Delay Tolerant Networking B. Sipos
Internet-Draft RKF Engineering
Intended status: Standards Track M. Demmer
Expires: July 30, 2020 UC Berkeley
J. Ott
Aalto University
S. Perreault
January 27, 2020

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-18

Abstract

This document describes a TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). This version of the TCPCL protocol is based on implementation issues in the earlier TCPCL Version 3 of RFC7242 and updates to the Bundle Protocol (BP) contents, encodings, and convergence layer requirements in BP 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.

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

1.1. Scope

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

- The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [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 entities (peers) within a network or across an internet. The mapping of Node ID to potential CL protocol and network address is left to implementation and configuration of the BP Agent and its various potential routing strategies.

- Logic for routing bundles along a path toward a bundle's endpoint. This CL protocol is involved only in transporting bundles between adjacent nodes in a routing sequence.

- Policies or mechanisms for assigning X.509 certificates, provisioning, deploying, or accessing certificates and private keys, deploying or accessing certificate revocation lists (CRLs), or configuring security parameters on an individual entity or across a network.

- Uses of TLS which are not based on X.509 certificate authentication (see Section 8.10.2) or in which authentication is not available (see Section 8.10.1).

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

2. Requirements Language

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

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions specific to the TCPCL protocol.

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 entitys in the network.
>
> A TCPCL Entity MAY passivley 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 closing 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 trasnfer 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.

```
+--------------------------------------------+
|                 TCPCL Entity               |
|                                            |      +----------------+
|   +--------------------------------+       |      |                |-+
|   | Actively Inititated Session #1 +------------->| Other          | |
|   +--------------------------------+       |      | TCPCL Entity's | |
|                  ...                       |      | Passive        | |
|   +--------------------------------+       |      | Listener       | |
|   | Actively Inititated Session #n +------------->|                | |
|   +--------------------------------+       |      +----------------+ |
|                                            |       +-----------------+
|      +---------------------------+         |
|  +---| +---------------------------+       |      +----------------+
|  |   | | Optional Passive          |       |      |                |-+
|  |   +-| Listener(s)               +<-------------+                | |
|  |     +---------------------------+       |      |                | |
|  |                                         |      | Other          | |
|  |    +---------------------------------+  |      | TCPCL Entity's | |
|  +--->| Passively Inititated Session #1 +-------->| Active         | |
|  |    +---------------------------------+  |      | Initiator(s)   | |
|  |                                         |      |                | |
|  |    +---------------------------------+  |      |                | |
|  +--->| Passively Inititated Session #n +-------->|                | |
|       +---------------------------------+  |      +----------------+ |
|                                            |       +-----------------+
+--------------------------------------------+
```

Figure 2: The relationships between TCPCL entities

```
+----------------------------+              +--------------------------+
|    "Own" TCPCL Session     |              |  "Other" TCPCL Session   |
|                            |              |                          |
| +-----------------------+  |              |  +---------------------+ |
| |   TCP Connection      |  |              |  |    TCP Connection   | |
| |                       |  |              |  |                     | |
| | +-------------------+ |  |              |  | +-----------------+ | |
| | | Optional Inbound  | |  |              |  | |  Peer Outbound  | | |
| | | Transfer Stream   |<-[Seg]--[Seg]--[Seg]-| | Transfer Stream | | |
| | |       -----       |--[Ack]----[Ack]------->|       -----     | | |
| | |     RECEIVER      | |  |              |  | |      SENDER     | | |
| | +-------------------+ |  |              |  | +-----------------+ | |
| |                       |  |              |  |                     | |
| | +-------------------+ |  |              |  | +-----------------+ | |
| | | Optional Outbound | |  |              |  | |  Peer Inbound   | | |
| | | Transfer Stream   |------[Seg]---[Seg]---->| Transfer Stream | | |
| | |       -----       |<--[Ack]----[Ack]-[Ack]-|       -----     | | |
| | |      SENDER       | |  |              |  | |    RECEIVER     | | |
| | +-------------------+ |  |              |  | +-----------------+ | |
| +-----------------------+  |              |  +---------------------+ |
+----------------------------+              +--------------------------+
```

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. 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 defintion but a logical description of how the CL can interact with the BP agent. Each of these interactions can 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:

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

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

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

Terminated: The session has finished normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed: The TCPCL supports indication when the live/ idle sub-state of the session changes. This occurs only when the top-level session state is "Established". The session transitions from Idle to Live at the at the start of a transfer in either transfer stream; the session transitions from Live to Idle at the end of a transfer when the other transfer stream does not have an ongoing transfer. Because 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 transfer 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 transfer success. The TCPCL itself does not have a notion of transfer timeout.

Reception Initialized: The TCPCL supports indication to the reciver just before any transmssion data is sent. This corresponds to reception of the XFER_SEGMENT message with the START flag of 1.

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.

3.2. 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 negotiate the use of TLS security (as described in Section 4). Once contact negotiation is complete, TCPCL messaging is available and the session negotiation is used to set parameters of the TCPCL session. One of these parameters is a Node ID of each TCPCL Entity. This is

used to assist in routing and forwarding messages by the BP Agent and is part of the authentication capability provided by TLS.

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 negligable 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 sequence of logical segments of data within XFER_SEGMENT messages. Multiple bundles can be transmitted consecutively in a single direction on a single TCPCL connection. Segments from different bundles are never interleaved. Bundle interleaving can be accomplished by fragmentation at the BP layer or by establishing multiple TCPCL sessions between the same peers. There is no fundamental limit on the number of TCPCL sessions which a single node can establish beyond the limit imposed by the number of available (ephemeral) TCP ports of the passive entity.

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 ending 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). Regardless of the reason, session termination is initiated by one of the entities and responded-to by the other as illustrated by Figure 13 and Figure 14. Even when there are no transfers queued or in-progress, the session termination procedure allows each entity to distinguish between a clean end to a session and the TCP connection being closed because of some underlying network issue.

Once a session is established, 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.

3.3. TCPCL States and Transitions

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

```
     +-------+
     | START |
     +-------+
         |
     TCP Establishment
         |
         V
   +-----------+            +---------------------+
   |    TCP    |----------->|  Contact / Session  |
   | Connected |            |     Negotiation     |
   +-----------+            +---------------------+
                                       |
          +-----Session Parameters-----+
          |         Negotiated
          V
   +-------------+                     +-------------+
   | Established |----New Transfer---->| Established |
   |   Session   |                     |   Session   |
   |    Idle     |<---Transfers Done---|     Live    |
   +-------------+                     +-------------+
         |                                    |
         +------------------------------------+
         |
         V
   +-------------+
   | Established |                    +-------------+
   |   Session   |----Transfers------>|     TCP     |
   |   Ending    |      Done          | Terminating |
   +-------------+                    +-------------+
                                              |
        +----------TCP Close Message----------+
        |
        V
    +-------+
    |  END  |
    +-------+
```

Figure 4: Top-level states of a TCPCL session

Notes on Established Session states:

Session "Live" means transmitting or reeiving over a transfer stream.

Session "Idle" means no transmission/reception over a transfer stream.

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

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

```
   +-------+
   | START |
   +-------+
       |
   TCP Connecting
       V
   +-----------+
   |    TCP    |            +---------+
   | Connected |--Send CH-->| Waiting |--Timeout-->[TCPCLOSE]
   +-----------+            +---------+
                                 |
                             Recevied CH
                                 V
                               [PCH]
```

Figure 5: Contact Initiation as Active Entity

```
   +-----------+             +---------+
   |   TCP     |--Wait for-->| Waiting |--Timeout-->[TCPCLOSE]
   | Connected |     CH      +---------+
   +-----------+                  |
                             Received CH
                                  V
                          +-----------------+
                          | Preparing reply |--Send CH-->[PSI]
                          +-----------------+
```

Figure 6: Contact Initiation as Passive Entity

```
   +-----------+
   |  Peer CH  |
   | available |
   +-----------+
         |
    Validate and
     Negotiate
         V
    +------------+
    | Negotiated |----Failure---->[TCPCLOSE]
    +------------+                      ^
       |       |                        |
     No TLS    +----Negotiate---+       |
       V              TLS       |    Failure
     +-----------+              V       |
     |   TCPCL   |            +---------------+
     | Messaging |<--Success--| TLS Finished |
     | Available |            +---------------+
     +-----------+
```

Figure 7: Processing of Contact Header [PCH]

Session negotiation involves exchanging a session initialization (SESS_INIT) message in both directions and deriving a negotiated state from the two messages. The session negotiation sequencing is performed either as the active or passive entity, and is illustrated in Figure 8 and Figure 9 respectively which both share the data validation and analyze final states of Figure 10. The validation here includes certificate validation and authentication when TLS is used for the session.

```
   +-----------+
   |   TCPCL   |                   +---------+
   | Messaging |--Send SESS_INIT-->| Waiting |--Timeout-->[ST]
   | Available |                   +---------+
   +-----------+                       |
                               Recevied SESS_INIT
                                       |
                                       V
                                     [PSI]
```

Figure 8: Session Initiation [SI] as Active Entity

```
 +-----------+
 |   TCPCL   |                  +---------+
 | Messaging |----Wait for ---->| Waiting |--Timeout-->[ST]
 | Available |    SESS_INIT     +---------+
 +-----------+                       |
                             Recevied SESS_INIT
                                     |
                             +-----------------+
                             | Preparing reply |--Send SESS_INIT-->[PSI]
                             +-----------------+
```

Figure 9: Session Initiation [SI] as Passive Entity

```
   +----------------+
   | Peer SESS_INIT |
   |   available    |
   +----------------+
           |
      Validate and
       Negotiate
           V
      +------------+
      | Negotiated |---Failure--->[ST]
      +------------+
           |
        Success
           V
     +--------------+
     | Established  |
     | Session Idle |
     +--------------+
```

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.

```
                                          +--Send XFER_SEGMENT--+
   +--------+                             |                     |
   | Stream |                       +-------------+             |
   |  Idle  |---Send XFER_SEGMENT-->| In Progress |<------------+
   +--------+                        +-------------+
                                          |
        +---------All segments sent-------+
        |
        V
   +---------+                       +--------+
   | Waiting |---- Receive Final---->| Stream |
   | for Ack |       XFER_ACK        |  IDLE  |
   +---------+                       +--------+
```

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.

```
                                            +-Receive XFER_SEGMENT-+
   +--------+                               |    Send XFER_ACK     |
   | Stream |                         +-------------+              |
   |  Idle  |--Receive XFER_SEGMENT-->| In Progress |<-------------+
   +--------+                         +-------------+
                                            |
        +--------Sent Final XFER_ACK--------+
        |
        V
   +--------+
   | Stream |
   |  Idle  |
   +--------+
```

Figure 12: Transfer receiver states

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

```
   +-----------+                   +---------+
   |  Session  |--Send SESS_TERM-->| Session |
   | Live/Idle |                   | Ending  |
   +-----------+                   +---------+
```

Figure 13: Session Termination [ST] from the Initiator

```
   +-----------+                   +---------+
   |  Session  |--Send SESS_TERM-->| Session |
   | Live/Idle |                   | Ending  |
   +-----------+<------+           +---------+
         |             |
    Receive SESS_TERM  |
         |             |
         +-------------+
```

Figure 14: Session Termination [ST] from the Responder

3.4. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation polcy 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 than 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 the two extremes of minimum overhead (large MRU, single-segment) and predictable message sizing (small MRU, highly segmented). Different policies can be applied to each transfer stream to and from any particular node. Additionally, future header and transfer extension types can apply further nuance to transfer policies and policy negotiation.

3.5. 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 can transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream. Although this example only demonstrates a single bundle transmission, it is also possible to pipeline multiple XFER_SEGMENT messages for different bundles without necessarily waiting for XFER_ACK messages to be returned for each one. However, interleaving data segments from different bundles is not allowed.

No errors or rejections are shown in this example.

```
                Entity A                             Entity B
                ========                             ========
       +-------------------------+
       |  Open TCP Connnection   | ->      +-------------------------+
       +-------------------------+      <- |    Accept Connection    |
                                           +-------------------------+

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

       +-------------------------+
```

```
       |        SESS_INIT        | ->      +-------------------------+
       +-------------------------+      <- |        SESS_INIT        |
                                           +-------------------------+

       +-------------------------+
       |   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] |
                                           +-------------------------+

       +-------------------------+
       |       SESS_TERM         | ->      +-------------------------+
       +-------------------------+      <- |        SESS_TERM        |
                                           +-------------------------+
       +-------------------------+         +-------------------------+
       |        TCP Close        | ->   <- |        TCP Close        |
       +-------------------------+         +-------------------------+
```

Figure 15: An example of the flow of protocol messages on a single TCP Session between two entities

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating entities. It is up to the implementation to decide how and when session setup is triggered. For example, some sessions 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 back-off mechanism to increase this delay in case of repeated failures. The upper limit on a re-attempt back-off is implementation defined but SHOULD be no longer than one minute before signaling to the BP agent that a connection cannot be made.

Once a TCP connection is established, the active entity SHALL immediately transmit its contact header. Upon reception of a contact header, the passive entity SHALL transmit its contact header. If the passive entity does not receive a Contact Header after some implementation-defined time duration after TCP connection is established, the entity SHALL close the TCP connection. The ordering of the contact header exchange allows the passive entity to avoid allocating resources to a potential TCPCL session until after a valid contact header has been received from the passive entity. This ordering also allows the passive peer to adapt to alternate TCPCL protocol versions.

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

4.2. Contact Header

This section describes the format of the contact header and the meaning of its fields.

If an entity is capable of exchanging messages according to TLS 1.2 [RFC5246] or any successors [RFC8446] that are compatible with TLS 1.2, the CAN_TLS flag within its contanct header SHALL be set to 1. This behavor prefers the use of TLS when possible, even if security policy does not allow or require authentication. This follows the opportunistic security model of [RFC7435].

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:

```
                       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       |
  +---------------+---------------+
```

Figure 16: Contact Header Format

See Section 4.3 for details on the use of each of these contact header fields.

The fields of the contact header are:

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

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

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

Name	Code	Description
CAN_TLS	0x01	If 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 ending it.

The first negotiation is on the TCPCL protocol version to use. The active entity always sends its Contact Header first and waits for a response from the passive entity. The active entity can repeatedly attempt different protocol versions in descending order until the passive entity accepts one with a corresponding Contact Header reply. Only upon response of a Contact Header from the passive entity 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 entity receives a Contact Header with a different protocol version than the one sent earlier on the TCP connection, the TCP connection SHALL be closed. If the active entity receives a SESS_TERM message with 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 entity receives a contact header containing a version that is greater than the current version of the TCPCL that the node implements, then the node SHALL shutdown the session with a reason code of "Version mismatch". If the passive entity receives a contact header with a version that is lower than the 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.

Once established, the lifetime of a TLS session SHALL be bound to the lifetime of the underlying TCP connection. Immediately prior to actively ending a TLS session after TCPCL session termination, the peer which sent the original (non-reply) SESS_TERM message SHOULD follow the Closure Alert procedure of [RFC5246] to cleanly terminate the TLS session. Because each TCPCL message is either fixed-length or self-indicates its length, the lack of a TLS Closure Alert will not cause data truncation or corruption.

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

4.4.1. TLS Handshake

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 TLS 1.2 [RFC5246] or any successors that are compatible with TLS 1.2. 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.

The parameters within each TLS negotiation are implementation dependent but any TCPCL node SHALL follow all recommended practices of BCP 195 [RFC7525], or any updates or successors that become part

of BCP 195. Within each TLS handshake, the following requirements apply (using the rough order in which they occur):

Client Hello: When a resolved host name was used to establish the TCP connection, the TLS ClientHello SHOULD include a Server Name Indication (SNI) from the active entity in accordance with [RFC6066]. When present, the SNI SHALL contain the same host name used to establish the TCP connection. Note: The SNI text is the network-layer name for the passive entity, which is not the Node ID of that entity.

Server Certificate: The passive entity SHALL supply a certificate within the TLS handshake to allow authentication of its side of the session. When assigned a stable host name or address, the passive entity certificate SHOULD contain a subjectAltName entry which authenticates that host name or address. The passive entity certificate SHOULD contain a subjectAltName entry of type uniformResourceIdentifier which authenticates the Node ID of the peer. The passive entity MAY use the SNI host name to choose an appropriate server-side certificate which authenticates that host name and corresponding Node ID.

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

Client Certificate: The active entity SHALL supply a certificate chain within the TLS handshake to allow authentication of its side of the session. When assigned a stable host name or address, the active entity certificate SHOULD contain a subjectAltName entry which authenticates that host name or address. The active entity certificate SHOULD contain a subjectAltName entry of type uniformResourceIdentifier which authenticates the Node ID of the peer.

All certificates supplied during TLS handshake SHALL conform with the profile of [RFC5280], or any updates or successors to that profile. When a certificate is supplied during TLS handshake, the full certification chain SHOULD be included unless security policy indicates that is unnecessary.

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

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

4.4.2. TLS Authentication

Using X.509 certificates exchanged during the TLS handshake, each of the entities can attempt to authenticate its peer at the network layer (host name and address) and at the application layer (BP Node ID). The Node ID exchanged in the Session Initialization is likely to be used by the BP agent for making transfer and routing decisions, so attempting host name validation is optional while attempting Node ID validation is required. The logic for attempting validation is separate from the logic for handling the result of validation, which is based on local security policy.

By using the SNI host name (see Section 4.4.1) a single passive entity can act as a convergence layer for multiple BP agents with distinct Node IDs. When this "virtual host" behavior is used, the host name is used as the indication of which BP Node the passive entity is attempting to communicate with. A virtual host CL entity can be authenticated by a certificate containing all of the host names and/or Node IDs being hosted or by several certificates each authenticating a single host name and/or Node ID.

Any certificate received during TLS handshake SHALL be validated up to one or more trusted certificate authority (CA) certificates. If certificate validation fails or if security policy disallows a certificate for any reason, the entity SHALL terminate the session (with a reason code of "Contact Failure").

Either during or immediately after the TLS handshake, each side of the TCP connection SHOULD perform host name validation of its peer in accordance with [RFC6125] unless it is not needed by security policy. The active entity SHALL attempt to authenticate the host name (of the passive entity) used to initiate the TCP connection. The active entity MAY attempt to authenticate the IP address of the other side of the TCP connection. The passive entity SHALL attempt to authenticate the IP address of the other side of the TCP connection. The passive entity MAY use the IP address to resolve one or more host names of the active entity and attempt to authenticate those. If host name validation fails (including failure because the certificate does not contain any DNS-ID) and security policy disallows an unauthticated host, the entity SHALL terminate the session (with a reason code of "Contact Failure").

Immediately before Session Parameter Negotiation, each side of the session SHALL perform Node ID validation of its peer as described

below. Node ID validation SHALL succeed if the associated certificate contains a subjectAltName entry of type uniformResourceIdentifier whose value matches the Node ID of the TCPCL entity. Unless specified otherwise by the definition of the URI scheme being authenticated, URI matching of Node IDs SHALL use the URI comparison logic of [RFC3986] and scheme-based normalization of those schemes specified in [I-D.ietf-dtn-bpbis]. This is similar to the URI-ID of [RFC6125] but does not require any structure to the scheme-specific-part of the URI. A URI scheme can refine this "exact match" logic with rules about how Node IDs within that scheme are to be compared with the certificate-authenticated URI. If Node ID validation fails (including failure because the certificate does not contain any subjectAltName URI) and security policy disallows an unauthticated Node ID, the entity SHALL terminate the session (with a reason code of "Contact Failure").

4.4.3. Example TLS Initiation

A summary of a typical TLS use is shown in the sequence in Figure 17 below.

```
                Entity A                             Entity B
               active peer                          passive peer

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

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

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

                         Host name validation occurs.
                      Secured TCPCL messaging can begin.

       +-------------------------+
       |        SESS_INIT        | ->      +-------------------------+
       +-------------------------+    <- |        SESS_INIT        |
                                           +-------------------------+

                          Node ID validation occurs.
                  Session is established, transfers can begin.

       +-------------------------+
       |       SESS_TERM         | ->      +-------------------------+
       +-------------------------+    <- |        SESS_TERM        |
                                           +-------------------------+

       +-------------------------+
       |    TLS Closure Alert    | ->      +-------------------------+
       +-------------------------+    <- |    TLS Closure Alert    |
                                           +-------------------------+

       +-------------------------+         +-------------------------+
       |        TCP Close        | ->   <- |        TCP Close        |
       +-------------------------+         +-------------------------+
```

Figure 17: A simple visual example of TCPCL TLS Establishment between two entities

4.5. Message Header

After the initial exchange of a contact header, 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 18: 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.

Name	Code	Description
SESS_INIT	0x07	Contains the session parameter inputs from one of the entities, as described in Section 4.6.
SESS_TERM	0x05	Indicates that one of the entities participating in the session wishes to cleanly terminate the session, as described in Section 6.
XFER_SEGMENT	0x01	Indicates the transmission of a segment of bundle data, as described in Section 5.2.2.
XFER_ACK	0x02	Acknowledges reception of a data segment, as described in Section 5.2.3.
XFER_REFUSE	0x03	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.2.4.
KEEPALIVE	0x04	Used to keep TCPCL session active, as described in Section 5.1.1.
MSG_REJECT	0x06	Contains a TCPCL message rejection, as described in Section 5.1.2.

Table 2: TCPCL Message Types

4.6. Session Initialization Message (SESS_INIT)

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

The format of a SESS_INIT message is as follows in Figure 19.

```
                   +-----------------------------+
                   |       Message Header        |
                   +-----------------------------+
                   |   Keepalive Interval (U16)  |
                   +-----------------------------+
                   |       Segment MRU (U64)     |
                   +-----------------------------+
                   |      Transfer MRU (U64)     |
                   +-----------------------------+
                   |     Node ID Length (U16)    |
                   +-----------------------------+
                   |    Node ID Data (variable)  |
                   +-----------------------------+
                   |      Session Extension      |
                   |      Items Length (U32)     |
                   +-----------------------------+
                   |      Session Extension      |
                   |         Items (var.)        |
                   +-----------------------------+
```

Figure 19: SESS_INIT Format

The fields of the SESS_INIT message are:

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

Node ID Length and Node ID Data: Together these fields represent a variable-length text string. The Node ID Length is a 16-bit unsigned integer indicating the number of octets of Node ID Data to follow. A zero-length Node ID SHALL be used to indicate the lack of Node ID rather than a truly empty Node ID. This case allows an entity to avoid exposing Node ID information on an untrusted network. A non-zero-length Node ID Data SHALL contain the UTF-8 encoded Node ID of the Entity which sent the SESS_INIT message. Every Node ID SHALL be a URI consistent with the requirements of [RFC3986] and the URI schemes of [I-D.ietf-dtn-bpbis]. The Node ID itself can be authenticated as described in Section 4.4.2.

Session Extension Length and Session Extension Items: 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 4.8. The full set of Session Extension Items apply for the duration of the TCPCL session to follow. The order and mulitplicity of these Session Extension Items MAY be significant, as defined in the associated type specification(s).

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 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. If the

either Transfer MRU or Segment MRU is unacceptable, the node SHALL terminate the session with a reason code of "Contact Failure".

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of this two contact headers' Keepalive Interval. The Session Keepalive interval is a parameter for the behavior described in Section 5.1.1. If the Session Keepalive interval is unacceptable, the node SHALL terminate the session with a reason code of "Contact Failure".

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. Because this state is negotiated over an unsecured medium, there is a risk of a TLS Stripping as described in Section 8. 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 reason is different than a failure of TLS handshake or TLS authentication after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is true and acceptable then 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.

4.8. Session Extension Items

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

The fields of the Session Extension Item are:

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

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification does not define any extension types directly, but does allocate an IANA registry for such codes (see Section 9.3).

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

Item Value: A variable-length data field 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 specifications SHOULD avoid the use of large data lengths, as no bundle transfers can begin until the full extension data is sent.

```
                     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...                                 |
+---------------+---------------+---------------+---------------+
```

Figure 20: 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

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 (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 2) with no additional data. Both sides SHALL send a KEEPALIVE message whenever the negotiated interval has elapsed with no transmission of any message (KEEPALIVE or other).

If no message (KEEPALIVE or other) has been received in a session after some implementation-defined time duration, then the node SHALL 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 keepalive 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 21.

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

Figure 21: 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 is indicated by the 'START' flag and the last segment is indicated by the 'END' flag.

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

```
                  +------------------------------+
                  |       Message Header         |
                  +------------------------------+
                  |     Message Flags (U8)       |
                  +------------------------------+
                  |      Transfer ID (U64)       |
                  +------------------------------+
                  |     Transfer Extension       |
                  |      Items Length (U32)      |
                  |   (only for START segment)   |
                  +------------------------------+
                  |     Transfer Extension       |
                  |         Items (var.)         |
                  |   (only for START segment)   |
                  +------------------------------+
                  |      Data length (U64)       |
                  +------------------------------+
                  | Data contents (octet string) |
                  +------------------------------+
```

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 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

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

Transfer Extension Length and Transfer Extension Items: Together these fields represent protocol extension data for this specification. The Transfer Extension Length and Transfer Extension Item fields SHALL only be present when the 'START' flag is set to 1 on the message. 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 5.2.5. The full set of transfer extension items apply only to the assoicated single transfer. The order and mulitplicity of these transfer extension items MAY be significant, as defined in the associated type specification(s).

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' flag SHALL be set to 1 when transmitting the first segment of a transfer. The 'END' flag SHALL be set to 1 when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the 'START' and 'END' flags SHALL be set to 1.

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' flag set to 1. 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.

5.2.3. Data Acknowledgments (XFER_ACK)

Although the TCP transport provides reliable transfer of data between transport peers, the typical BSD sockets interface provides no means to inform a sending application of when the receiving application has processed some amount of transmitted data. Thus, after transmitting some data, the TCPCL needs an additional mechanism to determine whether the receiving agent has successfully received 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 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

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

Acknowledged length: A 64-bit unsigned integer indicating the total number of octets in the transfer 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 SHOULD transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream.

For example, suppose the sending 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.

5.2.4. 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 an 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 any 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 6.

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

Name	Code	Description
Unknown	0x00	Reason for refusal is unknown or not specified.
Extension Failure	0x01	A failure processing the Transfer Extension Items ha occurred.
Completed	0x02	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
No Resources	0x03	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	0x04	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 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 SHALL 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' flag set to 1, indicating the start of a new transfer, and with a distinct Transfer ID value.

5.2.5. Transfer Extension Items

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

The fields of the Transfer Extension Item are:

Flags: A one-octet field containing generic bit flags about the Item, which are listed in Table 7. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver. If a TCPCL node receives a Transfer Extension Item with an unknown Item Type and the CRITICAL flag is 1, the node SHALL refuse the transfer with an XFER_REFUSE reason code of "Extension Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification allocates an IANA registry for such codes (see Section 9.4).

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

Item Value: A variable-length data field 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 specifications SHOULD avoid the use of large data lengths, as the associated transfer cannot begin until the full extension data is sent.

```
                        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...                                 |
   +---------------+---------------+---------------+---------------+
```

Figure 25: 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 7: Transfer Extension Item Flags

5.2.5.1. Transfer Length Extension

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

Multiple Transfer Length extension items SHALL NOT occur within the same transfer. The lack of a Transfer Length extension item in any transfer SHALL NOT imply anything about the potential length of the transfer. The Transfer Length extension SHALL be assigned transfer extension type ID 0x0001.

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

The format of the Transfer Length data is as follows in Figure 26.

```
                      +----------------------+
                      |  Total Length (U64)  |
                      +----------------------+
```

Figure 26: Format of Transfer Length data

The fields of the Transfer Length extension are:

Total Length: A 64-bit unsigned integer indicating the size of the data-to-be-transferred. The Total Length field SHALL be treated as authoritative by the receiver. If, for whatever reason, the actual total length of bundle data received differs from the value indicated by the Total Length value, the receiver SHALL treat the transmitted data as invalid.

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 flag of a SESS_TERM message SHALL be 0. Upon receiving a SESS_TERM message after not sending a SESS_TERM message in the same session, an entity SHALL send an acknowledging SESS_TERM message. When sent to acknowledge a termination, a SESS_TERM message SHALL have identical data content from the message being acknowledged except for the REPLY flag, which is set to 1 to indicate 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 for the remainder of the session. After receving 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. Not acknowledging received segments can result in unnecessary retransmission. When performing an unclean shutodwn, 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 close 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 27.

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

Figure 27: 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. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

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

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

Table 8: SESS_TERM Flags

Name	Code	Description
Unknown	0x00	A termination reason is not available.
Idle timeout	0x01	The session is being closed due to idleness.
Version mismatch	0x02	The node cannot conform to the specified TCPCL protocol version.
Busy	0x03	The node is too busy to handle the current session.
Contact Failure	0x04	The node cannot interpret or negotiate contact header option.
Resource Exhaustion	0x05	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 recevied), an entity SHALL 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 SHALL close the TCP connection. Each TCPCL message is contiguous in the octet stream and has no ability to be cut short and/or preempted by 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 ending 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 can exist.

An example implementation of the this draft of TCPCLv4 has been created as a GitHub project [github-dtn-bpbis-tcpcl] and is intented 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

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

8.1. Threat: Passive Leak of Node Data

When used without TLS security, the TCPCL exposes the Node ID and other configuration data to passive eavesdroppers. This occurs even when no transfers occur within a TCPCL session. This can be avoided by always using TLS, even if authentication is not available (see Section 8.10).

8.2. Threat: Passive Leak of Bundle Data

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 bundle security mechanisms defined in [I-D.ietf-dtn-bpsec] are to be used instead.

When used without TLS security, the TCPCL exposes all bundle data to passive eavesdroppers. This can be avoided by always using TLS, even if authentication is not available (see Section 8.10).

8.3. Threat: TCPCL Version Downgrade

When a TCPCL entity supports multiple versions of the protocol it is possible for a malicious or misconfigued peer to use an older version of TCPCL which does not support transport security. It is up to security policies within each TCPCL node to ensure that the TCPCL version in use meets transport security requirements.

8.4. Threat: Transport Security Stripping

When security policy allows non-TLS sessions, TCPCL does not protect against active network attackers. It is possible for a man-in-the-middle attacker to set the CAN_TLS flag to 0 on either side of the

Contact Header exchange. This leads to the "SSL Stripping" attack described in [RFC7457].

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

The negotiated use of TLS is identical behavior to STARTTLS use in [RFC2595] and [RFC4511].

8.5. Threat: Weak Ciphersuite Downgrade

Even when using TLS to secure the TCPCL session, the actual ciphersuite negotiated between the TLS peers can be insecure. Recommendations for ciphersuite use are included in BCP 195 [RFC7525]. It is up to security policies within each TCPCL node to ensure that the negotiated TLS ciphersuite meets transport security requirements.

8.6. Threat: Invalid Certificate Use

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

8.7. Threat: Symmetric Key Overuse

Even with a secure block cipher and securely-established session keys, there are limits to the amount of plaintext which can be safely encrypted with a given set of keys as described in [AEAD-LIMITS]. When permitted by the negotiated TLS version (see [RFC8446]), it is advisable to take advantage of session key updates to avoid those limits. When key updates are not possible, establishing new TCPCL/TLS session is an alternative to limit session key use.

8.8. Threat: BP Node Impersonation

The certificates exchanged by TLS enable authentication of peer host name and Node ID, but it is possible that a peer either not provide a valid certificate or that the certificate does not validate either

the host name or Node ID of the peer. Having a CA-validated certificate does not alone guarantee the identity of the network host or BP node from which the certificate is provided; additional validation procedures in Section 4.4.1 bind the host name or node ID based on the contents of the certificate.

The host name validation is a weaker form of authentication, because even if a peer is operating on an authenticated network host name it can provide an invalid Node ID and cause bundles to be "leaked" to an invalid node. Especially in DTN environments, network names and addresses of nodes can be time-variable so binding a certificate to a Node ID is a more stable identity. Trusting an authenticated host name can be feasable on a network secured by a private CA but is not advisable on the Internet when using a variety of public CAs.

Node ID validation ensures that the peer to which a bundle is transferred is in fact the node which the BP Agent expects it to be. It is a reasonable policy to skip host name validation if certificates can be guaranteed to validate the peer's Node ID. In circumstances where certificates can only be issued to network host names, Node ID validation is not possible but it could be reasonable to assume that a trusted host is not going to present an invalid Node ID.

8.9. Threat: Denial of Service

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

A malicious entity can continually establish TCPCL sessions and delay sending of protocol-required data to trigger timeouts. The victim entity can block TCP connections from network peers which are thought to be incorrectly behaving within TCPCL.

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

There is the possibility of a "data dribble" attack in which an entity presents a very small Segment MRU which causes transfers to be split among an large number of very small segments and causes the segmentation overhead to overwhelm the network througput. Similarly, an entity can present a very small Transfer MRU which will cause resources to be wasted on establishing and upkeeping a TCPCL session

over which a bundle could never be transferred. The victim entity can terminate the session during the negotiation of Section 4.7 if the MRUs are unacceptable.

The keepalive mechanism can be abused to waste throughput within a network link which would otherwise be usable for bundle transmissions. Due to the quantization of the Keepalive Interval parameter the smallest Session Keepalive is one second, which should be long enough to not flood the link. The victim entity can terminate the session during the negotiation of Section 4.7 if the Keepalive Interval is unacceptable.

8.10. Alternate Uses of TLS

This specification makes use of public key infrastructure (PKI) certificate validation and authentication within TLS. There are alternate uses of TLS which are not necessarily incompatible with the security goals of this specification, but are outside of the scope of this document.

8.10.1. TLS Without Authentication

In environments where PKI is available but there are restrictions on the issuance of certificates (including the contents of certificates), it may be possible to make use of TLS in a way which authenticates only the passive entity of a TCPCL session or which does not authenticate either entity. Using TLS in a way which does not authenticate both peer entities of each TCPCL session is outside of the scope of this document.

8.10.2. Non-Certificate TLS Use

In environments where PKI is unavailable, alternate uses of TLS which do not require certificates such as [RFC5489] are available and can be used to ensure confidentality within TCPCL. Using non-PKI node authentication methods is outside of the scope of this document.

9. IANA Considerations

Registration procedures referred to in this section are defined in [RFC8126].

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

9.1. Port Number

Within the port registry of [IANA-PORTS], TCP port number 4556 has been previously assigned as the default port for the TCP convergence layer in [RFC7242]. This assignment is unchanged by TCPCL version 4, but the assignment reference is updated to this specification. 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. The related assignments for UDP and DCCP port 4556 (both registered by [RFC7122]) are unchanged.

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

9.2. Protocol Versions

IANA has created, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version Numbers". The version number table is updated to include this specification. The registration procedure is RFC Required.

Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4	TCPCLv4	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 [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Session Extension Types" and initialize it with the contents of Table 10. The registration procedure is Expert Review 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.

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

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

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

Table 10: 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 [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Transfer Extension Types" and initialize it with the contents of Table 11. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

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

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

Code	Transfer Extension Type
0x0000	Reserved
0x0001	Transfer Length Extension
0x0002--0x7FFF	Unassigned
0x8000--0xFFFF	Private/Experimental Use

Table 11: 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 [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" and initialize it with the contents of Table 12. The registration procedure is RFC Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

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

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

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

Table 12: 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 [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" and initialize it with the contents of Table 13. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new XFER_REFUSE reason codes need to define the meaning of the reason and disambiguate it with pre-exisiting reasons. Each refusal reason needs to be usable by the receving BP Agent to make retransmission or re-routing decisions.

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

Code	Refusal Reason
0x00	Unknown
0x01	Extension Failure
0x02	Completed
0x03	No Resources
0x04	Retransmit
0x05--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 13: 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 [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 SESS_TERM Reason Codes" and initialize it with the contents of Table 14. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new SESS_TERM reason codes need to define the meaning of the reason and disambiguate it with pre-exisiting reasons. Each termination reason needs to be usable by the receving BP Agent to make re-connection decisions.

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

Code	Termination Reason
0x00	Unknown
0x01	Idle timeout
0x02	Version mismatch
0x03	Busy
0x04	Contact Failure
0x05	Resource Exhaustion
0x06--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 14: 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 [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 MSG_REJECT Reason Codes" and initialize it with the contents of Table 15. The registration procedure is Specification Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new MSG_REJECT reason codes need to define the meaning of the reason and disambiguate it with pre-exisiting reasons. Each rejection reason needs to be usable by the receving TCPCL Entity to make message sequencing and/or session termination decisions.

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

Code	Rejection Reason
0x00	reserved
0x01	Message Type Unknown
0x02	Message Unsupported
0x03	Message Unexpected
0x04--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 15: MSG_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

[I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-19 (work in progress), January 2020.

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

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

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

11.2. Informative References

[AEAD-LIMITS] Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", August 2017, <http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf>.

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

[I-D.ietf-dtn-bpsec] Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-15 (work in progress), January 2020.

[RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <https://www.rfc-editor.org/info/rfc2595>.

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

[RFC4511] Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI 10.17487/RFC4511, June 2006, <https://www.rfc-editor.org/info/rfc4511>.

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

[RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, DOI 10.17487/RFC5489, March 2009, <https://www.rfc-editor.org/info/rfc5489>.

[RFC7122] Kruse, H., Jero, S., and S. Ostermann, "Datagram Convergence Layers for the Delay- and Disruption-Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP)", RFC 7122, DOI 10.17487/RFC7122, March 2014, <https://www.rfc-editor.org/info/rfc7122>.

[RFC7242] Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol", RFC 7242, DOI 10.17487/RFC7242, June 2014, <https://www.rfc-editor.org/info/rfc7242>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <https://www.rfc-editor.org/info/rfc7435>.

[RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <https://www.rfc-editor.org/info/rfc7457>.

[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:

- Split contact header into pre-TLS protocol negotiation and SESS_INIT parameter negotiation. The contact header is now fixed-length.

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

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

- Renamed "Endpoint ID" to "Node ID" to conform with BPv7 terminology.

- Added session extension capability.

- Added transfer extension capability. Moved transfer total length into an extension item.

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

- Segments of all new IANA registries are reserved for private/ experimental use.

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

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

- Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.

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

- Renamed SHUTDOWN to SESS_TERM to deconflict term "shutdown" related to TCP connections.

- 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:

- Added contact negotiation failure SESS_TERM reason code.

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

- Added TLS session security mechanism.

- 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: ott@in.tum.de

Simon Perreault

Quebec, QC
Canada

Email: simon@per.reau.lt