodes

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. 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:

```
 0 1 2 3 4 5 6 7
+---------------+
| Message Type  |
+---------------+
```

Figure 6: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 3 below. Encoded values are listed in Section 8.4.

Type	Description
XFER_INIT	Contains the length (in octets) of the next transfer, as described in Section 5.3.2.
XFER_SEGMENT	Indicates the transmission of a segment of bundle data, as described in Section 5.3.3.
XFER_ACK	Acknowledges reception of a data segment, as described in Section 5.3.4.
XFER_REFUSE	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.3.5.
KEEPALIVE	Used to keep TCPCL session active, as described in Section 5.2.1.
SHUTDOWN	Indicates that one of the nodes participating in the session wishes to cleanly terminate the session, as described in Section 6.
MSG_REJECT	Contains a TCPCL message rejection, as described in Section 5.2.2.

Table 3: TCPCL Message Types

5.2. Upkeep and Status Messages

5.2.1. 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, nodes 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. Nodes 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 (RTO)), 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 3) 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 MAY terminate the session by transmitting a one-octet SHUTDOWN message (as described in Section 6.1) with reason code "Idle Timeout.

5.2.2. 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 follows:

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

Figure 7: 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

5.3. Bundle Transfer

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

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.

5.3.1. 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 SHUTDOWN 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.

5.3.2. 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 nodes to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving node for the upcoming bundle data. See Section 5.3.5 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:

```
                   +-----------------------------+
                   |       Message Header        |
                   +-----------------------------+
                   |      Transfer ID (U64)      |
                   +-----------------------------+
                   |  Total Bundle Length (U64)  |
                   +-----------------------------+
```

Figure 8: 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.

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).

5.3.3. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message follows in Figure 9.

```
+------------------------------+
|       Message Header         |
+------------------------------+
|     Message Flags (U8)       |
+------------------------------+
|      Transfer ID (U64)       |
+------------------------------+
|      Data length (U64)       |
+------------------------------+
| Data contents (octet string) |
+------------------------------+
```

Figure 9: 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 5.

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 5: 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 5. 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 nodes MAY be achieved using multiple TCPCL sessions.

5.3.4. 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 10.

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

Figure 10: 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 5.

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 channel.

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.

5.3.5. 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:

```
                   +-----------------------------+
                   |       Message Header        |
                   +-----------------------------+
                   |      Reason Code (U8)       |
                   +-----------------------------+
                   |      Transfer ID (U64)      |
                   +-----------------------------+
```

Figure 11: 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 6.

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

Name	Semantics
Unknown	Reason for refusal is unknown or not specified.
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 6: 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 a node 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. Shutdown Message (SHUTDOWN)

To cleanly shut down a session, a SHUTDOWN message MUST be transmitted by either node at any point following complete transmission of any other message. After sending a SHUTDOWN message, the sender of the message MAY send further acknowledgments (XFER_ACK or XFER_REFUSE) but no further data messages (XFER_INIT or XFER_SEGMENT). A receiving node SHOULD acknowledge all received data segments before sending a SHUTDOWN message to end the session. A transmitting node SHALL treat a SHUTDOWN message received mid-transfer (i.e. before the final acknowledgment) as a failure of the transfer.

After transmitting a SHUTDOWN message, a node MAY immediately close the associated TCP connection. Once the SHUTDOWN message is sent, any further received data on the TCP connection SHOULD be ignored. 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 SHUTDOWN message is as follows:

```
+-----------------------------------+
|          Message Header           |
+-----------------------------------+
|         Message Flags (U8)        |
+-----------------------------------+
|     Reason Code (optional U8)     |
+-----------------------------------+
| Reconnection Delay (optional U16) |
+-----------------------------------+
```

Figure 12: Format of SHUTDOWN Messages

The fields of the SHUTDOWN message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 7.

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 8. The Reason Code is present or absent as indicated by one of the flags.

Reconnection Delay: A 16-bit unsigned integer indicating the desired delay, in seconds, before re-attepmting a TCPCL session to the sending node. The Reconnection Delay is present or absent as indicated by one of the flags.

Name	Code	Description
D	0x01	If bit is set, indicates that a Reconnection Delay field is present.
R	0x02	If bit is set, indicates that a Reason Code field is present.
Reserved	others	

Table 7: SHUTDOWN Flags

It is possible for a node to convey optional information regarding the reason for session termination. To do so, the node MUST set the 'R' bit in the message flags and transmit a one-octet reason code immediately following the message header. The specified values of the reason code are:

Name	Description
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.
TLS Failure	The node failed to negotiate TLS session and cannot continue the session.
Resource Exhaustion	The node has run into some resource limit and cannot continue the session.

Table 8: SHUTDOWN Reason Codes

If a node does not want its peer to reopen a connection immediately, it SHALL set the ’D’ bit in the flags and include a reconnection delay to indicate when the peer is allowed to attempt another session setup. The Reconnection Delay value 0 SHALL be interpreted as an infinite delay, i.e., that the connecting node MUST NOT re-establish the session.

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, a node MUST always send the contact header to its peer before sending a SHUTDOWN message.

If reception of the contact header itself somehow fails (e.g. an invalid "magic string" is recevied), a node SHOULD close the TCP connection without sending a SHUTDOWN message. If the content of the Header Extension Items data disagrees with the Header Extension Length (i.e. the last Item claims to use more octets than are present in the Header 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 SHUTDOWN 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 SHUTDOWN 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 SHUTDOWN message indicating the reason code of "Idle timeout" (as described in Table 8).

7. Security Considerations

One security consideration for this protocol relates to the fact that nodes present their endpoint identifier as part of the contact header exchange. It would be possible for a node 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, a node 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 9 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 9: 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. A node MAY send a large amount of data over a TCPCL session, requiring the receiving node 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 node from continually establishing sessions and repeatedly trying to send copious amounts of bundle data. A listening node 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 SHUTDOWN message quickly or with a delay, etc.

8. IANA Considerations

In this section, registration procedures are as defined in [RFC5226].

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.

8.1. 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 node identifies its TCPCL protocol version in its initial contact (see Section 8.2), 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:	[RFC7242]
Port Number:	4556

8.2. 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	

8.3. Header 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 Header Extension Types" and initialize it with the contents of Table 10. The registration procedure is RFC Required within the lower range 0x0001--0x3fff. 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--0x3fff	Unassigned
0x8000--0xffff	Private/Experimental Use

Table 10: Header Extension Type Codes

8.4. 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 11. The registration procedure is RFC Required.

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

Table 11: Message Type Codes

8.5. 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 12. The registration procedure is RFC Required.

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

Table 12: XFER_REFUSE Reason Codes

8.6. SHUTDOWN 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 SHUTDOWN Reason Codes" and initialize it with the contents of Table 13. The registration procedure is RFC Required.

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

Table 13: SHUTDOWN Reason Codes

8.7. 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 14. 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 14: REJECT Reason Codes

9. Acknowledgments

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

10. References

10.1. Normative References

[I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-10 (work in progress), November 2017.

[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>.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[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>.

[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>.

10.2. Informative References

[I-D.ietf-dtn-bpsec] Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-06 (work in progress), October 2017.

[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>.

[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>.

Appendix A. Significant changes from RFC7242

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

- Changed contact header content to limit number of negotiated options.

- Added contact option to negotiate maximum segment size (per each direction).

- Added contact header extension capability.

- Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.

- Expanded Message Header to octet-aligned fields instead of bit-packing.

- Added a bundle transfer identification number to all bundle-related messages (XFER_INIT, XFER_SEGMENT, XFER_ACK, XFER_REFUSE).

- Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.

- Removed all uses of SDNV fields and replaced with fixed-bit-length fields.

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

- Added contact negotiation failure SHUTDOWN reason code.

- Added MSG_REJECT message to indicate an unknown or unhandled message was received.

- Added TLS session security mechanism.

- Added TLS failure and Resource Exhaustion SHUTDOWN reason code.

Authors' Addresses

Brian Sipos
RKF Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
US

Email: BSipos@rkf-eng.com

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

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