Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2018

S. Burleigh
D. Horres
JPL, Calif. Inst. Of Technology
K. Viswanathan
M. Benson
F. Templin
Boeing Research & Technology

March 2, 2018

Architecture for Delay-Tolerant Key Administration
draft-burleigh-dtnwg-dtka-01.txt

Abstract

Delay-Tolerant Key Administration (DTKA) is a system of public-key management protocols intended for use in Delay Tolerant Networking (DTN). This document outlines a DTKA proposal for space-based communications, which are characterized by long communication delays and planned communication contacts.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Delay-Tolerant Key Administration (DTKA) is a system of public-key management protocols intended for use in Delay Tolerant Networking (DTN) [RFC4838]. This document outlines a DTKA proposal for space-based communications, which are characterized by long communication delays and planned communication contacts. The proposal satisfies the requirements for DTN Security Key Management [I-D.templin-dtnskmreq].

1.1. Motivation and Design Strategy

In general, on-demand interactive communications, like client-server interactions, are not feasible in DTN's network model. Terrestrial public-key management protocols require on-demand interactions with remote computing nodes to distribute and validate public-keys. For example, terrestrial public-key management protocols require on-demand interactions with a remote trusted authority (Certificate Revocation List (CRL)) to determine if a given public-key certificate has been revoked or not. Therefore, such terrestrial public-key management protocols cannot be used in DTN.

Periodic and planned communications are an inherent property of space-based communication systems. Thus, the core principle of DTKA is to exploit this property of space-based communication systems in order to avoid the need for on-demand interactive communications for key management. Therefore, the design strategy for DTKA is to pro-actively distribute authenticated public-keys to all nodes in a given DTN instance in advance to ensure that keys will be available when needed even if there may be significant delays or disruptions. This design strategy is to be contrasted with protocols for terrestrial Public-Key Infrastructures, in which authenticated public-keys are exchanged interactively, just-in-time and on demand.

1.2. Scope

DTKA was originally designed for space-based DTN environments, but it could potentially be used in terrestrial DTN environments as well.

1.3. About This Document

This document describes the high-level architecture of DTKA and lists the architectural entities, their interactions, and system assumptions.

1.4. Related Documents

The following documents provide the necessary context for the high-level design described in this document.

RFC 4838 [RFC4838] describes the architecture for DTN and is titled, "Delay-Tolerant Networking Architecture." That document provides a high-level overview of DTN architecture and the decisions that underpin the DTN architecture.

RFC 5050 [RFC5050] describes the protocol and message formats for DTN and is titled, "Bundle Protocol Specification." That document provides the format of the network protocol message for DTN, called a Bundle, along with descriptions of processes for generating, sending, forwarding, and receiving Bundles. It also specifies an encoding format called SDNV (Self-Delimiting Numeric Values) for use in DTN. Each bundle comprises a primary block, a payload block, and zero or more additional extension blocks. A node may receive and process a bundle even when the bundle contains one or more extension blocks that the node is not equipped to process.

RFC 6257 [RFC6257] is titled, "Bundle Security Protocol Specification." It specifies the message formats and processing rules for providing three types of security services to bundles, namely: confidentiality, integrity, and authentication. It does not specify mechanisms for key management. Rather, it assumes that cryptographic keys are somehow in place and then specifies how the keys shall be used to provide the security services. Additionally, it attempts to standardize a default cipher suite for DTN.

The revised Internet Draft [I-D.ietf-dtn-bpsec] for DTN communication security is titled, "Bundle Security Protocol Specification (bpsec)." When compared with RFC 6257, it is silent on concepts such as Security Regions, at-most-once-delivery option, and cipher suite specification. It deletes the Bundle Authentication Block and generalized the Payload Integrity and Payload Confidentiality Blocks to Block Integrity Block and Block Confidentiality Block. It provides more detailed specification for bundle canonicalization and rules for processing bundles received from other nodes. Like RFC 6257, the draft does not describe any key management mechanisms for DTN but assumes that a suitable key management mechanism shall be in place.

5050bis [I-D.ietf-dtn-bpbis] is an Internet Draft on standards track that intends to update RFC 5050. It introduces a new concept called "node ID" as distinguished from the existing concept of "endpoint ID": a single DTN endpoint may contain one or more nodes. It also migrates some primary block fields into extension blocks, making the primary block immutable. In the Security Considerations section, 5050bis explicitly describes end-to-end security using Block-Integrity-Block (BIB) and Block-Confidentiality-Block (BCB). It does not specify link-by-link security considerations to be part of the bundle protocol level using the Bundle-Authenticity-Block (BAB), which was described in RFC 6257. The convergence layers may provide link-by-link authentication instead of bundle protocol agent.

The Internet Draft for specifying requirements for DTN Key Management [I-D.templin-dtnskmreq] is titled, "DTN Security Key

Management - Requirements and Design." It sketches nine requirements and four design criteria for DTN Key Management system. The last two requirements are the need to support revocation in a delay tolerant manner. It also specifies the requirements for avoiding single points of failure and opportunities for the presence of multiple key management authorities.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT","SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. Lower case uses of these words are not to be interpreted as carrying RFC2119 significance.

3. High Level Architecture

```
+---------+               +-----------+2.ListOfAuthenticated +------+
|Key      |1.(NodeID, Key)| Key       |(Node ID, Key)        | Key  |
|Owner    +---------------> Authority +----------------------> User |
|(Node ID)|               |           |                      |      |
+---^-----+               +-----------+                      +---^--+
    |                     3.Secure Communications                |
    +--------------------------------------------------------------+
```

Figure 1: Abstract Data-Flow-Diagram for DTKA

The DTKA system includes Key Owners, Key Agents (which, in aggregate, constitute the Key Authority), and Key Users. For the sake of simplicity and to promote conceptual clarity, Figure 1 shows a single Key Agent. In order to avoid a single point-of-trust, DTKA provides mechanisms to distribute the Key Authority function among one or more DTKA Key Agents using an erasure-coding technique. This trust-distributing mechanism is discussed later in this document.

Each Key Owner has a unique DTN Node ID and chooses its own public-private key pair. In order to associate a public-key (Key) with its Node ID, a Key Owner sends an assertion of the form: (Node ID, Key) to the Key Authority. Key Owners need to authenticate their respective keys in one of two ways:

1. in the case of out-of-band bootstrapping, Key Authority shall rely on the physical security of the out-of-band channel to validate the integrity of the received message and the Key Owner needs to sign the association (Node ID, Key) using the private key corresponding to the Key. Association realized using such an

interaction will be called Out-of-band-authentication (OOBAuth); or,

2. in the case of in-band authentication, the Key Owner or a trusted third-party needs to sign the association (Node ID, Key) using the private key corresponding to the previously authenticated and currently effective public-key for that NodeID. If the Key Owner signs the association, there will be roll-over association. If a trusted third-party signs the association, the association will have the type endorse so as to indicate an endorsement.

Each Key User periodically receives a list of authenticated public-keys from the Key Authority and uses the authenticated public-keys as needed.

3.1. Application Domains

DTN can be used in various theatres such as space, airspace, on earth and at sea. There can be more than one installation of DTN in each of these theatres administered by different administrative entities, which may represent countries, companies and institutions. A particular installation of DTN with a single aggregate key authority is called an Application Domain.

3.2. System Entities

The architectural elements of DTKA, which shall henceforth be called DTKA Entities, are listed below.

DTKA Key Agent (DTKA-KA)
DTKA-KA is part of the root of trust for authenticated distribution of public-keys for a given application domain. All DTKA Entities must have physically authenticated public-keys of all DTKA Key Agents (DTKA-KAs), which together constitute the DTKA Key Authority for a given application domain.

DTKA Key Owner[Node ID] (DTKA-KO[Node ID])
DTKA-KO[Node ID] is a computing node that has possession of the private key corresponding to the public-key authenticated for a given Node Identity (Node ID) by the DTKA-KAs for the Key Owner’s application domain.

DTKA Key User (DTKA-KU)
DTKA-KU is a computing node that receives authenticated public-keys from DTKA-KAs and distributes the same within a single computing machine through a suitable Interprocess Communication mechanism, which is outside the scope of this document.

DTKA Key Manager (DTKA-KM) and DTKA Key Manager Client (DTKA-KMC)
DTKA-KM is a DTKA Key User that receives authenticated public-keys from DTKA-KAs and distributes the same over a communication network to DTKA-KMCs, which are not DTKA Entities. DTKA-KMC can be a DTN node that can receive key distributions from DTKA KMs. The communication and security protocols for the interactions between DTKA-KMs and DTKA-KMCs are outside the scope of this document.

3.3. System Interconnections

```
       ++                                                          ++
     ++ +----------------------------------------------------------+ ++
    ++ Sub-second One-Way-Light-Time (OWLT) and Rarely Disrupted Link ++
     ++ +----------^--------------------------------------^--------+ ++
       ++          |                                      |        ++
           +-------v-------+                      +-------v-------+
           | DTKA Entity   |                      | DTKA Entity   |
           |               |                      |               |
           | +-----------+ |                      | +-----------+ |
           | | DTKA Key  | | .................... | | DTKA Key  | |
           | | Agent     | |                      | | Agent     | |
           | +-----------+ |                      | +-----------+ |
           | +-----------+ |                      | +-----------+ |
           | |   TSM     | |                      | |   TSM     | |
           | +-----------+ |                      | +-----------+ |
           +-------^-------+                      +-------^-------+
       ++          |                                      |        ++
     ++ +----------v--------------------------------------v--------+ ++
    ++         Communication Link with Delay and Disruptions          ++
     ++ +----------^--------------------------------------^--------+ ++
       ++          |                                      |        ++
           +-------v-------+                      +-------v-------+
           | DTKA Entity   |                      | DTKA Entity   |
           |               |                      |               |
           | +-----------+ |                      | +-----------+ |
           | | DTKA Key  | |                      | | DTKA Key  | |
           | | Owner     | |                      | | User      | |
           | +-----------+ |                      | +-----------+ |
           | +-----------+ |                      | +-----------+ |
           | |Autonomous | |                      | |Autonomous | |
           | |Clock      | |                      | |Clock      | |
           | +-----------+ |                      | +-----------+ |
           +---------------+                      +---------------+
```

Figure 2: DTKA System Interconnections

Figure 2 depicts the system level interconnections that are assumed for the design of DTKA. An application domain can have one or more DTKA-KAs, all of which must be interconnected using a sub-second One-Way-Light-Time (OWLT) and rarely disrupted link. Such communication link can be realized using terrestrial Internet or specialized point-to-point space communication techniques. This link shall be used by the DTKA-KAs to synchronize between themselves. The DTKA-KAs shall run a reliable Time Synchronization Mechanism (TSM), like the Network Time Protocol (NTP) service. TSM shall ensure that time is synchronized between the DTKA-KAs that realize the DTKA Key Authority for a given application domain.

A potentially delayed and frequently disrupted communication link is assumed to interconnect DTKA-KAs, DTKA-KOs and DTKA-KUs. This delayed-and-disrupted communication link is used by the DTKA-KAs to multicast authenticated public-key associations to DTKA-KUs. The DTKA-KUs are assumed to have access to autonomous clocks. Autonomous clocks keep time without external correction signals and with an allowed drift in the order of a few seconds. But, delay-tolerant mechanisms for clock agreement such as issuance of UTC offsets in network management messages may be present.

3.4. Architectural Assumption on Communication

In the subsequent sections, it shall be seen that DTKA-KAs shall dispatch updates to the list of authenticated public-keys in the system using erasure coding techniques. It is evident that at least a sub-set of such communications updates must reach each DTKA-KU. Therefore, the DTN upon which the DTKA operates must satisfy the following communication assumption before DTKA can function along expected lines: all addressed receivers MUST receive sufficient number of bundles from the DTKA-KAs before the earliest effective time among the effective times of all public-key associations in the payloads of the bundles. Note that the underlying DTN will not be aware of the effective times of the public-key associations in the payloads of the bundles.

The above assumption can be restated using DTKA protocol terminologies, which shall be seen in the subsequent sections, as follows: All addressed receivers MUST receive enough of the code blocks for a given bulletin to enable reassembly of that bulletin before the earliest effective-time among all associations in the bulletin.

3.5. System Security Configuration

The current public-keys of all designated DTKA-KAs for a given application domain must be securely configured into every DTKA-KA and DTKA-KU that needs to participate in that application domain; this is a pre-condition for initializing those DTKA-Entities. This process will ensure that the DTKA Agents are established as the root of trust for that application domain.

4. Detailed Design

4.1. Message Formats

Every DTKA-KA in an application domain will receive requests for associating public-keys with Node IDs from the respective DTKA-KOs. After authenticating the requests and any pending revocations (as described in Section 4.4 below), every DTKA-KA reaches consensus with all other DTKA-KAs, which constitute the Key Authority in its application domain, on some subset of the authenticated requests and revocations. The protocols and algorithms for DTKA-KA consensus is an implementation aspect and out-of-scope of this document. After each successful consensus, each DTKA-KA must increment its local value called Bulletin Serial Number (BSN) and agree on the Trust Model Number (TMN) for the bulletin. Thereafter, each DTKA-KA must independently multicast to all participating DTKA Entities the subset of authenticated list of address-and-key associations on which consensus was reached along with the new BSN value. The message format for this multicast, which is called a Bulletin, supports message authentication and redundancy. The goal of message authentication is to prevent DTKA Entities’ acceptance of malicious multicast messages issued by hostile nodes. The goal of message redundancy is to ensure that a minimal set of collaborating DTKA-KAs in the application domain will be able to successfully send out-of-band-authentication (OOBAuth) or revocations for address-and-key associations to all DTKA Entities -- the DTKA Entities need not know which DTKA-KAs are not collaborating.

As mentioned previously, bulletin is a collection of association blocks (or Key Information Message [KIM] data structure) such that each association block represents a single association of a Node ID with a public-key as depicted in Figure 3. Each block issues either an out-of-band-authentication (OOBAuth) or endorse or revoke or roll-over instruction to the receiving DTKA Entities, which use the key information message to execute the instruction locally. The semantics for each of the instruction shall be described in subsequent sections. The block labelled "Bulletin Hash" contains the cryptographic hash computed over all association blocks (key information messages), the Bulletin Serial Number (BSN) and the Trust

Model Number (TMN) in that bulletin. The BSN is a unique and sequential bulletin identifier. The TMN is a unique identifier to indicate the trust model configuration that is to be used to validate this bulletin. The trust model configuration can be seen as a list of DTKA-KAs (Key Agents), who are trusted to authenticate this bulletin to all DTKA Users in the system. The trust model configuration is also used to indicate the t-out-of-n threshold configuration that shall be described in the next paragraph. The precise syntax for the trust model configuration is a DTKA-KA implementation aspect and, is therefore, out-of-scope of this document.

```
+--------+--------+---+---+----------------------------------------+  +---+
|Bulletin|Bulletin|TMN|BSN|Key information message (KIM):          |  |   |
|        |hash    |   |   |{([Node ID, Effective Time, Public Key],|..|KIM|
|        |        |   |   |  OOBAuth/endorse/revoke/roll_over)}    |  |   |
+--------+--------+---+---+----------------------------------------+  +---+
```

Figure 3: Bulletin

After forming a bulletin, a (Q+k)-erasure code algorithm is used to create an erasure code for the bulletin. Thus, receipt of any Q distinct code blocks will be sufficient to decode the bulletin. To ensure that the incapacity or compromise -- or veto (disagreement on bulletin content) -- of any single DTKA-KA will not result in malfunction of the key authority mechanism, each DTKA-KA is assigned primary responsibility for transmission of some limited subset of the bulletin’s code blocks and backup responsibility for some other limited subset. The assigned code block subsets for the various DTKA-KAs are selected in such a way that every code block is to be transmitted by two different DTKA-KAs. The combination of these two transmission redundancy mechanisms (parity code blocks and duplicate transmissions), together with reliable bundle transmission at the convergence layer under bundle multicast, minimizes the likelihood of any client node being unable to reconstruct the bulletin from the code blocks it receives.

During system initialization, the code-block assignments for each DTKA-KA need to be configured into every DTKA Entity. The code-block assignment for the example considered in this section is shown below in the table, in which an x-mark depicts the assignment of a code block to a DTKA-KA. It can be seen in the table that, in this example, code-blocks from at least five (t=5) DTKA-KAs must be received before the bulletin blocks can be decoded. Also, when all DTKA-KAs multicast their pre-defined code blocks, n * m (8*3 = 24) code blocks are sent to all DTKA Entities. To further defend against a compromised DTKA-KA node introducing error into the key distribution system:

- o All nodes are informed of the code block subsets for which all DTKA-KA nodes are responsible. Any received code block that was transmitted by a DTKA-KA node which was not responsible for transmission of that code block is discarded by the receiving node.

- o Each code block issued by the each KA is signed under that KA’s private key. The bulletin hash in the code block uniquely identifies the bulletin that will be reconstructed using this code block. Every transmitted code block is accompanied by the bulletin hash. All - and only - code blocks tagged with the unique bulletin hash are reassembled into the bulletin identified by that hash.

- o If the hash of a bulletin reassembled from a set of received code blocks is not verified then, for each the DTKA-KA node that transmitted one or more of the constituent code blocks, all code blocks transmitted by that node are excluded from the reassembled bulletin. Upon success, the node whose transmitted code blocks had been excluded from the reassembled bulletin may be presumed to be compromised.

Code Block Numbers (0 to (Q + k -1))	0	1	2	3	4	5	6	7
KA 1	x	x	x					
KA 2		x	x	x				
KA 3			x	x	x			
KA 4				x	x	x		
KA 5					x	x	x	
KA 6						x	x	x
KA 7	x						x	x
KA 8	x	x						x

Table 1: Example Trust-Table: Code Blocks Assignments for Key Agents

The message format for transmitting the assigned code-blocks by each DTKA-KA is shown in Figure 4. Note that each such message is the payload of a Bundle and that the authenticity of that payload is nominally protected by a Block Integrity Block containing a digital signature computed in the private key of the issuing Key Agent; the message itself contains no self-authentication material. Reading the figure left to right, we have: (a) a field indicating the type of this message, namely Bulletin code block; (b) the bulletin hash as defined in Figure 3; (c) the trust model number that provides trust-table configuration as depicted in Table 1; (d) the code-block numbers (column numbers in the trust-table) for which code-blocks are available in this code block message; and, (e) the specified code-blocks from the DTKA-KA. The identity of the DTKA-KA (KAx) that generated the code blocks must be available as the source node ID of the DTN bundle that carried this code block message. KAx is used to validate the signature in the bundle’s Block Integrity Block before the message is delivered to DTKA by the underlying DTN protocol layer.

```
      +----------+----------+---+-----------+------------+
      | Bulletin | Bulletin |TMN| Code Block| Code Blocks|
      | Codeblock| Hash     |   | Numbers   |            |
      +----------+----------+---+-----------+------------+
```

Figure 4: Message Format for Code Blocks

4.2. Non-receipt of a Bulletin

When a DTKA Entity receives sufficient number of bulletin blocks from the DTKA Key Agents, it can reconstruct the corresponding bulletin with its unique Bundle Serial Number (BSN) in the format depicted in Figure 3. By maintaining a historical list of successfully reconstructed BSN values and analysing for gaps in the BSN historical list, a DTKA Entity can detect non-receipt of past bulletins. Upon such a detection, the DTKA Entity must send a request to all the DTKA Key Agents in the format specified in Figure 5 in order to request retransmission of the past bulletins for a given BSN value. When such request is received by a DTKA Key Agent, the DTKA Key Agent must retransmit its code blocks corresponding to the requested BSN only to the requesting DTKA Entity in the format shown in Figure 4. The security for this communication from the DTKA Key Agents must be similar to the security for the bulletin broadcast communication. Upon receiving sufficient number of bulletin blocks for the requested bulletin, the requesting DTKA Entity may reconstruct the bulletin and verify that the bulletin with the requested BSN has indeed been received. Thereupon, the DTKA Entity must update its BSN historical list with the received BSN value.

```
      +----------+-----------+--------------+-------+
      | Bulletin | Request   | Requesting   |List   |
      | Request  | Timestamp | Node (Node ID)|of BSNs|
      +----------+-----------+--------------+-------+
```

Figure 5: Message Format for Requesting Retransmission of Bulletin

4.3. Node Registration

In order to register a new DTKA-KO in the system, DTKA requires the DTKA-KO with a Node ID (DTKA-KO[Node ID]) to generate a public-private key pair and preserve the secrecy of its private key. The DTKA-KO[Node ID] needs to generate an association message of the form (Node ID, effective-time, public-key), where effective-time specifies the start time after which the public-key is valid. That is, each bundle sent by this node is to be authenticated using the node’s most recently effective public key whose effective time is less than the bundle’s creation time. The DTKA-KO[Node ID] must send the association message, along with a signature on the message using its private key, to the DTKA-KA as depicted in Figure 6. Since DTKA-KA would not have seen the association of the public-key to that key owner previously, it cannot trust that the message indeed originated from DTKA-KO[Node ID]. Therefore, for registration purposes, this initial message from the DTKA-KO[Node ID] to the DTKA-KA MUST be protected by transmitting it over an independently (e.g., physically) authenticated channel. The independently authenticated channel can be realized by physically securing the access to the DTKA-KA server, using a physical communication medium, such as a USB dongle, and manually verifying the authenticity of the communication from the DTKA-KO. The manual verification is a one-time process for a given Key Owner. When an application domain has more than one DTKA-KA (KAx), the message from DTKA-KO[Node ID] must be sent to each DTKA-KA (KAx) in a similarly secure manner.

Although the messages to DTKA-KA (KAx) are independently authenticated, the DTKA-KO[Node ID] must sign the association message using its private key. The signature is not intended to cryptographically authenticate the message but only to prove to the DTKA-KA that the DTKA-KO[Node ID] is indeed in possession of the private key. This self-signed message by the DTKA-KO is useful to ensure that the physical courier, which is used to realize the physically authenticated channel, has not tampered the message sent by the DTKA-KO to the DTKA-KA. Additionally, the self-signed message is useful to audit the operations of the DTKA-KA.

```
      +---------------------+                      +---------------+
      |   DTKA-KO[Node ID]  |                      | DTKA-KA (KAx) |
      +--------|------------+                      +-------|-------+
             **|*******************************************|**
             * |                                           | *
             * |{[Node ID, Effective Time, Public Key, s]  | *
             * | such that s = Sign(Private Key, [Node ID, | *
             * |        Effective Time, Public Key,...])}  | *
             * +-------------------------------------------> *
             * |                                           | *
             * | Physically authenticated channel(USB,...) | *
             **|*******************************************|**
               |                                           |
               |                                        +--+
               | TRUE = Verify(Public Key, s, [Node ID, |  |
               |       Effective Time, Public Key,...]) |  |
               |                                        +-->
               |                                           |
               +                                           +
```

Figure 6: Interaction Diagram 1: Node Registration

Each DTKA-KA will insert the received association message into its next bulletin (refer to Figure 3), for multicast as an out-of-band-authentication (OOBAuth) association: when registration is received through a physically authenticated channel. The bulletin will be multicast to all DTKA Entities using the protocol described in Section 4.7.

As an alternative to the use of a physically authenticated channel, the registration association message may be sent by a trusted third-party node whose authenticated public key is already registered and known to all DTKA-KAs, so that the message may be authenticated by verifying the digital signature (formed using the trusted third-party node’s current private key) in the BIB of the bundle containing the message. Each DTKA-KA will insert such association requests in its next bulletin for multicast as an endorsed association by tagging the corresponding Key Information message in the bulletin as "endorse" (refer to Figure 3). The bulletin will be multicast to all DTKA Entities using the protocol described in Section 4.7.

4.4. Key Revocation

Manual decisions trigger the key revocation procedure. Every DTKA-KA in an application domain is assumed to have a human operator who can trigger the revocation process. When a key is to be revoked, the human operator will need to authenticate to the respective DTKA-KA (KAx) server, identify the public-key and Node ID to be revoked, and instruct that DTKA-KA (KAx) revocation software to schedule a revocation message. The revocation software in DTKA-KA (KAx) will multicast a message as shown in Figure 7. The process for sending out the code-blocks by all the DTKA-KAs with this revocation information is described in Section 4.1.

```
    +---------+                                              +---------+
    | DTKA-KA |                                              | DTKA-KA |
    |  (KAx)  |                                              |  (KAy)  |
    +---|-----+                                              +---|-----+
        |                                                        |
        |                                                        |
        | Multicast{m , s  such that                             |
        | s = Sign(PrivateKey[KAx], m) and                       |
        | m = [KAx, Revoke, [Node ID, Effective Time, Pub Key)]} |
        +-------------------------------------------------------->
        |                                                        |
        |                                                     +--+
        |         TRUE = Verify(Public Key[KAx], s, [Node ID, |  |
        |               Effective Time, New Public Key,...])  |  |
        |                                                     +-->
        |                                                        |
        +                                                        +
```

Figure 7: Interaction Diagram 1.1: Key Revocation

4.5. Key Roll-over

When a DTKA-KO[Node ID] has been registered by the DTKA-KA using the protocol described in Figure 6, the DTKA-KO[Node ID] can periodically roll-over to a new public-private key pair by following the key roll-over protocol described in Figure 8. The protocol for key roll-over is similar to the one for key registration except that: (a) the protocol can be executed using DTN bundles issued by the KO itself without requiring any independently secured out-of-band communication channels; and, (b) the old (current) public-key is used to authenticate the association of the new public-key with the Node ID for that DTKA KO. The DTKA-KO [Node ID] must send this message to every key agent in its application domain. Upon accepting the roll-over message from the DTKA-KO[Node ID], each key agent will schedule the roll-over instruction for identified Node ID and public-key in its next bulletin as described in Section 4.1. A DTKA-KO can schedule any number of future roll-overs but the number of such roll-over schedules may need to be limited to avoid Denial of Service attacks by registered nodes -- but this topic is beyond the scope of this document.

```
      +---------------------+                          +---------------+
      |   DTKA-KO[Node ID]  |                          | DTKA-KA (KAx) |
      +--------|------------+                          +-------|-------+
               |                                               |
               |{[Node ID, Effective Time, New Public Key, s]  |
               | such that s = Sign(Old Private Key, [Node ID, |
               |        Effective Time, New Public Key,...])}  |
               +----------------------------------------------->
               |                                               |
               |                                               |
               |                                            +--+
               | TRUE = Verify(Old Public Key, s, [Node ID, |  |
               |       Effective Time, New Public Key,...]) |  |
               |                                            +-->
               |                                               |
               +                                               +
```

Figure 8: Interaction Diagram 1.2: Key Rollover

4.6. Key Endorsement

When a DTKA-KO[Node ID] is not registered and does not have access to any out-of-band authentication channel with any DTKA-KA, the DTKA-KO[Node ID] will need to have access to an out-of-band authentication channel for a trusted third-party (TTP), which is registered with the DTKA-KA. Upon receiving the (Node ID, Key, Effective time) information from the DTKA-KO[Node ID] over the out-of-band authentication channel, the trusted third-party needs to relay that information to the DTKA-KA by signing the information under its authenticated public key. This interaction is depicted in Figure 9. Upon accepting the endorse message from the trusted third-party, each key agent will schedule an endorse instruction for identified Node ID and public-key in its next bulletin as described in Section 4.1.

```
      +---------------------+                          +---------------+
      |Trusted third-party  |                          | DTKA-KA (KAx) |
      +--------|------------+                          +-------|-------+
               |                                               |
               |{[Node ID, Public Key, Effective Time, s]      |
               | such that s = Sign(TTP Private Key, [Node ID, |
               |         Public Key, Effective Time,...])}     |
               +----------------------------------------------->
               |                                               |
               |                                               |
               |                                            +--+
               | TRUE = Verify(TTP Public Key, s, [Node ID, |  |
               |       Effective Time, Public Key,...])     |  |
               |                                            +-->
               |                                               |
               +                                               +
```

Figure 9: Interaction Diagram 1.3: Key Endorsement

4.7. Key Distribution

Each DTKA-KA collects multiple out-of-band-authentication (OOBAuth), revocation, roll-over and endorse association messages from different parties by following the protocols described in Section 4.3, Section 4.4, Section 4.5 and Figure 9. Then, each DTKA-KA forms and sends multicast communications for the code blocks for its bulletin to all DTKA Key Users as explained in Section 4.1. The DTKA-KUs verify the authenticity of each code block from all the DTKA-KAs before using the code blocks to decode the bulletin, which will contains out-of-band key authentication, key revocation, key roll-over and endorse instructions. The DTKA-KUs perform these instructions in their respective local key database. This interaction between the DTKA-KAs and the DTKA-KUs of an application domain is shown in Figure 10.

```
      +-------------------+                            +--------------+
      |   DTKA-KA (KAx)   |                            |   DTKA-KU    |
      +---------+---------+                            +--------+-----+
                |                                               |
                +---------------------------------------------->
                | Send(KAx, BulletinHash, CodeBlockNumber       |
                |   CodeBlock of Bulletin such that Bulletin =  |
                |   array[Node ID, Effective Time, PubKey])     |
                |                                               |
                |                                           +---+
                |                   Wait for code blocks    |   |
                |                   from key authorities    +--->
                |                                               |
                |                                           +---+
                |            Decode Bulletin using code     |   |
                |            blocks from key authorities    +--->
                +                                               +
```

Figure 10: Interaction Diagram 2: Bulk Key Distribution

4.8. Secure Communications

After receiving out-of-band-authentication (OOBAuth), roll-over or endorse information, every DTKA-KU shall have authenticated public-keys for different Node IDs in its local database. These authenticated public-keys can be used to authenticate messages received from the DTKA-KO[Node ID] and to send confidential messages to the DTKA-KO[Node ID] after the specified effective-time for each Node ID and public-key pair. This interaction is specified in Figure 11.

```
      +---------------------+                     +--------------+
      |  DTKA-KO[Node ID]   |                     |   DTKA-KU    |
      +--------+-----------+                      +--------+-----+
               |Secure communications                      |
               |[Node ID, Creation Time, Signature, Data] |
               +------------------------------------------>
               |                                          |
               |Secure communications                     |
               |[Node ID, Creation Time, Encrypted Data]  |
               <------------------------------------------+
               |                                          |
               +                                          +
```

Figure 11: Interaction Diagram 3: Secure communication

4.9. Communication Stack View

DTKA is designed to be a special DTN application that shall perform key management operations using the services of the Bundle Protocol and BPSec. DTKA will use BP, which in turn will use BPSec to authenticate the messages containing the public-keys that are subsequently to be used by BPSec for securing future communications as shown in Figure 12.

```
     +-------------+---------+   +-------------------------------------+
     |Applications |  DTKA   +--->                                     |
     +-------------+---------+   |        Local Keys Database          |
     |Bundle Protocol (BPSec)<---+(Node ID, Public Key, Effective Time)|
     +-----------------------+   |                                     |
     |   Convergence Layer   |   +-------------------------------------+
     +-----------------------+
     |    Transport Layer    |
     +-----------------------+
     |     Network Layer     |
     +------------------------+
     |    Physical Layer     |
     +-----------------------+
```

Figure 12: Block Diagram: Communication Stack View for DTKA

5. IANA Considerations

This document potentially contains IANA considerations depending on the design choices adopted for future work. But, in its present form, there are no immediate IANA considerations.

6. Security Considerations

Security issues and considerations are discussed through out this document.

7. References

7.1. Normative References

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

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

[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>.

7.2. Informative References

[I-D.templin-dtnskmreq]
Templin, F. and S. Burleigh, "DTN Security Key Management - Requirements and Design", draft-templin-dtnskmreq-00 (work in progress), February 2015.

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

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

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

Authors’ Addresses

Scott Burleigh
JPL, Calif. Inst. Of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
USA

Email: Scott.Burleigh@jpl.nasa.gov

David Horres
JPL, Calif. Inst. Of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
USA

Email: David.C.Horres@jpl.nasa.gov

Kapali Viswanathan
Boeing Research & Technology
Boeing International Corporation India Private Limited
A Block, 4th Floor, Lake View Building
Bagmane Tech Park, C.V. Raman Nagar
Bangalore, KA 560093
IN

Email: kapaleeswaran.viswanathan@boeing.com

Michael W. Benson
Boeing Research & Technology
The Boeing Company
499 Boeing Boulevard
Huntsville, AL 35824
USA

Email: michael.w.benson@boeing.com

Fred L. Templin
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
USA

Email: fltemplin@acm.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 3, 2019

S. Burleigh
D. Horres
JPL, Calif. Inst. Of Technology
K. Viswanathan
M. Benson
F. Templin
Boeing Research & Technology

August 30, 2018

Architecture for Delay-Tolerant Key Administration
draft-burleigh-dtnwg-dtka-02.txt

Abstract

Delay-Tolerant Key Administration (DTKA) is a system of public-key management protocols intended for use in Delay Tolerant Networking (DTN). This document outlines a DTKA proposal for space-based communications, which are characterized by long communication delays and planned communication contacts.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Delay-Tolerant Key Administration (DTKA) is a system of public-key management protocols intended for use in Delay Tolerant Networking (DTN) [RFC4838]. This document outlines a DTKA proposal for space-based communications, which are characterized by long communication delays and planned communication contacts. The proposal satisfies the requirements for DTN Security Key Management [I-D.templin-dtnskmreq].

1.1. Motivation and Design Strategy

In general, on-demand interactive communications, like client-server interactions, are not feasible in DTN's network model. Terrestrial public-key management protocols require on-demand interactions with remote computing nodes to distribute and validate public-keys. For example, terrestrial public-key management protocols require on-demand interactions with a remote trusted authority (Certificate Revocation List (CRL)) to determine if a given public-key certificate has been revoked or not. Therefore, such terrestrial public-key management protocols cannot be used in DTN.

Periodic and planned communications are an inherent property of space-based communication systems. Thus, the core principle of DTKA is to exploit this property of space-based communication systems in order to avoid the need for on-demand interactive communications for key management. Therefore, the design strategy for DTKA is to pro-actively distribute authenticated public-keys to all nodes in a given DTN instance in advance to ensure that keys will be available when needed even if there may be significant delays or disruptions. This design strategy is to be contrasted with protocols for terrestrial Public-Key Infrastructures, in which authenticated public-keys are exchanged interactively, just-in-time and on demand.

1.2. Scope

DTKA was originally designed for space-based DTN environments, but it could potentially be used in terrestrial DTN environments as well.

1.3. About This Document

This document describes the high-level architecture of DTKA and lists the architectural entities, their interactions, and system assumptions.

1.4. Related Documents

The following documents provide the necessary context for the high-level design described in this document.

RFC 4838 [RFC4838] describes the architecture for DTN and is titled, "Delay-Tolerant Networking Architecture." That document provides a high-level overview of DTN architecture and the decisions that underpin the DTN architecture.

RFC 5050 [RFC5050] describes the protocol and message formats for DTN and is titled, "Bundle Protocol Specification." That document provides the format of the network protocol message for DTN, called a Bundle, along with descriptions of processes for generating, sending, forwarding, and receiving Bundles. It also specifies an encoding format called SDNV (Self-Delimiting Numeric Values) for use in DTN. Each bundle comprises a primary block, a payload block, and zero or more additional extension blocks. A node may receive and process a bundle even when the bundle contains one or more extension blocks that the node is not equipped to process.

RFC 6257 [RFC6257] is titled, "Bundle Security Protocol Specification." It specifies the message formats and processing rules for providing three types of security services to bundles, namely: confidentiality, integrity, and authentication. It does not specify mechanisms for key management. Rather, it assumes that cryptographic keys are somehow in place and then specifies how the keys shall be used to provide the security services. Additionally, it attempts to standardize a default cipher suite for DTN.

The revised Internet Draft [I-D.ietf-dtn-bpsec] for DTN communication security is titled, "Bundle Security Protocol Specification (bpsec)." When compared with RFC 6257, it is silent on concepts such as Security Regions, at-most-once-delivery option, and cipher suite specification. It deletes the Bundle Authentication Block and generalized the Payload Integrity and Payload Confidentiality Blocks to Block Integrity Block and Block Confidentiality Block. It provides more detailed specification for bundle canonicalization and rules for processing bundles received from other nodes. Like RFC 6257, the draft does not describe any key management mechanisms for DTN but assumes that a suitable key management mechanism shall be in place.

5050bis [I-D.ietf-dtn-bpbis] is an Internet Draft on standards track that intends to update RFC 5050. It introduces a new concept called "node ID" as distinguished from the existing concept of "endpoint ID": a single DTN endpoint may contain one or more nodes. It also migrates some primary block fields into extension blocks, making the primary block immutable. In the Security Considerations section, 5050bis explicitly describes end-to-end security using Block-Integrity-Block (BIB) and Block-Confidentiality-Block (BCB). It does not specify link-by-link security considerations to be part of the bundle protocol level using the Bundle-Authenticity-Block (BAB), which was described in RFC 6257. The convergence layers may provide link-by-link authentication instead of bundle protocol agent.

The Internet Draft for specifying requirements for DTN Key Management [I-D.templin-dtnskmreq] is titled, "DTN Security Key

Management - Requirements and Design." It sketches nine requirements and four design criteria for DTN Key Management system. The last two requirements are the need to support revocation in a delay tolerant manner. It also specifies the requirements for avoiding single points of failure and opportunities for the presence of multiple key management authorities.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT","SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. Lower case uses of these words are not to be interpreted as carrying RFC2119 significance.

3. High Level Architecture

```
   +---------+               +-----------+2.ListOfAuthenticated +------+
   |Key      |1.(NodeID, Key)| Key       |(Node ID, Key)        | Key  |
   |Owner    +---------------> Authority +----------------------> User |
   |(Node ID)|               |           |                      |      |
   +---^-----+               +-----------+                      +---^--+
       |                     3.Secure Communications                |
       +------------------------------------------------------------+
```

Figure 1: Abstract Data-Flow-Diagram for DTKA

The DTKA system includes Key Owners, Key Agents (which, in aggregate, constitute the Key Authority), and Key Users. For the sake of simplicity and to promote conceptual clarity, Figure 1 shows a single Key Agent. In order to avoid a single point-of-trust, DTKA provides mechanisms to distribute the Key Authority function among one or more DTKA Key Agents using an erasure-coding technique. This trust-distributing mechanism is discussed later in this document.

Each Key Owner has a unique DTN Node ID and chooses its own public-private key pair. In order to associate a public-key (Key) with its Node ID, a Key Owner sends an assertion of the form: (Node ID, Key) to the Key Authority. Key Owners need to authenticate their respective keys in one of two ways:

1. in the case of out-of-band bootstrapping, Key Authority shall rely on the physical security of the out-of-band channel to validate the integrity of the received message and the Key Owner needs to sign the association (Node ID, Key) using the private key corresponding to the Key. Association realized using such an

interaction will be called Out-of-band-authentication (OOBAuth); or,

2. in the case of in-band authentication, the Key Owner or a trusted third-party needs to sign the association (Node ID, Key) using the private key corresponding to the previously authenticated and currently effective public-key for that NodeID. If the Key Owner signs the association, there will be roll-over association. If a trusted third-party signs the association, the association will have the type endorse so as to indicate an endorsement.

Each Key User periodically receives a list of authenticated public-keys from the Key Authority and uses the authenticated public-keys as needed.

3.1. Application Domains

DTN can be used in various theatres such as space, airspace, on earth and at sea. There can be more than one installation of DTN in each of these theatres administered by different administrative entities, which may represent countries, companies and institutions. A particular installation of DTN with a single aggregate key authority is called an Application Domain.

3.2. System Entities

The architectural elements of DTKA, which shall henceforth be called DTKA Entities, are listed below.

DTKA Key Agent (DTKA-KA)
DTKA-KA is part of the root of trust for authenticated distribution of public-keys for a given application domain. All DTKA Entities must have physically authenticated public-keys of all DTKA Key Agents (DTKA-KAs), which together constitute the DTKA Key Authority for a given application domain.

DTKA Key Owner[Node ID] (DTKA-KO[Node ID])
DTKA-KO[Node ID] is a computing node that has possession of the private key corresponding to the public-key authenticated for a given Node Identity (Node ID) by the DTKA-KAs for the Key Owner’s application domain.

DTKA Key User (DTKA-KU)
DTKA-KU is a computing node that receives authenticated public-keys from DTKA-KAs and distributes the same within a single computing machine through a suitable Interprocess Communication mechanism, which is outside the scope of this document.

   DTKA Key Manager (DTKA-KM) and DTKA Key Manager Client (DTKA-KMC)
      DTKA-KM is a DTKA Key User that receives authenticated public-keys
      from DTKA-KAs and distributes the same over a communication
      network to DTKA-KMCs, which are not DTKA Entities. DTKA-KMC can
      be a DTN node that can receive key distributions from DTKA KMs.
      The communication and security protocols for the interactions
      between DTKA-KMs and DTKA-KMCs are outside the scope of this
      document.

3.3. System Interconnections

```
       ++                                                          ++
     ++ +----------------------------------------------------------+ ++
    ++ Sub-second One-Way-Light-Time (OWLT) and Rarely Disrupted Link ++
     ++ +----------^--------------------------------------^--------+ ++
       ++          |                                      |        ++
           +-------v-------+                      +-------v-------+
           | DTKA Entity   |                      | DTKA Entity   |
           |               |                      |               |
           | +-----------+ |                      | +-----------+ |
           | | DTKA Key  | | .................... | | DTKA Key  | |
           | | Agent     | |                      | | Agent     | |
           | +-----------+ |                      | +-----------+ |
           | +-----------+ |                      | +-----------+ |
           | |   TSM     | |                      | |   TSM     | |
           | +-----------+ |                      | +-----------+ |
           +-------^-------+                      +-------^-------+
       ++          |                                      |        ++
     ++ +----------v--------------------------------------v--------+ ++
    ++         Communication Link with Delay and Disruptions          ++
     ++ +----------^--------------------------------------^--------+ ++
       ++          |                                      |        ++
           +-------v-------+                      +-------v-------+
           | DTKA Entity   |                      | DTKA Entity   |
           |               |                      |               |
           | +-----------+ |                      | +-----------+ |
           | | DTKA Key  | |                      | | DTKA Key  | |
           | | Owner     | |                      | | User      | |
           | +-----------+ |                      | +-----------+ |
           | +-----------+ |                      | +-----------+ |
           | |Autonomous | |                      | |Autonomous | |
           | |Clock      | |                      | |Clock      | |
           | +-----------+ |                      | +-----------+ |
           +---------------+                      +---------------+
```

Figure 2: DTKA System Interconnections

Figure 2 depicts the system level interconnections that are assumed for the design of DTKA. An application domain can have one or more DTKA-KAs, all of which must be interconnected using a sub-second One-Way-Light-Time (OWLT) and rarely disrupted link. Such communication link can be realized using terrestrial Internet or specialized point-to-point space communication techniques. This link shall be used by the DTKA-KAs to synchronize between themselves. The DTKA-KAs shall run a reliable Time Synchronization Mechanism (TSM), like the Network Time Protocol (NTP) service. TSM shall ensure that time is synchronized between the DTKA-KAs that realize the DTKA Key Authority for a given application domain.

A potentially delayed and frequently disrupted communication link is assumed to interconnect DTKA-KAs, DTKA-KOs and DTKA-KUs. This delayed-and-disrupted communication link is used by the DTKA-KAs to multicast authenticated public-key associations to DTKA-KUs. The DTKA-KUs are assumed to have access to autonomous clocks. Autonomous clocks keep time without external correction signals and with an allowed drift in the order of a few seconds. But, delay-tolerant mechanisms for clock agreement such as issuance of UTC offsets in network management messages may be present.

3.4. Architectural Assumption on Communication

In the subsequent sections, it shall be seen that DTKA-KAs shall dispatch updates to the list of authenticated public-keys in the system using erasure coding techniques. It is evident that at least a sub-set of such communications updates must reach each DTKA-KU. Therefore, the DTN upon which the DTKA operates must satisfy the following communication assumption before DTKA can function along expected lines: all addressed receivers MUST receive sufficient number of bundles from the DTKA-KAs before the earliest effective time among the effective times of all public-key associations in the payloads of the bundles. Note that the underlying DTN will not be aware of the effective times of the public-key associations in the payloads of the bundles.

The above assumption can be restated using DTKA protocol terminologies, which shall be seen in the subsequent sections, as follows: All addressed receivers MUST receive enough of the code blocks for a given bulletin to enable reassembly of that bulletin before the earliest effective-time among all associations in the bulletin.

3.5. System Security Configuration

The current public-keys of all designated DTKA-KAs for a given application domain must be securely configured into every DTKA-KA and DTKA-KU that needs to participate in that application domain; this is a pre-condition for initializing those DTKA-Entities. This process will ensure that the DTKA Agents are established as the root of trust for that application domain.

4. Detailed Design

4.1. Message Formats

Every DTKA-KA in an application domain will receive requests for associating public-keys with Node IDs from the respective DTKA-KOs. After authenticating the requests and any pending revocations (as described in Section 4.4 below), every DTKA-KA reaches consensus with all other DTKA-KAs, which constitute the Key Authority in its application domain, on some subset of the authenticated requests and revocations. The protocols and algorithms for DTKA-KA consensus is an implementation aspect and out-of-scope of this document. After each successful consensus, each DTKA-KA must increment its local value called Bulletin Serial Number (BSN) and agree on the Trust Model Number (TMN) for the bulletin. Thereafter, each DTKA-KA must independently multicast to all participating DTKA Entities the subset of authenticated list of address-and-key associations on which consensus was reached along with the new BSN value. The message format for this multicast, which is called a Bulletin, supports message authentication and redundancy. The goal of message authentication is to prevent DTKA Entities’ acceptance of malicious multicast messages issued by hostile nodes. The goal of message redundancy is to ensure that a minimal set of collaborating DTKA-KAs in the application domain will be able to successfully send out-of-band-authentication (OOBAuth) or revocations for address-and-key associations to all DTKA Entities -- the DTKA Entities need not know which DTKA-KAs are not collaborating.

As mentioned previously, bulletin is a collection of association blocks (or Key Information Message [KIM] data structure) such that each association block represents a single association of a Node ID with a public-key as depicted in Figure 3. Each block issues either an out-of-band-authentication (OOBAuth) or endorse or revoke or roll-over instruction to the receiving DTKA Entities, which use the key information message to execute the instruction locally. The semantics for each of the instruction shall be described in subsequent sections. The block labelled "Bulletin Hash" contains the cryptographic hash computed over all association blocks (key information messages), the Bulletin Serial Number (BSN) and the Trust

Model Number (TMN) in that bulletin. The BSN is a unique and sequential bulletin identifier. The TMN is a unique identifier to indicate the trust model configuration that is to be used to validate this bulletin. The trust model configuration can be seen as a list of DTKA-KAs (Key Agents), who are trusted to authenticate this bulletin to all DTKA Users in the system. The trust model configuration is also used to indicate the t-out-of-n threshold configuration that shall be described in the next paragraph. The precise syntax for the trust model configuration is a DTKA-KA implementation aspect and, is therefore, out-of-scope of this document.

```
+--------+--------+---+---+----------------------------------------+  +---+
|Bulletin|Bulletin|TMN|BSN|Key information message (KIM):          |  |   |
|        |hash    |   |   |{([Node ID, Effective Time, Public Key],|..|KIM|
|        |        |   |   |  OOBAuth/endorse/revoke/roll_over)}    |  |   |
+--------+--------+---+---+----------------------------------------+  +---+
```

Figure 3: Bulletin

After forming a bulletin, a (Q+k)-erasure code algorithm is used to create an erasure code for the bulletin. Thus, receipt of any Q distinct code blocks will be sufficient to decode the bulletin. To ensure that the incapacity or compromise -- or veto (disagreement on bulletin content) -- of any single DTKA-KA will not result in malfunction of the key authority mechanism, each DTKA-KA is assigned primary responsibility for transmission of some limited subset of the bulletin’s code blocks and backup responsibility for some other limited subset. The assigned code block subsets for the various DTKA-KAs are selected in such a way that every code block is to be transmitted by two different DTKA-KAs. The combination of these two transmission redundancy mechanisms (parity code blocks and duplicate transmissions), together with reliable bundle transmission at the convergence layer under bundle multicast, minimizes the likelihood of any client node being unable to reconstruct the bulletin from the code blocks it receives.

During system initialization, the code-block assignments for each DTKA-KA need to be configured into every DTKA Entity. The code-block assignment for the example considered in this section is shown below in the table, in which an x-mark depicts the assignment of a code block to a DTKA-KA. It can be seen in the table that, in this example, code-blocks from at least five (t=5) DTKA-KAs must be received before the bulletin blocks can be decoded. Also, when all DTKA-KAs multicast their pre-defined code blocks, n * m (8*3 = 24) code blocks are sent to all DTKA Entities. To further defend against a compromised DTKA-KA node introducing error into the key distribution system:

- All nodes are informed of the code block subsets for which all DTKA-KA nodes are responsible. Any received code block that was transmitted by a DTKA-KA node which was not responsible for transmission of that code block is discarded by the receiving node.

- Each code block issued by the each KA is signed under that KA’s private key. The bulletin hash in the code block uniquely identifies the bulletin that will be reconstructed using this code block. Every transmitted code block is accompanied by the bulletin hash. All - and only - code blocks tagged with the unique bulletin hash are reassembled into the bulletin identified by that hash.

- If the hash of a bulletin reassembled from a set of received code blocks is not verified then, for each the DTKA-KA node that transmitted one or more of the constituent code blocks, all code blocks transmitted by that node are excluded from the reassembled bulletin. Upon success, the node whose transmitted code blocks had been excluded from the reassembled bulletin may be presumed to be compromised.

Code Block Numbers (0 to (Q + k -1))	0	1	2	3	4	5	6	7
KA 1	x	x	x					
KA 2		x	x	x				
KA 3			x	x	x			
KA 4				x	x	x		
KA 5					x	x	x	
KA 6						x	x	x
KA 7	x						x	x
KA 8	x	x						x

Table 1: Example Trust-Table: Code Blocks Assignments for Key Agents

The message format for transmitting the assigned code-blocks by each DTKA-KA is shown in Figure 4. Note that each such message is the payload of a Bundle and that the authenticity of that payload is nominally protected by a Block Integrity Block containing a digital signature computed in the private key of the issuing Key Agent; the message itself contains no self-authentication material. Reading the figure left to right, we have: (a) a field indicating the type of this message, namely Bulletin code block; (b) the bulletin hash as defined in Figure 3; (c) the trust model number that provides trust-table configuration as depicted in Table 1; (d) the code-block numbers (column numbers in the trust-table) for which code-blocks are available in this code block message; and, (e) the specified code-blocks from the DTKA-KA. The identity of the DTKA-KA (KAx) that generated the code blocks must be available as the source node ID of the DTN bundle that carried this code block message. KAx is used to validate the signature in the bundle's Block Integrity Block before the message is delivered to DTKA by the underlying DTN protocol layer.

```
+----------+----------+---+-----------+------------+
| Bulletin | Bulletin |TMN| Code Block| Code Blocks|
| Codeblock| Hash     |   | Numbers   |            |
+----------+----------+---+-----------+------------+
```

Figure 4: Message Format for Code Blocks

4.2. Non-receipt of a Bulletin

When a DTKA Entity receives sufficient number of bulletin blocks from the DTKA Key Agents, it can reconstruct the corresponding bulletin with its unique Bundle Serial Number (BSN) in the format depicted in Figure 3. By maintaining a historical list of successfully reconstructed BSN values and analysing for gaps in the BSN historical list, a DTKA Entity can detect non-receipt of past bulletins. Upon such a detection, the DTKA Entity must send a request to all the DTKA Key Agents in the format specified in Figure 5 in order to request retransmission of the past bulletins for a given BSN value. When such request is received by a DTKA Key Agent, the DTKA Key Agent must retransmit its code blocks corresponding to the requested BSN only to the requesting DTKA Entity in the format shown in Figure 4. The security for this communication from the DTKA Key Agents must be similar to the security for the bulletin broadcast communication. Upon receiving sufficient number of bulletin blocks for the requested bulletin, the requesting DTKA Entity may reconstruct the bulletin and verify that the bulletin with the requested BSN has indeed been received. Thereupon, the DTKA Entity must update its BSN historical list with the received BSN value.

```
+----------+-----------+--------------+-------+
| Bulletin | Request   | Requesting   |List   |
| Request  | Timestamp | Node (Node ID)|of BSNs|
+----------+-----------+--------------+-------+
```

Figure 5: Message Format for Requesting Retransmission of Bulletin

4.3. Node Registration

In order to register a new DTKA-KO in the system, DTKA requires the DTKA-KO with a Node ID (DTKA-KO[Node ID]) to generate a public-private key pair and preserve the secrecy of its private key. The DTKA-KO[Node ID] needs to generate an association message of the form (Node ID, effective-time, public-key), where effective-time specifies the start time after which the public-key is valid. That is, each bundle sent by this node is to be authenticated using the node’s most recently effective public key whose effective time is less than the bundle’s creation time. The DTKA-KO[Node ID] must send the association message, along with a signature on the message using its private key, to the DTKA-KA as depicted in Figure 6. Since DTKA-KA would not have seen the association of the public-key to that key owner previously, it cannot trust that the message indeed originated from DTKA-KO[Node ID]. Therefore, for registration purposes, this initial message from the DTKA-KO[Node ID] to the DTKA-KA MUST be protected by transmitting it over an independently (e.g., physically) authenticated channel. The independently authenticated channel can be realized by physically securing the access to the DTKA-KA server, using a physical communication medium, such as a USB dongle, and manually verifying the authenticity of the communication from the DTKA-KO. The manual verification is a one-time process for a given Key Owner. When an application domain has more than one DTKA-KA (KAx), the message from DTKA-KO[Node ID] must be sent to each DTKA-KA (KAx) in a similarly secure manner.

Although the messages to DTKA-KA (KAx) are independently authenticated, the DTKA-KO[Node ID] must sign the association message using its private key. The signature is not intended to cryptographically authenticate the message but only to prove to the DTKA-KA that the DTKA-KO[Node ID] is indeed in possession of the private key. This self-signed message by the DTKA-KO is useful to ensure that the physical courier, which is used to realize the physically authenticated channel, has not tampered the message sent by the DTKA-KO to the DTKA-KA. Additionally, the self-signed message is useful to audit the operations of the DTKA-KA.

```
      +---------------------+                      +---------------+
      |   DTKA-KO[Node ID]  |                      | DTKA-KA (KAx) |
      +--------|------------+                      +-------|-------+
             **|*******************************************|**
             * |                                           | *
             * |{[Node ID, Effective Time, Public Key, s]  | *
             * | such that s = Sign(Private Key, [Node ID, | *
             * |       Effective Time, Public Key,...])}   | *
             * +------------------------------------------>  *
             * |                                           | *
             * | Physically authenticated channel(USB,...) | *
             **|*******************************************|**
               |                                           |
               |                                        +--+
               | TRUE = Verify(Public Key, s, [Node ID, |  |
               |       Effective Time, Public Key,...]) |  |
               |                                        +-->
               |                                           |
               +                                           +
```

Figure 6: Interaction Diagram 1: Node Registration

Each DTKA-KA will insert the received association message into its next bulletin (refer to Figure 3), for multicast as an out-of-band-authentication (OOBAuth) association: when registration is received through a physically authenticated channel. The bulletin will be multicast to all DTKA Entities using the protocol described in Section 4.7.

As an alternative to the use of a physically authenticated channel, the registration association message may be sent by a trusted third-party node whose authenticated public key is already registered and known to all DTKA-KAs, so that the message may be authenticated by verifying the digital signature (formed using the trusted third-party node’s current private key) in the BIB of the bundle containing the message. Each DTKA-KA will insert such association requests in its next bulletin for multicast as an endorsed association by tagging the corresponding Key Information message in the bulletin as "endorse" (refer to Figure 3). The bulletin will be multicast to all DTKA Entities using the protocol described in Section 4.7.

4.4. Key Revocation

Manual decisions trigger the key revocation procedure. Every DTKA-KA in an application domain is assumed to have a human operator who can trigger the revocation process. When a key is to be revoked, the human operator will need to authenticate to the respective DTKA-KA (KAx) server, identify the public-key and Node ID to be revoked, and instruct that DTKA-KA (KAx) revocation software to schedule a revocation message. The revocation software in DTKA-KA (KAx) will multicast a message as shown in Figure 7. The process for sending out the code-blocks by all the DTKA-KAs with this revocation information is described in Section 4.1.

```
    +---------+                                                +---------+
    | DTKA-KA |                                                | DTKA-KA |
    |  (KAx)  |                                                |  (KAy)  |
    +---|-----+                                                +---|-----+
        |                                                          |
        |                                                          |
        | Multicast{m , s  such that                               |
        | s = Sign(PrivateKey[KAx], m) and                         |
        | m = [KAx, Revoke, [Node ID, Effective Time, Pub Key)]}   |
        +--------------------------------------------------------->
        |                                                          |
        |                                                       +--+
        |         TRUE = Verify(Public Key[KAx], s, [Node ID,   |  |
        |               Effective Time, New Public Key,...])    |  |
        |                                                       +-->
        |                                                          |
        +                                                          +
```

Figure 7: Interaction Diagram 1.1: Key Revocation

4.5. Key Roll-over

When a DTKA-KO[Node ID] has been registered by the DTKA-KA using the protocol described in Figure 6, the DTKA-KO[Node ID] can periodically roll-over to a new public-private key pair by following the key roll-over protocol described in Figure 8. The protocol for key roll-over is similar to the one for key registration except that: (a) the protocol can be executed using DTN bundles issued by the KO itself without requiring any independently secured out-of-band communication channels; and, (b) the old (current) public-key is used to authenticate the association of the new public-key with the Node ID for that DTKA KO. The DTKA-KO [Node ID] must send this message to every key agent in its application domain. Upon accepting the roll-over message from the DTKA-KO[Node ID], each key agent will schedule the roll-over instruction for identified Node ID and public-key in its next bulletin as described in Section 4.1. A DTKA-KO can schedule any number of future roll-overs but the number of such roll-over schedules may need to be limited to avoid Denial of Service attacks by registered nodes -- but this topic is beyond the scope of this document.

```
      +---------------------+                          +---------------+
      |   DTKA-KO[Node ID]  |                          | DTKA-KA (KAx) |
      +--------|------------+                          +-------|-------+
               |                                               |
               |{[Node ID, Effective Time, New Public Key, s]  |
               | such that s = Sign(Old Private Key, [Node ID, |
               |        Effective Time, New Public Key,...])}  |
               +----------------------------------------------->
               |                                               |
               |                                               |
               |                                            +--+
               | TRUE = Verify(Old Public Key, s, [Node ID, |  |
               |       Effective Time, New Public Key,...]) |  |
               |                                            +-->
               |                                               |
               +                                               +
```

Figure 8: Interaction Diagram 1.2: Key Rollover

4.6. Key Endorsement

When a DTKA-KO[Node ID] is not registered and does not have access to any out-of-band authentication channel with any DTKA-KA, the DTKA-KO[Node ID] will need to have access to an out-of-band authentication channel for a trusted third-party (TTP), which is registered with the DTKA-KA. Upon receiving the (Node ID, Key, Effective time) information from the DTKA-KO[Node ID] over the out-of-band authentication channel, the trusted third-party needs to relay that information to the DTKA-KA by signing the information under its authenticated public key. This interaction is depicted in Figure 9. Upon accepting the endorse message from the trusted third-party, each key agent will schedule an endorse instruction for identified Node ID and public-key in its next bulletin as described in Section 4.1.

```
      +---------------------+                           +---------------+
      |Trusted third-party  |                           | DTKA-KA (KAx) |
      +--------|------------+                           +-------|-------+
               |                                                |
               |{[Node ID, Public Key, Effective Time, s]       |
               | such that s = Sign(TTP Private Key, [Node ID,  |
               |         Public Key, Effective Time,...])}      |
               +------------------------------------------------>
               |                                                |
               |                                                |
               |                                             +--+
               | TRUE = Verify(TTP Public Key, s, [Node ID,  |  |
               |       Effective Time, Public Key,...])      |  |
               |                                             +-->
               |                                                |
               +                                                +
```

Figure 9: Interaction Diagram 1.3: Key Endorsement

4.7. Key Distribution

Each DTKA-KA collects multiple out-of-band-authentication (OOBAuth), revocation, roll-over and endorse association messages from different parties by following the protocols described in Section 4.3, Section 4.4, Section 4.5 and Figure 9. Then, each DTKA-KA forms and sends multicast communications for the code blocks for its bulletin to all DTKA Key Users as explained in Section 4.1. The DTKA-KUs verify the authenticity of each code block from all the DTKA-KAs before using the code blocks to decode the bulletin, which will contains out-of-band key authentication, key revocation, key roll-over and endorse instructions. The DTKA-KUs perform these instructions in their respective local key database. This interaction between the DTKA-KAs and the DTKA-KUs of an application domain is shown in Figure 10.

```
      +-------------------+                            +--------------+
      |   DTKA-KA (KAx)   |                            |   DTKA-KU    |
      +---------+---------+                            +--------+-----+
                |                                               |
                +----------------------------------------------->
                | Send(KAx, BulletinHash, CodeBlockNumber       |
                |   CodeBlock of Bulletin such that Bulletin =  |
                |   array[Node ID, Effective Time, PubKey])     |
                |                                               |
                |                                           +---+
                |                  Wait for code blocks     |   |
                |                  from key authorities     +--->
                |                                               |
                |                                           +---+
                |           Decode Bulletin using code      |   |
                |           blocks from key authorities     +--->
                +                                               +
```

Figure 10: Interaction Diagram 2: Bulk Key Distribution

4.8. Secure Communications

After receiving out-of-band-authentication (OOBAuth), roll-over or endorse information, every DTKA-KU shall have authenticated public-keys for different Node IDs in its local database. These authenticated public-keys can be used to authenticate messages received from the DTKA-KO[Node ID] and to send confidential messages to the DTKA-KO[Node ID] after the specified effective-time for each Node ID and public-key pair. This interaction is specified in Figure 11.

```
      +---------------------+                    +--------------+
      |  DTKA-KO[Node ID]   |                    |   DTKA-KU    |
      +--------+------------+                    +--------+-----+
               |Secure communications                     |
               |[Node ID, Creation Time, Signature, Data] |
               +------------------------------------------>
               |                                          |
               |Secure communications                     |
               |[Node ID, Creation Time, Encrypted Data]  |
               <------------------------------------------+
               |                                          |
               +                                          +
```

Figure 11: Interaction Diagram 3: Secure communication

4.9. Communication Stack View

DTKA is designed to be a special DTN application that shall perform key management operations using the services of the Bundle Protocol and BPSec. DTKA will use BP, which in turn will use BPSec to authenticate the messages containing the public-keys that are subsequently to be used by BPSec for securing future communications as shown in Figure 12.

```
     +-------------+---------+   +-------------------------------------+
     |Applications |  DTKA   +--->                                     |
     +-------------+---------+   |        Local Keys Database          |
     |Bundle Protocol (BPSec)<---+(Node ID, Public Key, Effective Time)|
     +-----------------------+   |                                     |
     |   Convergence Layer   |   +-------------------------------------+
     +-----------------------+
     |    Transport Layer    |
     +-----------------------+
     |     Network Layer     |
     +------------------------+
     |    Physical Layer     |
     +-----------------------+
```

Figure 12: Block Diagram: Communication Stack View for DTKA

5. IANA Considerations

This document potentially contains IANA considerations depending on the design choices adopted for future work. But, in its present form, there are no immediate IANA considerations.

6. Security Considerations

Security issues and considerations are discussed through out this document.

7. References

7.1. Normative References

[I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-11 (work in progress), May 2018.

[I-D.ietf-dtn-bpsec]
Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-07 (work in progress), July 2018.

[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>.

7.2. Informative References

[I-D.templin-dtnskmreq]
Templin, F. and S. Burleigh, "DTN Security Key Management - Requirements and Design", draft-templin-dtnskmreq-00 (work in progress), February 2015.

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

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

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

Authors’ Addresses

Scott Burleigh
JPL, Calif. Inst. Of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
USA

Email: Scott.Burleigh@jpl.nasa.gov

David Horres
JPL, Calif. Inst. Of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
USA

Email: David.C.Horres@jpl.nasa.gov

Kapali Viswanathan
Boeing Research & Technology
Boeing International Corporation India Private Limited
A Block, 4th Floor, Lake View Building
Bagmane Tech Park, C.V. Raman Nagar
Bangalore, KA 560093
IN

Email: kapaleeswaran.viswanathan@boeing.com

Michael W. Benson
Boeing Research & Technology
The Boeing Company
499 Boeing Boulevard
Huntsville, AL 35824
USA

Email: michael.w.benson@boeing.com

Fred L. Templin
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
USA

Email: fltemplin@acm.org

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

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

January 28, 2018

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

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

An important goal of the DTN architecture is to accommodate a wide range of networking technologies and environments. The protocol used for DTN communications is the Bundle Protocol Version 7 (BPv7) [I-D.ietf-dtn-bpbis], an application-layer protocol that is used to construct a store-and-forward overlay network. BPv7 requires the services of a "convergence-layer adapter" (CLA) to send and receive bundles using the service of some "native" link, network, or Internet protocol. This document describes one such convergence-layer adapter that uses the well-known Transmission Control Protocol (TCP). This convergence layer is referred to as TCP Convergence Layer Version 4 (TCPCLv4). For the remainder of this document, the abbreviation "BP" without the version suffix refers to BPv7. For the remainder of this document, the abbreviation "TCPCL" without the version suffix refers to TCPCLv4.

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

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

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

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

- o The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [RFC5050] and [I-D.ietf-dtn-bpbis]. This includes the concept of bundle fragmentation or bundle encapsulation. The TCPCL transfers bundles as opaque data blocks.

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

1.1. Convergence Layer Services

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

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

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

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

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

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

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

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

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

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

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

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

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

Reception Failure The TCPCL supports positive indication of certain reasons for reception failure, notably when the local node rejects an attempted transfer for some local policy reason or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

2. Requirements Language

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

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions specific to the TCPCL protocol.

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

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

TCPCL Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two bundle nodes. The lifetime of a TCPCL session is bound to the lifetime of an underlying TCP connection. A TCPCL session is terminated when the TCP connection ends, due either to one or both nodes actively terminating the TCP connection or due to network errors causing a failure of the TCP connection. For the remainder of this document, the term "session" without the prefix "TCPCL" refers to a TCPCL session.

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

Transfer: This refers to the procedures and mechanisms for conveyance of an individual bundle from one node to another. Each transfer within TCPCL is identified by a Transfer ID number which is unique only to a single direction within a single Session.

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

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

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

3. General Protocol Description

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

3.1. TCPCL Session Overview

First, one node establishes a TCPCL session to the other by initiating a TCP connection in accordance with [RFC0793]. After setup of the TCP connection is complete, an initial contact header is exchanged in both directions to set parameters of the TCPCL session and exchange a singleton endpoint identifier for each node (not the singleton Endpoint Identifier (EID) of any application running on the node) to denote the bundle-layer identity of each DTN node. This is used to assist in routing and forwarding messages (e.g. to prevent loops).

Once the TCPCL session is established and configured in this way, bundles can be transferred in either direction. Each transfer is performed by an initialization (XFER_INIT) message followed by one or more logical segments of data within an XFER_SEGMENT message. Multiple bundles can be transmitted consecutively on a single TCPCL connection. Segments from different bundles are never interleaved. Bundle interleaving can be accomplished by fragmentation at the BP layer or by establishing multiple TCPCL sessions between the same peers.

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

A TCPCL receiver can interrupt the transmission of a bundle at any point in time by replying with a XFER_REFUSE message, which causes the sender to stop transmission of the associated bundle (if it hasn’t already finished transmission) Note: This enables a cross-layer optimization in that it allows a receiver that detects that it already has received a certain bundle to interrupt transmission as early as possible and thus save transmission capacity for other bundles.

For sessions that are idle, a KEEPALIVE message is sent at a negotiated interval. This is used to convey node live-ness information during otherwise message-less time intervals.

A SHUTDOWN message is used to start the closing of a TCPCL session (see Section 6.1). During shutdown sequencing, in-progress transfers can be completed but no new transfers can be initiated. A SHUTDOWN message can also be used to refuse a session setup by a peer (see Section 4.3). It is an implementation matter to determine whether or not to close a TCPCL session while there are no transfers queued or in-progress.

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

3.2. Example Message Exchange

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

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

No errors or rejections are shown in this example.

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

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

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

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

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating nodes. It is up to the implementation to decide how and when session setup is triggered.

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

4.1. TCP Connection

To establish a TCPCL session, a node MUST first establish a TCP connection with the intended peer node, typically by using the services provided by the operating system. Destination port number 4556 has been assigned by IANA as the Registered Port number for the TCP convergence layer. Other destination port numbers MAY be used per local configuration. Determining a peer’s destination port number (if different from the registered TCPCL port number) is up to the implementation. Any source port number MAY be used for TCPCL sessions. Typically an operating system assigned number in the TCP Ephemeral range (49152-65535) is used.

If the node is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. A node MAY decide to re-attempt to establish the connection. If it does so, it MUST NOT overwhelm its target with repeated connection attempts. Therefore, the node MUST retry the connection setup no earlier than some delay time from the last attempt, and it SHOULD use a (binary) exponential backoff mechanism to increase this delay in case of repeated failures. In case a SHUTDOWN message specifying a reconnection delay is received, that delay is used as the initial delay. The default initial re-attempt delay SHOULD be no shorter than 1 second and SHOULD be configurable since it will be application and network type dependent.

Once a TCP connection is established, each node MUST immediately transmit a contact header over the TCP connection. The format of the contact header is described in Section 4.2.

4.2. Contact Header

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

Upon receipt of the contact header, both nodes perform the validation and negotiation procedures defined in Section 4.3. After receiving the contact header from the other node, either node MAY refuse the session by sending a SHUTDOWN message with an appropriate reason code.

The format for the Contact Header is as follows:

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

Figure 3: Contact Header Format

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

magic: A four-octet field that always contains the octet sequence 0x64 0x74 0x6e 0x21, i.e., the text string "dtn!" in US-ASCII (and UTF-8).

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

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

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

Segment MRU: A 64-bit unsigned integer indicating the largest allowable single-segment data payload size to be received in this session. Any XFER_SEGMENT sent to this peer SHALL have a data payload no longer than the peer's Segment MRU. The two nodes of a single session MAY have different Segment MRUs, and no relation between the two is required.

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

EID Length and EID Data: Together these fields represent a variable-length text string. The EID Length is a 16-bit unsigned integer indicating the number of octets of EID Data to follow. A zero EID Length SHALL be used to indicate the lack of EID rather than a truly empty EID. This case allows a node to avoid exposing EID information on an untrusted network. A non-zero-length EID Data SHALL contain the UTF-8 encoded EID of some singleton endpoint in which the sending node is a member, in the canonical format of <scheme name>:<scheme-specific part>. This EID encoding is consistent with [I-D.ietf-dtn-bpbis].

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

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

Table 1: Contact Header Flags

4.2.1. Header Extension Items

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

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

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

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

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

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

Figure 4: Header Extension Item Format

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

Table 2: Header Extension Item Flags

4.3. Validation and Parameter Negotiation

Upon reception of the contact header, each node follows the following procedures to ensure the validity of the TCPCL session and to negotiate values for the session parameters.

If the magic string is not present or is not valid, the connection MUST be terminated. The intent of the magic string is to provide some protection against an inadvertent TCP connection by a different protocol than the one described in this document. To prevent a flood of repeated connections from a misconfigured application, a node MAY elect to hold an invalid connection open and idle for some time before closing it.

A connecting TCPCL node SHALL send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If a connecting node receives a SHUTDOWN message with reason of "Version Mismatch", that node MAY attempt further TCPCL sessions with the peer using earlier protocol version numbers in decreasing order. Managing multi-TCPCL-session state such as this is an implementation matter.

If a node receives a contact header containing a version that is greater than the current version of the protocol that the node implements, then the node SHALL shutdown the session with a reason code of "Version mismatch". If a node receives a contact header with a version that is lower than the version of the protocol that the node implements, the node MAY either terminate the session (with a reason code of "Version mismatch") or the node MAY adapt its operation to conform to the older version of the protocol. The decision of version fall-back is an implementation matter.

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

Transfer MTU and Segment MTU: The maximum transmit unit (MTU) for whole transfers and individual segments are idententical to the Transfer MRU and Segment MRU, respectively, of the recevied contact header. A transmitting peer can send individual segments with any size smaller than the Segment MTU, depending on local policy, dynamic network conditions, etc. Determining the size of each transmitted segment is an implementation matter.

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of this two contact headers' Keepalive Interval. The Session Keepalive interval is a parameter for the behavior described in Section 5.2.1.

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two contact headers' CAN_TLS flags. A local security policy is then applied to determine of the negotated value of Enable TLS is acceptable. If not, the node SHALL shutdown the session with a reason code of "Contact Failure". Note that this contact failure is different than a "TLS Failure" after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is true and acceptable then TLS negotiation feature (described in Section 4.4) begins immediately following the contact header exchange.

Once this process of parameter negotiation is completed (which includes a possible completed TLS handshake of the connection to use TLS), this protocol defines no additional mechanism to change the parameters of an established session; to effect such a change, the TCPCL session MUST be terminated and a new session established.

4.4. Session Security

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

The use of TLS is negotated using the Contact Header as described in Section 4.3. After negotiating an Enable TLS parameter of true, and before any other TCPCL messages are sent within the session, the session nodes SHALL begin a TLS handshake in accordance with [RFC5246]. The parameters within each TLS negotiation are implementation dependent but any TCPCL node SHOULD follow all recommended best practices of [RFC7525]. By convention, this protocol uses the node which initiated the underlying TCP connection as the "client" role of the TLS handshake request.

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

4.4.1. TLS Handshake Result

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

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

4.4.2. Example TLS Initiation

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

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

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

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

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

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

                       ... secured TCPCL messaging ...

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

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

5. Established Session Operation

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

5.1. Message Type Codes

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

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

Figure 6: Format of the Message Header

The message header fields are as follows:

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

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

Table 3: TCPCL Message Types

5.2. Upkeep and Status Messages

5.2.1. Session Upkeep (KEEPALIVE)

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

As described in Section 4.3, a negotiated parameter of each session is the Session Keepalive interval. If the negotiated Session Keepalive is zero (i.e. one or both contact headers contains a zero Keepalive Interval), then the keepalive feature is disabled. There is no logical minimum value for the keepalive interval, but when used for many sessions on an open, shared network a short interval could lead to excessive traffic. For shared network use, nodes SHOULD choose a keepalive interval no shorter than 30 seconds. There is no logical maximum value for the keepalive interval, but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens-of-minutes. Nodes SHOULD choose a keepalive interval no longer than 10 minutes (600 seconds).

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

The format of a KEEPALIVE message is a one-octet message type code of KEEPALIVE (as described in Table 3) with no additional data. Both sides SHOULD send a KEEPALIVE message whenever the negotiated interval has elapsed with no transmission of any message (KEEPALIVE or other).

If no message (KEEPALIVE or other) has been received in a session after some implementation-defined time duration, then the node MAY terminate the session by transmitting a one-octet SHUTDOWN message (as described in Section 6.1) with reason code "Idle Timeout.

5.2.2. Message Rejection (MSG_REJECT)

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

The format of a MSG_REJECT message follows:

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

Figure 7: Format of MSG_REJECT Messages

The fields of the MSG_REJECT message are:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 4.

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

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

Table 4: MSG_REJECT Reason Codes

5.3. Bundle Transfer

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

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one node to the other. In TCPCL a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU (see Section 4.2). The choice of the length to use for segments is an implementation matter, but each segment MUST be no larger than the receiving node’s maximum receive unit (MRU) (see the field "Segment

MRU" of Section 4.2). The first segment for a bundle MUST set the 'START' flag, and the last one MUST set the 'end' flag in the XFER_SEGMENT message flags.

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

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

5.3.1. Bundle Transfer ID

Each of the bundle transfer messages contains a Transfer ID which is used to correlate messages (from both sides of a transfer) for each bundle. A Transfer ID does not attempt to address uniqueness of the bundle data itself and has no relation to concepts such as bundle fragmentation. Each invocation of TCPCL by the bundle protocol agent, requesting transmission of a bundle (fragmentary or otherwise), results in the initiation of a single TCPCL transfer. Each transfer entails the sending of a XFER_INIT message and some number of XFER_SEGMENT and XFER_ACK messages; all are correlated by the same Transfer ID.

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

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

5.3.2. Transfer Initialization (XFER_INIT)

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

The purpose of the XFER_INIT message is to allow nodes to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving node for the upcoming bundle data. See Section 5.3.5 for details on when refusal based on XFER_INIT content is acceptable.

The Total Bundle Length field within a XFER_INIT message SHALL be treated as authoritative by the receiver. If, for whatever reason, the actual total length of bundle data received differs from the value indicated by the XFER_INIT message, the receiver SHOULD treat the transmitted data as invalid.

The format of the XFER_INIT message is as follows:

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

Figure 8: Format of XFER_INIT Messages

The fields of the XFER_INIT message are:

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

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

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

5.3.3. Data Transmission (XFER_SEGMENT)

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

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

Figure 9: Format of XFER_SEGMENT Messages

The fields of the XFER_SEGMENT message are:

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

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

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

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

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

Table 5: XFER_SEGMENT Flags

The flags portion of the message contains two optional values in the two low-order bits, denoted ’START’ and ’END’ in Table 5. The ’START’ bit MUST be set to one if it precedes the transmission of the first segment of a transfer. The ’END’ bit MUST be set to one when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the ’START’ and ’END’ bits MUST be set to one.

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

5.3.4. Data Acknowledgments (XFER_ACK)

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

The format of an XFER_ACK message follows in Figure 10.

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

Figure 10: Format of XFER_ACK Messages

The fields of the XFER_ACK message are:

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

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

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

A receiving TCPCL node SHALL send an XFER_ACK message in response to each received XFER_SEGMENT message. The flags portion of the XFER_ACK header SHALL be set to match the corresponding DATA_SEGMENT message being acknowledged. The acknowledged length of each XFER_ACK contains the sum of the data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer. The sending node MAY transmit multiple XFER_SEGMENT messages without necessarily waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a channel.

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

5.3.5. Transfer Refusal (XFER_REFUSE)

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

There is no required relation between the Transfer MRU of a TCPCL node (which is supposed to represent a firm limitation of what the node will accept) and sending of a XFER_REFUSE message. A XFER_REFUSE can be used in cases where the agent’s bundle storage is temporarily depleted or somehow constrained. A XFER_REFUSE can also be used after the bundle header or any bundle data is inspected by an agent and determined to be unacceptable.

A receiver MAY send an XFER_REFUSE message as soon as it receives a XFER_INIT message without waiting for the next XFER_SEGMENT message. The sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

The format of the XFER_REFUSE message is as follows:

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

Figure 11: Format of XFER_REFUSE Messages

The fields of the XFER_REFUSE message are:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 6.

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

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

Table 6: XFER_REFUSE Reason Codes

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

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

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

6. Session Termination

This section describes the procedures for ending a TCPCL session.

6.1. Shutdown Message (SHUTDOWN)

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

After transmitting a SHUTDOWN message, a node MAY immediately close the associated TCP connection. Once the SHUTDOWN message is sent, any further received data on the TCP connection SHOULD be ignored. Any delay between request to terminate the TCP connection and actual closing of the connection (a "half-closed" state) MAY be ignored by the TCPCL node.

The format of the SHUTDOWN message is as follows:

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

Figure 12: Format of SHUTDOWN Messages

The fields of the SHUTDOWN message are:

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

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

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

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

Table 7: SHUTDOWN Flags

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

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

Table 8: SHUTDOWN Reason Codes

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

A session shutdown MAY occur immediately after transmission of a contact header (and prior to any further message transmit). This MAY, for example, be used to notify that the node is currently not able or willing to communicate. However, a node MUST always send the contact header to its peer before sending a SHUTDOWN message.

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

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

6.2. Idle Session Shutdown

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

If there is a configured time to close idle links and if no TCPCL messages (other than KEEPALIVE messages) has been received for at least that amount of time, then either node MAY terminate the session by transmitting a SHUTDOWN message indicating the reason code of "Idle timeout" (as described in Table 8).

7. Security Considerations

One security consideration for this protocol relates to the fact that nodes present their endpoint identifier as part of the contact header exchange. It would be possible for a node to fake this value and present the identity of a singleton endpoint in which the node is not a member, essentially masquerading as another DTN node. If this identifier is used outside of a TLS-secured session or without further verification as a means to determine which bundles are transmitted over the session, then the node that has falsified its identity would be able to obtain bundles that it otherwise would not have. Therefore, a node SHALL NOT use the EID value of an unsecured contact header to derive a peer node’s identity unless it can corroborate it via other means. When TCPCL session security is mandated by a TCPCL peer, that peer SHALL transmit initial unsecured contact header values indicated in Table 9 in order. These values avoid unnecessarily leaking session parameters and will be ignored when secure contact header re-exchange occurs.

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

Table 9: Recommended Unsecured Contact Header

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

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

Another consideration for this protocol relates to denial-of-service attacks. A node MAY send a large amount of data over a TCPCL session, requiring the receiving node to handle the data, attempt to stop the flood of data by sending a XFER_REFUSE message, or forcibly terminate the session. This burden could cause denial of service on other, well-behaving sessions. There is also nothing to prevent a malicious node from continually establishing sessions and repeatedly trying to send copious amounts of bundle data. A listening node MAY take countermeasures such as ignoring TCP SYN messages, closing TCP connections as soon as they are established, waiting before sending the contact header, sending a SHUTDOWN message quickly or with a delay, etc.

8. IANA Considerations

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

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

8.1. Port Number

Port number 4556 has been previously assigned as the default port for the TCP convergence layer in [RFC7242]. This assignment is unchanged by protocol version 4. Each TCPCL node identifies its TCPCL protocol version in its initial contact (see Section 8.2), so there is no ambiguity about what protocol is being used.

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

8.2. Protocol Versions

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version

Numbers" and initialize it with the following table. The registration procedure is RFC Required.

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

8.3. Header Extension Types

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

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Header Extension Types" and initialize it with the contents of Table 10. The registration procedure is RFC Required within the lower range 0x0001--0x3fff. Values in the range 0x8000--0xffff are reserved for use on private networks for functions not published to the IANA.

Code	Message Type
0x0000	Reserved
0x0001--0x3fff	Unassigned
0x8000--0xffff	Private/Experimental Use

Table 10: Header Extension Type Codes

8.4. Message Types

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

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" and initialize it with the contents of Table 11. The registration procedure is RFC Required.

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

Table 11: Message Type Codes

8.5. XFER_REFUSE Reason Codes

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

IANA will create, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" and initialize it with the contents of Table 12. The registration procedure is RFC Required.

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

Table 12: XFER_REFUSE Reason Codes

8.6. SHUTDOWN Reason Codes

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

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

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

Table 13: SHUTDOWN Reason Codes

8.7. MSG_REJECT Reason Codes

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

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

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

Table 14: REJECT Reason Codes

9. Acknowledgments

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

10. References

10.1. Normative References

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

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

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

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

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

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

10.2. Informative References

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

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

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

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

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

Appendix A. Significant changes from RFC7242

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

- o Changed contact header content to limit number of negotiated options.
- o Added contact option to negotiate maximum segment size (per each direction).
- o Added contact header extension capability.
- o Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.
- o Expanded Message Header to octet-aligned fields instead of bit-packing.
- o Added a bundle transfer identification number to all bundle-related messages (XFER_INIT, XFER_SEGMENT, XFER_ACK, XFER_REFUSE).
- o Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.
- o Removed all uses of SDNV fields and replaced with fixed-bit-length fields.

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

- o Added contact negotiation failure SHUTDOWN reason code.
- o Added MSG_REJECT message to indicate an unknown or unhandled message was received.
- o Added TLS session security mechanism.
- o Added TLS failure and Resource Exhaustion SHUTDOWN reason code.

Authors' Addresses

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

Email: BSipos@rkf-eng.com

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

Email: demmer@cs.berkeley.edu

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

Email: jo@netlab.tkk.fi

Simon Perreault
Quebec, QC
Canada

Email: simon@per.reau.lt

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: 7 June 2021

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

4 December 2020

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-24

Abstract

This document describes a TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). This version of the TCPCL protocol resolves 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. This version of TCPCL also includes security and extensibility mechanisms.

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 convergence layer (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 entities in a routing sequence.

* Policies or mechanisms for issuing Public Key Infrastructure Using X.509 (PKIX) certificates; provisioning, deploying, or accessing certificates and private keys; deploying or accessing certificate revocation lists (CRLs); or configuring security parameters on an individual entity or across a network.

* Uses of TLS which are not based on PKIX certificate authentication (see Section 8.12.2) or in which authentication of both entities is not possible (see Section 8.12.1).

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.

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

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

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

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

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

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

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

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

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

Transfer Stream: A Transfer stream is a uni-directional user-data path within a TCPCL Session. Transfers sent over a transfer stream are serialized, meaning that one transfer must complete its transmission prior to another transfer being started over the same transfer stream. At the stream layer there is no logical relationship between transfers in that stream; it's only within the BP agent that transfers are fully decoded as bundles. 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 guaranteed to be unique only to a single direction within a single Session.

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

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

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

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

The relationship between connections, sessions, and streams is shown in Figure 3.

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

Figure 2: The relationships between TCPCL entities

```
+---------------------------+              +---------------------------+
|    "Own" TCPCL Session    |              |  "Other" TCPCL Session    |
|                           |              |                           |
| +----------------------+  |              |  +----------------------+ |
| |   TCP Connection     |  |              |  |   TCP Connection     | |
| |                      |  |              |  |                      | |
| | +-----------------+  |  |   Messages   |  |  +-----------------+ | |
| | |   Own Inbound   |  +--------------------+  |  Peer Outbound  | | |
| | | Transfer Stream |                          | Transfer Stream | | |
| | |       -----     |<---[Seg]--[Seg]--[Seg]---|       -----     | | |
| | |     RECEIVER    |---[Ack]----[Ack]-------->|      SENDER     | | |
| | +-----------------+                          +-----------------+ | |
| |                                                                  | |
| | +-----------------+                          +-----------------+ | |
| | | Own Outbound    |-------[Seg]---[Seg]----->|  Peer Inbound   | | |
| | | Transfer Stream |<---[Ack]----[Ack]-[Ack]--| Transfer Stream | | |
| | |       -----     |                          |       -----     | | |
| | |      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 entity 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 definition 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 preemptively attempt to establish a TCPCL session with a peer entity. Each session attempt can send a different set of session negotiation parameters as directed by the BP agent.

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

Session State Changed: The TCPCL entity indicates to the BP agent 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. When the session is established, the peer Node ID (along with indication of whether or not it was authenticated) and the negotiated session parameters (see Section 4.7) are also communicated to the BP agent.

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

Terminated: The session has finished normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed: The TCPCL entity indicates to the BP agent 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," so a transfer in either direction can block an immediate start of a new transfer in the session.

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

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

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

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

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

Interrupt Reception: The TCPCL entity 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 entity indicates to the BP agent when a bundle has been fully transferred from a peer entity.

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

3.2. TCPCL Session Overview

First, one entity establishes a TCPCL session to the other by initiating a TCP connection in accordance with [RFC0793]. After setup of the TCP connection is complete, an initial Contact Header is exchanged in both directions to establish a shared TCPCL version and negotiate the use of TLS security (as described in Section 4). Once contact negotiation is complete, TCPCL messaging is available and the session negotiation is used to set parameters of the TCPCL session. One of these parameters is a Node ID that each TCPCL Entity is acting as. 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 negligible compared to TCPCL data sizes.

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

A feature of this protocol is for the receiving entity to send acknowledgment (XFER_ACK) messages as bundle data segments arrive. The rationale behind these acknowledgments is to enable the transmitting entity to determine how much of the bundle has been received, so that 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 entity live-ness information during otherwise message-less time intervals.

A SESS_TERM message is used to initiate the ending of a TCPCL session (see Section 6.1). During termination sequencing, in-progress transfers can be completed but no new transfers can be initiated. A SESS_TERM message can also be used to refuse a session setup by a peer (see Section 4.3). Regardless of the reason, session termination is initiated by one of the entities and 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 normal 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 receiving over a transfer stream.

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

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

Contact negotiation involves exchanging a Contact Header (CH) in 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 negotiation of the Processing of Contact Header "[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]
        +-----------+            +---------+
                                      |
                                  Received 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-->[PCH]
                               +-----------------+
```

Figure 6: Contact Initiation as Passive Entity

```
               +-----------+
               |  Peer CH  |
               | available |
               +-----------+
                     |
                Validate and
                 Negotiate
                     V
                +------------+
                | Negotiated |--Failure-->[TCPCLOSE]
                +------------+
                   |       |
                 No TLS    +----Negotiate---+      [ST]
                   |              TLS       |       ^
                   V                        |    Failure
                 +-----------+              V       |
                 |   TCPCL   |            +----------------+
                 | Messaging |<--Success--| TLS Handshake |
                 | 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 negotiation 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 |                   +---------+
        +-----------+                       |
                                    Received SESS_INIT
                                            |
                                            V
                                          [PSI]
```

Figure 8: Session Initiation [SI] as Active Entity

```
 +-----------+
 |   TCPCL   |                  +---------+
 | Messaging |----Wait for ---->| Waiting |--Timeout-->[ST]
 | Available |    SESS_INIT     +---------+
 +-----------+                       |
                             Received 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 in the Ending state 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. PKIX Environments and CA Policy

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

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

DTN-Ignorant CAs: It is expected that Internet-scale "public" CAs will continue to focus on DNS names as the preferred PKIX identifier. There are large infrastructures already in-place for managing network-level authentication and protocols to manage identity verification in those environments [RFC8555]. The TCPCL allows for this type of environment by authenticating a lower-level identifier for a peer and requiring the entity to trust that the Node ID given by the peer (during session initialization) is valid. This situation is not ideal, as it allows vulnerabilities described in Section 8.9, but still provides some amount of mutual authentication to take place for a TCPCL session.

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

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

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

3.5. Session Keeping Policies

This specification gives requirements about how to initiate, sustain, and terminate a TCPCL session but does not impose any requirements on how sessions need to be managed by a BP agent. It is a network administration matter to determine an appropriate session keeping policy, but guidance given here can be used to steer policy toward performance goals.

Persistent Session: This policy preemptively establishes a single session to known entities in the network and keeps the session active using KEEPALIVEs. Benefits of this policy include reducing the total amount of TCP data needing to be exchanged for a set of transfers (assuming KEEPALIVE size is significantly smaller than transfer size), and allowing the session state to indicate peer connectivity. Drawbacks include wasted network resources when a session is mostly idle or when the network connectivity is inconsistent (which requires re-establishing failed sessions), and potential queueing issues when multiple transfers are requested simultaneously. This policy assumes that there is agreement between pairs of entities as to which of the peers will initiate sessions; if there is no such agreement, there is potential for duplicate sessions to be established between peers.

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

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

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

3.6. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. A receiving entity can set the Segment MRU in its SESS_INIT message to determine the largest acceptable segment size, and a transmitting entity can segment a transfer into any sizes smaller than the receiver's Segment MRU. It is a network administration matter to determine an appropriate segmentation policy for entities 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 entity 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 entity, the actual transfer segmentation only needs to guarantee than any individual segment is no larger than that MRU. In a situation where TCP throughput is dynamic, the transfer segmentation size can also be dynamic in order to control message transmission duration.

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

3.7. Example Message Exchange

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

No errors or rejections are shown in this example.

```
                 Entity A                             Entity B
                 ========                             ========
        +-------------------------+
        |  Open TCP Connection    | ->      +-------------------------+
        +-------------------------+      <- |    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 can be opened proactively and maintained for as long as is possible given the network conditions, while other sessions are 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 NOT retry the connection setup 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 (60 seconds) 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. Once a TCP connection is established, the passive entity SHALL wait for the peer's 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. Entities SHOULD choose a Contact Header reception timeout interval no longer than one minute (60 seconds). Upon reception of a Contact Header, the passive entity SHALL transmit its Contact Header. The ordering of the Contact Header exchange allows the passive entity to avoid allocating resources to a potential TCPCL session until after a valid Contact Header has been received from the active entity. This ordering also allows the passive peer to adapt to alternate TCPCL protocol versions.

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

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

4.2. Contact Header

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

If the entity is configured to enable exchanging messages according to TLS 1.3 [RFC8446] or any successors which are compatible with that TLS ClientHello, the the CAN_TLS flag within its Contact Header SHALL be set to 1. The RECOMMENDED policy is to enable TLS for all sessions, even if security policy does not allow or require authentication. This follows the opportunistic security model of [RFC7435], though an active attacker could interfere with the exchange in such cases (see Section 8.4).

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

The format for the Contact Header is as follows:

```
                     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 has enabled TLS security.
Reserved	others	

Table 1: Contact Header Flags

4.3. Contact Validation and Negotiation

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

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

The first negotiation 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. During contact initiation, the active TCPCL entity SHALL send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If the active entity receives a Contact Header with a lower protocol version than the one sent earlier on the TCP connection, the TCP connection SHALL be closed. If the active entity receives a SESS_TERM message with reason of "Version Mismatch", that entity MAY attempt further TCPCL sessions with the peer using earlier protocol version numbers in decreasing order. Managing multi-TCPCL-session state such as this is an implementation matter.

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

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

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

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two Contact Headers' CAN_TLS flags. A local security policy is then applied to determine of the negotiated value of Enable TLS is acceptable. It can be a reasonable security policy to require or disallow the use of TLS depending upon the desired network flows. The RECOMMENDED policy is to require TLS for all sessions, even if security policy does not allow or require authentication. Because this state is negotiated over an unsecured medium, there is a risk of a TLS Stripping as described in Section 8.4.

If the Enable TLS state is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note that this contact failure reason is different than a failure of 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.

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 TLS is established, there is no mechanism available to downgrade the TCPCL session to non-TLS operation.

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

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

4.4.1. Entity Identification

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

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

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

DNS Name: If CA policy forbids a certificate to contain an arbitrary

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

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

This specification defines a NODE-ID of a certificate as being the subjectAltName entry of type uniformResourceIdentifier whose value is a URI consistent with the requirements of [RFC3986] and the URI schemes of the IANA "Bundle Protocol URI Scheme Type" registry [IANA-BUNDLE]. This is similar to the URI-ID of [RFC6125] but does not require any structure to the scheme-specific-part of the URI. Unless specified otherwise by the definition of the URI scheme being authenticated, URI matching of a NODE-ID SHALL use the URI comparison logic of [RFC3986] and scheme-based normalization of those schemes specified in [I-D.ietf-dtn-bpbis]. 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 NODE-ID.

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

4.4.2. Certificate Profile for TCPCL

All end-entity certificates used by a TCPCL entity SHALL conform to [RFC5280], or any updates or successors to that profile. When an end-entity certificate is supplied, the full certification chain SHOULD be included unless security policy indicates that is unnecessary.

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

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

All end-entity and CA certificates used for TCPCL SHOULD contain both a Subject Key Identifier and an Authority Key Identifier extension in accordance with [RFC5280]. TCPCL entities SHOULD NOT rely on either a Subject Key Identifier and an Authority Key Identifier being present in any received certificate. Including key identifiers simplifies the work of an entity needing to assemble a certification chain.

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

This document defines a PKIX Extended Key Usage key purpose "id-kp-bundleSecurity" in Section 9.9 which MAY be used to restrict a certificate's use. The "id-kp-bundleSecurity" purpose can be combined with other purposes in the same certificate. Although not specifically required by TCPCL, some networks or TLS implementations assume the use of "id-kp-clientAuth" and "id-kp-serverAuth" are needed for, respectively, the client-side and server-side of TLS authentication. For interoperability, a TCPCL end-entity certificate MAY contain an Extended Key Usage with both "id-kp-clientAuth" and "id-kp-serverAuth" values.

The PKIX Key Usage bits which are consistent with TCPCL security are: digitalSignature, keyEncipherment, and keyAgreement. The specific algorithms used during the TLS handshake will determine which of those key uses are exercised.

When allowed by CA policy, a TCPCL end-entity certificate SHOULD contain an Online Certificate Status Protocol (OCSP) URI within an Authority Information Access extension in accordance with Section 4.2.2.1 of [RFC5280].

4.4.3. 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 [RFC8446]. By convention, this protocol uses the entity which initiated the underlying TCP connection (the active peer) as the "client" role of the TLS handshake request.

The TLS handshake, if it occurs, is considered to be part of the contact negotiation before the TCPCL session itself is established. Specifics about sensitive data exposure are discussed in Section 8.

The parameters within each TLS negotiation are implementation dependent but any TCPCL entity 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 DNS name was used to establish the TCP connection, the TLS ClientHello SHOULD include a "server_name" extension in accordance with [RFC6066]. When present, the "server_name" extension SHALL contain a "HostName" value taken from the DNS name (of the passive entity) which was resolved. Note: The "HostName" in the "server_name" extension is the network name for the passive entity, not the Node ID of that entity.

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

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. The supplied end-entity certificate SHALL conform to the profile of Section 4.4.2.

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

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

4.4.4. TLS Authentication

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

1. Validating the certification path from the end-entity certificate up to a trusted root CA.

2. Validating the Extended Key Usage (EKU) and other properties of the end-entity certificate.

3. Authenticating identities from a valid end-entity certificate.

4. Applying security policy to the result of each identity type authentication.

The result of validating a peer identity (see Section 4.4.1) against one or more type of certificate claim is one of the following:

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

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

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

4.4.4.1. Certificate Path and Purpose Validation

For any peer end-entity certificate received during TLS handshake, the entity SHALL perform the certification path validation of [RFC5280] up to one of the entity’s trusted CA certificates. If enabled by local policy, the entity SHALL perform an OCSP check of each certificate providing OCSP authoritiy information in accordance with [RFC6960]. If certificate validation fails or if security policy disallows a certificate for any reason, the entity SHALL fail the TLS handshake with a "bad_certificate" alert. Leaving out part of the certification chain can cause the entity to fail to validate a certificate if the left-out certificates are unknown to the entity (see Section 8.6).

For the end-entity peer certificate received during TLS handshake, the entity SHALL apply security policy to the Key Usage extension (if present) and Extended Key Usage extension (if present) in accordance with Section 4.2.1.12 of [RFC5280] and the profile in Section 4.4.2.

4.4.4.2. Network-Level Authentication

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

* If the active entity resolved a DNS name (of the passive entity) in order to initiate the TCP connection that DNS name SHALL be used as a DNS-ID reference identifier.

* The IP address of the other side of the TCP connection SHALL be used as an IPADDR-ID reference identifier.

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

4.4.4.3. Node ID Authentication

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

4.4.5. Policy Recommendations

A RECOMMENDED security policy is to enable the use of OCSP checking during TLS handshake. A RECOMMENDED security policy is that if an Extended Key Usage is present that it needs to contain "id-kp-bundleSecurity" (of Section 4.4.4.1) to be usable with TCPCL security. A RECOMMENDED security policy is to require a validated Node ID (of Section 4.4.4.3) and to ignore any network-level identifier (of Section 4.4.4.2).

This policy relies on and informs the certificate requirements in Section 4.4.3. This policy assumes that a DTN-aware CA (see Section 3.4) will only issue a certificate for a Node ID when it has verified that the private key holder actually controls the DTN node; this is needed to avoid the threat identified in Section 8.9. This policy requires that a certificate contain a NODE-ID and allows the certificate to also contain network-level identifiers. A tailored policy on a more controlled network could relax the requirement on Node ID validation and allow just network-level identifiers to authenticate a peer.

4.4.6. Example TLS Initiation

A summary of a typical TLS use is shown in the sequence in Figure 17 below. In this example the active peer terminates the session but termination can be initiated from either peer.

```
               Entity A                            Entity B
              active peer                        passive peer

      +-------------------------+
      |  Open TCP Connection    | ->      +-------------------------+
      +-------------------------+     <- |    Accept Connection    |
                                          +-------------------------+
      +-------------------------+
      |     Contact Header      | ->      +-------------------------+
      +-------------------------+     <- |     Contact Header      |
                                          +-------------------------+

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

                 DNS-ID and IPADDR-ID authentication occurs.
                     Secured TCPCL messaging can begin.

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

                        NODE-ID authentication 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 and (if TLS is negotiated to be used) the TLS handshake, all messages transmitted over the session are identified by a one-octet header with the following structure:

```
 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 6.1.
XFER_SEGMENT	0x01	Indicates the transmission of a segment of bundle data, as described in Section 5.2.2.
XFER_ACK	0x02	Acknowledges reception of a data segment, as described in Section 5.2.3.
XFER_REFUSE	0x03	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.2.4.
KEEPALIVE	0x04	Used to keep 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 minimum interval, in seconds, to negotiate as the Session Keepalive using the method of Section 4.7.

Segment MRU: A 64-bit unsigned integer indicating the largest allowable single-segment data payload size to be received in this session. Any XFER_SEGMENT sent to this peer SHALL have a data payload no longer than the peer’s Segment MRU. The two entities of a single session MAY have different Segment MRUs, and no 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 the IANA "Bundle Protocol URI Scheme Type" registry [IANA-BUNDLE]. The Node ID itself can be authenticated as described in Section 4.4.4.

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 multiplicity of these Session Extension Items is significant, as defined in the associated type specification(s). If the content of the Session Extension Items data disagrees with the Session Extension Length (e.g., the last Item claims to use more octets than are present in the Session Extension Length), the reception of the SESS_INIT is considered to have failed.

If an entity receives a peer Node ID which is not authenticated (by the procedure of Section 4.4.4.3) that Node ID SHOULD NOT be used by a BP agent for any discovery or routing functions. Trusting an unauthenticated Node ID can lead to the threat described in Section 8.9.

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

4.7. Session Parameter Negotiation

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

Transfer MTU and Segment MTU: The maximum transmit unit (MTU) for whole transfers and individual segments are identical to the Transfer MRU and Segment MRU, respectively, of the received SESS_INIT message. A transmitting peer can send individual segments with any size smaller than the Segment MTU, depending on local policy, dynamic network conditions, etc. Determining the size of each transmitted segment is an implementation matter. If either the Transfer MRU or Segment MRU is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure".

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of the two Keepalive Interval values from the two SESS_INIT messages. The Session Keepalive interval is a parameter for the behavior described in Section 5.1.1. If the Session Keepalive interval is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note: a negotiated Session Keepalive of zero indicates that KEEPALIVEs are disabled.

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

4.8. Session Extension Items

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

The fields of the Session Extension Item are:

Item 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 terminate 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 create 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 (within the minimum imposed by the positive-value encoding), but when used for many sessions on an open, shared network a short interval could lead to excessive traffic. For shared network use, entities SHOULD choose a keepalive interval no shorter than 30 seconds. There is no logical maximum value for the keepalive interval (within the maximum imposed by the fixed-size encoding), but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens-of-minutes. Entities SHOULD choose a keepalive interval no longer than 10 minutes (600 seconds).

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 can experience noticeable latency.

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

If no message (KEEPALIVE or other) has been received in a session after some implementation-defined time duration, then the entity SHALL terminate the session by transmitting a SESS_TERM message (as described in Section 6.1) with 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)

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

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

The format of a MSG_REJECT message is 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 entity.
Message Unsupported	0x02	A message was received but the TCPCL entity cannot comply with the message contents.
Message Unexpected	0x03	A message was received while the session is in a state in which the message is not expected.

Table 4: MSG_REJECT Reason Codes

5.2. Bundle Transfer

All of the messages 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 entity 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 NOT be larger than the receiving entity'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. The sending entity originates a transfer ID and the receiving entity uses that same Transfer ID in acknowledgements.

Transfer IDs from each entity SHALL be unique within a single TCPCL session. Upon exhaustion of the entire 64-bit Transfer ID space, the sending entity SHALL terminate the session with SESS_TERM reason code "Resource Exhaustion". For bidirectional bundle transfers, a TCPCL entity SHOULD NOT rely on any relation between Transfer IDs originating from each side of the TCPCL session.

Although there is not a strict requirement for Transfer ID initial values or ordering (see Section 8.13), in the absence of any other mechanism for generating Transfer IDs an entity SHALL use the following algorithm: The initial Transfer ID from each entity is zero and subsequent Transfer ID values are incremented from the prior Transfer ID value by one.

5.2.2. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message 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 associated single transfer. The order and multiplicity of these transfer extension items is significant, as defined in the associated type specification(s). If the content of the Transfer Extension Items data disagrees with the Transfer Extension Length (e.g., the last Item claims to use more octets than are present in the Transfer Extension Length), the reception of the XFER_SEGMENT is considered to have failed.

Data length: A 64-bit unsigned integer indicating the number of octets in 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 flag values in the two low-order bits, denoted 'START' and 'END' in Table 5. The 'START' flag SHALL be set to 1 when transmitting the first segment of a transfer. The 'END' flag SHALL be set to 1 when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the 'START' and 'END' flags SHALL be set to 1.

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

5.2.3. Data Acknowledgments (XFER_ACK)

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

The format of 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 entity SHALL send an XFER_ACK message in response to each received XFER_SEGMENT message after the segment has been fully processed. The flags portion of the XFER_ACK header SHALL be set to match the corresponding XFER_SEGMENT message being acknowledged (including flags not decodable to the entity). The acknowledged length of each XFER_ACK contains the sum of the data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer. The sending entity SHOULD transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream.

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

5.2.4. Transfer Refusal (XFER_REFUSE)

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

There is no required relation between the Transfer MRU of a TCPCL entity (which is supposed to represent a firm limitation of what the entity 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 transfer receiver MAY send an XFER_REFUSE message as soon as it receives any XFER_SEGMENT message. The transfer sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

The TCPCL itself does not have any required behavior to respond to an XFER_REFUSE based on its Reason Code; the refusal is passed up as an indication to the BP agent that the transfer has been refused. If a transfer refusal has a Reason Code which is not decodable to the BP agent, the agent SHOULD treat the refusal as having an Unknown reason.

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.
Completed	0x01	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
No Resources	0x02	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	0x03	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.
Not Acceptable	0x04	Some issue with the bundle data or the transfer extension data was encountered. The sender SHOULD NOT retry the same bundle with the same extensions.
Extension Failure	0x05	A failure processing the Transfer Extension Items has occurred.
Session Terminating	0x06	The receiving entity is in the process of terminating the session. The sender MAY retry the same bundle at a later time in a different session.

Table 6: XFER_REFUSE Reason Codes

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

The bundle transfer refusal MAY be sent before an entire data segment is received. If a sender receives a XFER_REFUSE message, the sender MUST complete the transmission of any partially sent XFER_SEGMENT message. There is no way to interrupt an individual TCPCL message partway through sending it. The sender MUST NOT 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 can cross on the wire; if this happens, subsequent segments of the bundle SHALL also be refused with a XFER_REFUSE message.

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

5.2.5. Transfer Extension Items

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

The fields of the Transfer Extension Item are:

Item 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 entity receives a Transfer Extension Item with an unknown Item Type and the CRITICAL flag is 1, the entity 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 creates 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 entity for the upcoming bundle data.

Multiple Transfer Length extension items SHALL NOT occur within the same transfer. The lack of a Transfer Length extension item in any transfer SHALL NOT imply anything 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 and send an XFER_REFUSE with a Reason Code of "Not Acceptable".

6. Session Termination

This section describes the procedures for terminating a TCPCL session. The purpose of terminating a session is to allow transfers to complete before the TCP connection is closed but not allow any new transfers to start. A session state change is necessary for this to happen because transfers can be in-progress in either direction (transfer stream) within a session. Waiting for a transfer to complete in one direction does not control or influence the possibility of a transfer in the other direction. Either peer of a session can terminate an established session at any time.

6.1. Session Termination Message (SESS_TERM)

To cleanly terminate a session, a SESS_TERM message SHALL be transmitted by either entity at any point following complete transmission of any other message. When sent to initiate a termination, the REPLY flag of a SESS_TERM message SHALL be 0. Upon receiving a SESS_TERM message after not sending a SESS_TERM message in the same session, an entity SHALL send an acknowledging SESS_TERM message. When sent to acknowledge a termination, a SESS_TERM message SHALL have identical data content from the message being acknowledged except for the REPLY flag, which is set to 1 to indicate acknowledgement.

Once a SESS_TERM message is sent the state of that TCPCL session changes to Ending. While the session is in the Ending state, an entity MAY finish an in-progress transfer in either direction. While the session is in the Ending state, an entity SHALL NOT begin any new outgoing transfer for the remainder of the session. While the session is in the Ending state, an entity SHALL NOT accept any new incoming transfer for the remainder of the session. If a new incoming transfer is attempted while in the Ending state, the receiving entity SHALL send an XFER_REFUSE with a Reason Code of "Session Terminating".

There are circumstances where an entity has an urgent need to close a TCP connection associated with a TCPCL session, without waiting for transfers to complete but also in a way which doesn't force timeouts to occur; for example, due to impending shutdown of the underlying data link layer. Instead of following a clean termination sequence, after transmitting a SESS_TERM message an entity MAY perform an unclean termination by immediately closing the associated TCP connection. When performing an unclean termination, an entity SHOULD acknowledge all received XFER_SEGMENTs with an XFER_ACK before closing the TCP connection. Not acknowledging received segments can result in unnecessary bundle or bundle fragment retransmission. 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 entity. If the underlying TCP connection is closed during a transmission (in either transfer stream), the transfer SHALL be indicated to the BP agent as failed (see the transmission failure and reception failure indications of Section 3.1).

The TCPCL itself does not have any required behavior to respond to an SESS_TERM based on its Reason Code; the termination is passed up as an indication to the BP agent that the session state has changed. If a termination has a Reason Code which is not decodable to the BP agent, the agent SHOULD treat the termination as having an Unknown reason.

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 terminated due to idleness.
Version mismatch	0x02	The entity cannot conform to the specified TCPCL protocol version.
Busy	0x03	The entity is too busy to handle the current session.
Contact Failure	0x04	The entity cannot interpret or negotiate a Contact Header or SESS_INIT option.
Resource Exhaustion	0x05	The entity has run into some resource limit and cannot continue the session.

Table 9: SESS_TERM Reason Codes

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

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

If a session is to be terminated before a protocol message has completed being sent, then the entity MUST NOT transmit the SESS_TERM message but still SHALL close the TCP connection. Each TCPCL message is contiguous in the octet stream and has no ability to be cut short and/or preempted by 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 termination of idle sessions. Determining the length of time to wait before terminating idle sessions, if they are to be terminated at all, is an implementation and configuration matter.

If there is a configured time to terminate idle sessions and if no TCPCL messages (other than KEEPALIVE messages) has been received for at least that amount of time, then either entity 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 intended to use as a proof-of-concept and as a possible source of interoperability testing. This example implementation uses D-Bus as the CL-BP Agent interface, so it only runs on hosts which provide the Python "dbus" library.

8. Security Considerations

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 8.12).

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 8.12).

8.3. Threat: TCPCL Version Downgrade

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

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

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 on-path attacker to set the CAN_TLS flag to 0 on either side of the Contact Header exchange, which will cause the negotiation of Section 4.3 to disable TLS. This leads to the "SSL Stripping" attack described in [RFC7457].

The purpose of the CAN_TLS flag is to allow the use of 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.

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

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

8.5. Threat: Weak TLS Configurations

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 entity to ensure that the negotiated TLS ciphersuite meets transport security requirements.

8.6. Threat: Untrusted End-Entity Certificate

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

8.7. Threat: Certificate Validation Vulnerabilities

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

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

8.8. Threat: Symmetric Key Limits

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.

8.9. Threat: BP Node Impersonation

The certificates exchanged by TLS enable authentication of peer DNS 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 DNS-ID/IPADDR-ID or NODE-ID of the peer (see Section 3.4). Having a CA-validated certificate does not alone guarantee the identity of the network host or BP node from which the certificate is provided; additional validation procedures in Section 4.4.3 bind the DNS-ID/IPADDR-ID or NODE-ID based on the contents of the certificate.

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

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. In circumstances where certificates can only be issued to DNS names, Node ID validation is not possible but it could be reasonable to assume that a trusted host is not going to present an invalid Node ID. Determining when a DNS-ID/IPADDR-ID authentication can be trusted to validate a Node ID is also a policy matter outside of the scope of this document.

One mitigation to arbitrary entities with valid PKIX certificates impersonating arbitrary Node IDs is the use of the PKIX Extended Key Usage key purpose "id-kp-bundleSecurity" in Section 9.9. When this Extended Key Usage is present in the certificate, it represents a stronger assertion that the private key holder should in fact be trusted to operate as a DTN Node.

8.10. Threat: Denial of Service

The behaviors described in this section all amount to a potential denial-of-service to a TCPCL entity. The denial-of-service could be limited to an individual TCPCL session, could affect other well-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 actual bundle data segments. Similarly, an entity can present a very small Transfer MRU which will cause resources to be wasted on establishment and upkeep of a TCPCL session over which a bundle could never be transferred. The victim entity can terminate the session during 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.

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

8.11. Mandatory-to-Implement TLS

Following IETF best current practice, TLS is mandatory to implement for all TCPCL implementations but TLS is optional to use for a given TCPCL session. The recommended configuration of Section 4.2 is to always enable TLS, but entites are permitted to disable TLS based on local configration. The configuration to enable or disable TLS for an entity or a session is outside of the scope of this document. The configuration to disable TLS is different from the threat of TLS stripping described in Section 8.4.

8.12. Alternate Uses of TLS

This specification makes use of PKIX 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. The following subsections give examples of alternate TLS uses.

8.12.1. TLS Without Authentication

In environments where PKI is available but there are restrictions on the issuance of certificates (including the contents of certificates), it may be possible to make use of TLS in a way 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 successfully authenticate some claim of both peer entities of a TCPCL session is outside of the scope of this document but does have similar properties to the opportunistic security model of [RFC7435].

8.12.2. Non-Certificate TLS Use

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

8.13. Predictability of Transfer IDs

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

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

Table 10

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	

Table 11

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 12. 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 12: 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 13. 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 13: 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 14. 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 receiving 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 14: 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 15. 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-existing reasons. Each refusal reason needs to be usable by the receiving 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	Completed
0x02	No Resources
0x03	Retransmit
0x04	Not Acceptable
0x05	Extension Failure
0x06	Session Terminating
0x07--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 15: 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 16. 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-existing reasons. Each termination reason needs to be usable by the receiving 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 16: 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 17. 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-existing reasons. Each rejection reason needs to be usable by the receiving 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 17: MSG_REJECT Reason Codes

9.9. Object Identifier for PKIX Extended Key Usage

IANA has created, under the "Structure of Management Information (SMI) Numbers" registry [IANA-SMI], a sub-registry titled "SMI Security for PKIX Extended Key Purpose". The extended key purpose table is updated to include a purpose "id-kp-bundleSecurity" for identifying DTN endpoints as in the following table.

Decimal	Description	References
KP-TBD	id-kp-bundleSecurity	This specification.

Table 18

This also corresponds with the following SMI for that key purpose:

```
<CODE BEGINS>
id-kp-bundleSecurity OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) kp(3) KP-TBD }
<CODE ENDS>
```

10. Acknowledgments

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

11. References

11.1. Normative References

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

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

[IANA-SMI] IANA, "Structure of Management Information (SMI) Numbers", <https://www.iana.org/assignments/smi-numbers/>.

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

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

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

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

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

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

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

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

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

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

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

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

[I-D.ietf-dtn-bpbis] Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", Work in Progress, Internet-Draft, draft-ietf-dtn-bpbis-29, 17 November 2020, <https://tools.ietf.org/html/draft-ietf-dtn-bpbis-29>.

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>.

[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>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

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

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

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

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

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

[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>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/info/rfc8555>.

[I-D.ietf-dtn-bpsec] Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", Work in Progress, Internet-Draft, draft-ietf-dtn-bpsec-25, 1 December 2020, <https://tools.ietf.org/html/draft-ietf-dtn-bpsec-25>.

[I-D.ietf-dtn-bibect] Burleigh, S., "Bundle-in-Bundle Encapsulation", Work in Progress, Internet-Draft, draft-ietf-dtn-bibect-03, 18 February 2020, <https://tools.ietf.org/html/draft-ietf-dtn-bibect-03>.

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

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 (network byte order) 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 connection security mechanism.

* Added "Not Acceptable", "Extension Failure", and "Session Terminating" XFER_REFUSE reason codes.

* 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
FI-02015 Aalto
Finland

Email: ott@in.tum.de

Simon Perreault
Quebec QC
Canada

Email: simon@per.reau.lt