Internet Research Task Force (IRTF)
Request for Comments: 7242
Category: Experimental
ISSN: 2070-1721

M. Demmer
UC Berkeley
J. Ott
Aalto University
S. Perreault

June 2014

Delay-Tolerant Networking TCP Convergence-Layer Protocol

Abstract

This document describes the protocol for the TCP-based convergence layer for Delay-Tolerant Networking (DTN). It is the product of the IRTF's DTN Research Group (DTNRG).

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Delay-Tolerant Networking Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7242.

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 (BP) [RFC5050], an application-layer protocol that is used to construct a store-and-

forward overlay network. As described in the Bundle Protocol specification [RFC5050], it 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 TCPCL.

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

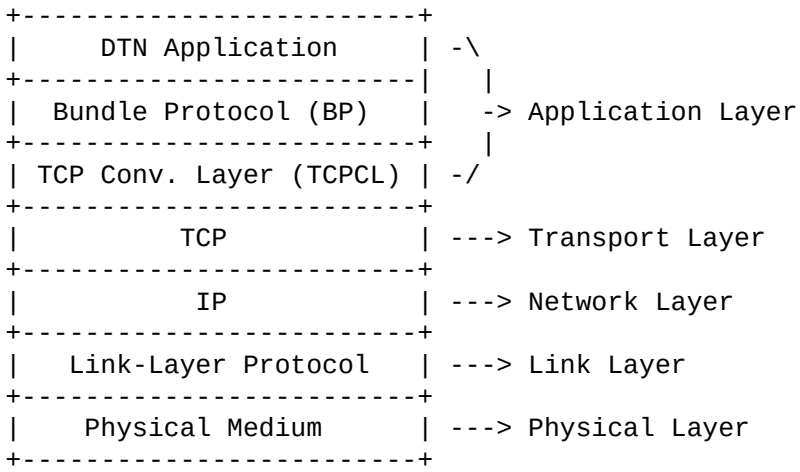

Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol in the Internet Protocol Stack

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

- The format of protocol data units of the Bundle Protocol, as those are defined elsewhere [RFC5050].
- Mechanisms for locating or identifying other bundle nodes within an internet.

Note that this document describes version 3 of the protocol. Versions 0, 1, and 2 were never specified in an Internet-Draft, RFC, or any other public document. These prior versions of the protocol were, however, implemented in the DTN reference implementation [DTNIMPL] in prior releases; hence, the current version number reflects the existence of those prior versions.

This is an experimental protocol produced within the IRTF's Delay-Tolerant Networking Research Group (DTNRG). It represents the consensus of all active contributors to this group. If this protocol is used on the Internet, IETF standard protocols for security and congestion control should be used.

2. Definitions

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].

The terms defined in Section 3.1 of [RFC5050] are used extensively in this document.

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions that are interpreted to be specific to the operation of the TCPCL protocol, as described below.

TCP Connection -- A TCP connection refers to a transport connection using TCP as the transport protocol.

TCPCL Connection -- A TCPCL connection (as opposed to a TCP connection) is a TCPCL communication relationship between two bundle nodes. The lifetime of a TCPCL connection is bound to the lifetime of an underlying TCP connection. Therefore, a TCPCL connection is initiated when a bundle node initiates a TCP connection to be established for the purposes of bundle communication. A TCPCL connection 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 "connection" without the prefix "TCPCL" shall refer to a TCPCL connection.

Connection parameters -- The connection parameters are a set of values used to affect the operation of the TCPCL for a given connection. 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 4.2.

Transmission -- Transmission refers to the procedures and mechanisms (described below) for conveyance of a bundle from one node to another.

3. General Protocol Description

The service of this protocol is the transmission of DTN bundles over 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.

First, one node establishes a TCPCL connection to the other by initiating a TCP connection. After setup of the TCP connection is complete, an initial contact header is exchanged in both directions to set parameters of the TCPCL connection 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 connection is established and configured in this way, bundles can be transmitted in either direction. Each bundle is transmitted in one or more logical segments of formatted bundle data. Each logical data segment consists of a DATA_SEGMENT message header, a Self-Delimiting Numeric Value (SDNV) as defined in [RFC5050] (see also [RFC6256]) containing the length of the segment, and finally the byte range of the bundle data. The choice of the length to use for segments is an implementation matter. The first segment for a bundle must set the 'start' flag, and the last one must set the 'end' flag in the DATA_SEGMENT message header.

If multiple bundles are transmitted on a single TCPCL connection, they MUST be transmitted consecutively. Interleaving data segments from different bundles is not allowed. Bundle interleaving can be accomplished by fragmentation at the BP layer.

An optional feature of the protocol is for the receiving node to send acknowledgments as bundle data segments arrive (ACK_SEGMENT). 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 connection is interrupted, it can perform reactive fragmentation to avoid re-sending the already transmitted part of the bundle.

When acknowledgments are enabled, then for each data segment that is received, the receiving node sends an ACK_SEGMENT code followed by an SDNV containing the cumulative length of the bundle that has been received. The sending node may transmit multiple DATA_SEGMENT messages without necessarily waiting for the corresponding ACK_SEGMENT responses. This enables pipelining of messages on a channel. In addition, there is no explicit flow control on the TCPCL layer.

Another optional feature is that a receiver may interrupt the transmission of a bundle at any point in time by replying with a REFUSE_BUNDLE message, which causes the sender to stop transmission of the current bundle, after completing transmission of a partially sent data segment. 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 connections that are idle, a KEEPALIVE message may optionally be sent at a negotiated interval. This is used to convey liveness information.

Finally, before connections close, a SHUTDOWN message is sent on the channel. After sending a SHUTDOWN message, the sender of this message may send further acknowledgments (ACK_SEGMENT or REFUSE_BUNDLE) but no further data messages (DATA_SEGMENT). A SHUTDOWN message may also be used to refuse a connection setup by a peer.

3.1. Bidirectional Use of TCP Connection

There are specific messages for sending and receiving operations (in addition to connection setup/teardown). TCPCL is symmetric, i.e., both sides can start sending data segments in a connection, 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 visually depicts the protocol exchange for a simple session, showing the connection 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 DATA_SEGMENT messages without necessarily waiting for the corresponding ACK_SEGMENT 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 DATA_SEGMENT

messages for different bundles without necessarily waiting for ACK_SEGMENT messages to be returned for each one. However, interleaving data segments from different bundles is not allowed.

No errors or rejections are shown in this example.

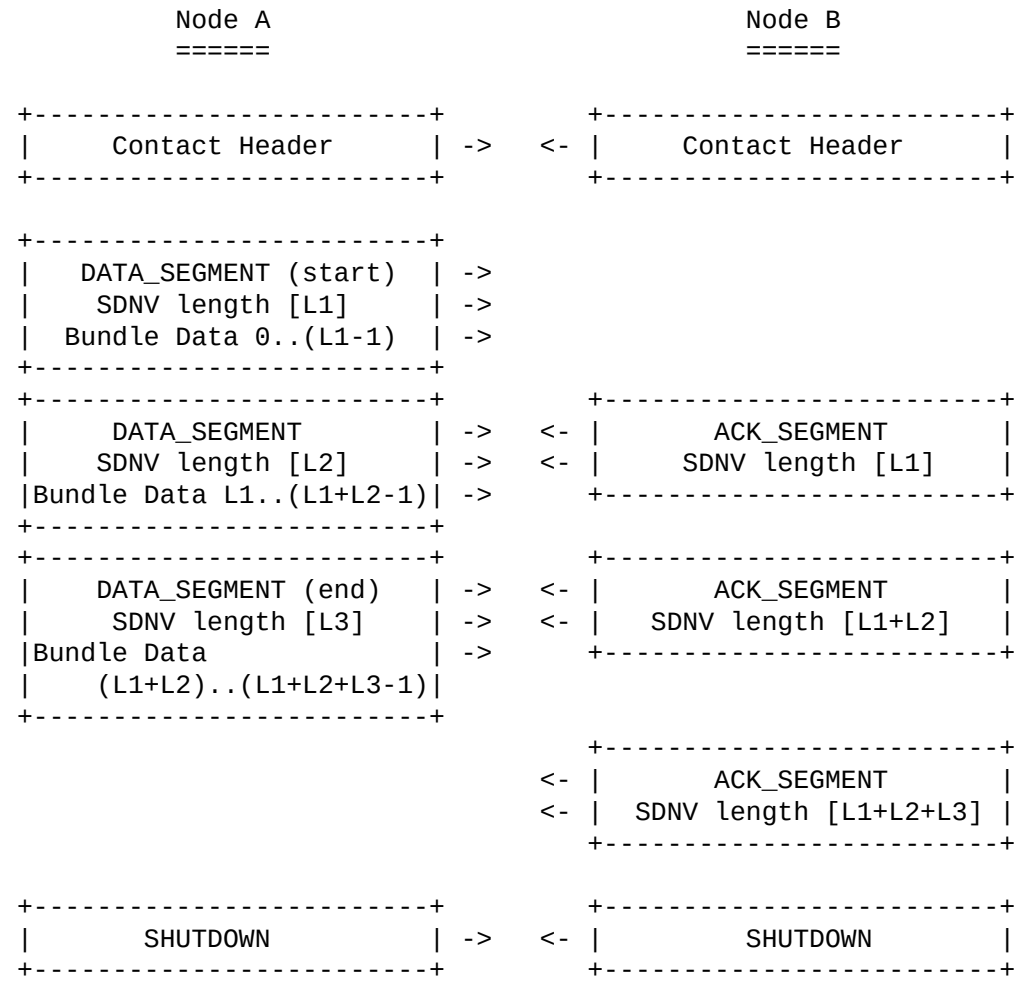

Figure 2: A Simple Visual Example of the Flow of Protocol Messages on a Single TCP Session between Two Nodes (A and B)

4. Connection Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL connection must first be established between communicating nodes. It is up to the implementation to decide how and when connection setup is triggered. For example, some connections may be opened proactively and maintained for as long as is possible given the network

conditions, while other connections may be opened only when there is a bundle that is queued for transmission and the routing algorithm selects a certain next-hop node.

To establish a TCPCL connection, a node must first establish a TCP connection with the intended peer node, typically by using the services provided by the operating system. Port number 4556 has been assigned by IANA as the well-known port number for the TCP convergence layer. Other port numbers MAY be used per local configuration. Determining a peer's port number (if different from the well-known TCPCL port) is up to the implementation.

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 only after some delay (a 1-second minimum is RECOMMENDED), 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 delay SHOULD be at least 1 second but SHOULD be configurable since it will be application and network type dependent.

The node MAY declare failure after one or more connection attempts and MAY attempt to find an alternate route for bundle data. Such decisions are up to the higher layer (i.e., the BP).

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 4.1.

Upon receipt of the contact header, both nodes perform the validation and negotiation procedures defined in Section 4.2

After receiving the contact header from the other node, either node MAY also refuse the connection by sending a SHUTDOWN message. If connection setup is refused, a reason MUST be included in the SHUTDOWN message.

4.1. 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.

The format for the Contact Header is as follows:

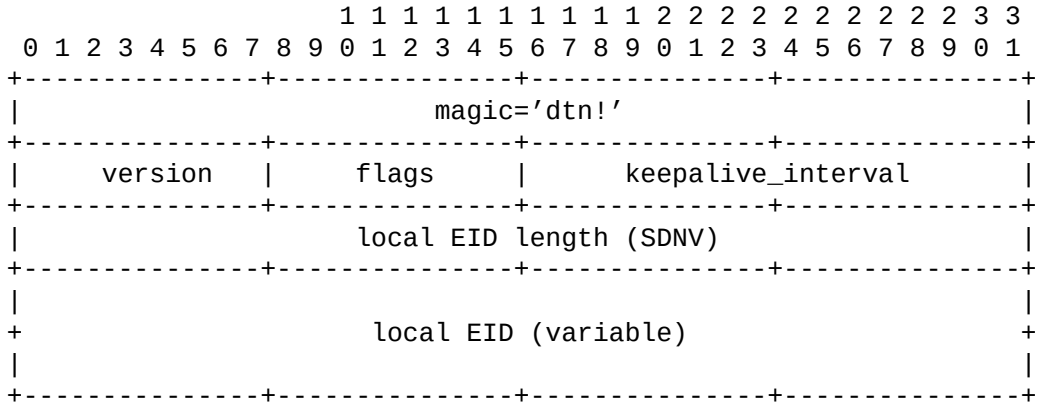

Figure 3: Contact Header Format

The fields of the contact header are:

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

version: A one-byte field value containing the value 3 (current version of the protocol).

flags: A one-byte field containing optional connection flags. The first four bits are unused and MUST be set to zero upon transmission and MUST be ignored upon reception. The last four bits are interpreted as shown in Table 1 below.

keepalive_interval: A two-byte integer field containing the number of seconds between exchanges of KEEPALIVE messages on the connection (see Section 5.6). This value is in network byte order, as are all other multi-byte fields described in this protocol.

local EID length: A variable-length SDNV field containing the length of the endpoint identifier (EID) for some singleton endpoint in which the sending node is a member. A four-byte SDNV is depicted for clarity of the figure.

local EID: A byte string containing the EID of some singleton endpoint in which the sending node is a member, in the canonical format of <scheme name>:<scheme-specific part>. An eight-byte EID is shown for clarity of the figure.

Value	Meaning
00000001	Request acknowledgment of bundle segments.
00000010	Request enabling of reactive fragmentation.
00000100	Indicate support for bundle refusal. This flag MUST NOT be set to '1' unless support for acknowledgments is also indicated.
00001000	Request sending of LENGTH messages.

Table 1: Contact Header Flags

The manner in which values are configured and chosen for the various flags and parameters in the contact header is implementation dependent.

4.2. Validation and Parameter Negotiation

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

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 SHOULD interpret all fields and messages as it would normally. 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 connection due to the version mismatch or may adapt its operation to conform to the older version of the protocol. This decision is an implementation matter.

A node calculates the parameters for a TCPCL connection by negotiating the values from its own preferences (conveyed by the contact header it sent) with the preferences of the peer node (expressed in the contact header that it received). This negotiation MUST proceed in the following manner:

- The parameter for requesting acknowledgment of bundle segments is set to true iff the corresponding flag is set in both contact headers.

- The parameter for enabling reactive fragmentation is set to true iff the corresponding flag is set in both contact headers.

- The bundle refusal capability is set to true if the corresponding flag is set in both contact headers and if segment acknowledgment has been enabled.

- The keepalive_interval parameter is set to the minimum value from both contact headers. If one or both contact headers contains the value zero, then the keepalive feature (described in Section 5.6) is disabled.

- The flag requesting sending of LENGTH messages is handled as described in Section 5.5.

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

5. Established Connection Operation

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

5.1. Message Type Codes

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

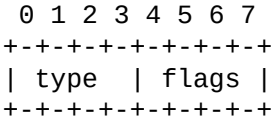

Figure 4: Format of the One-Byte Message Header

type: Indicates the type of the message as per Table 2 below

flags: Optional flags defined based on message type.

The types and values for the message type code are as follows.

Type	Code	Description
	0x0	Reserved.
DATA_SEGMENT	0x1	Indicates the transmission of a segment of bundle data, as described in Section 5.2.
ACK_SEGMENT	0x2	Acknowledges reception of a data segment, as described in Section 5.3
REFUSE_BUNDLE	0x3	Indicates that the transmission of the current bundle shall be stopped, as described in Section 5.4.
KEEPALIVE	0x4	KEEPALIVE message for the connection, as described in Section 5.6.
SHUTDOWN	0x5	Indicates that one of the nodes participating in the connection wishes to cleanly terminate the connection, as described in Section 6.
LENGTH	0x6	Contains the length (in bytes) of the next bundle, as described in Section 5.5.
	0x7-0xf	Unassigned.

Table 2: TCPCL Message Types

5.2. Bundle Data Transmission (DATA_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a DATA_SEGMENT message follows:

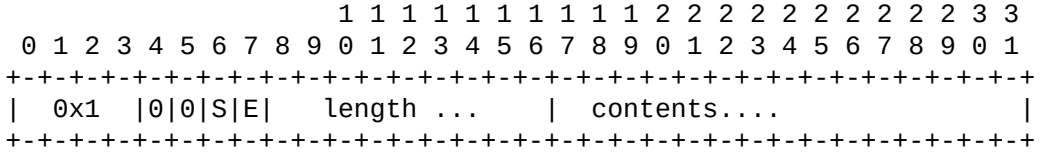

Figure 5: Format of DATA_SEGMENT Messages

The type portion of the message header contains the value 0x1.

The flags portion of the message header byte contains two optional values in the two low-order bits, denoted 'S' and 'E' above. The 'S' bit MUST be set to one if it precedes the transmission of the first segment of a new bundle. The 'E' bit MUST be set to one when transmitting the last segment of a bundle.

Following the message header, the length field is an SDNV containing the number of bytes of bundle data that are transmitted in this segment. Following this length is the actual data contents.

Determining the size of the segment is an implementation matter. In particular, a node may, based on local policy or configuration, only ever transmit bundle data in a single segment, in which case both the 'S' and 'E' bits MUST be set to one.

In the Bundle Protocol specification [RFC5050], a single bundle comprises a primary bundle block, a payload block, and zero or more additional bundle blocks. The relationship between the protocol blocks and the convergence-layer segments is an implementation-specific decision. In particular, a segment MAY contain more than one protocol block; alternatively, a single protocol block (such as the payload) MAY be split into multiple segments.

However, a single segment MUST NOT contain data of more than a single bundle.

Once a transmission of a bundle has commenced, the node MUST only send segments containing sequential portions of that bundle until it sends a segment with the 'E' bit set.

5.3. Bundle Acknowledgments (ACK_SEGMENT)

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, a Bundle Protocol agent needs an additional mechanism to determine whether the receiving agent has successfully received the segment.

To this end, the TCPCL protocol offers an optional feature whereby a receiving node transmits acknowledgments of reception of data segments. This feature is enabled if, and only if, during the exchange of contact headers, both parties set the flag to indicate that segment acknowledgments are enabled (see Section 4.1). If so, then the receiver MUST transmit a bundle acknowledgment message when it successfully receives each data segment.

The format of a bundle acknowledgment is as follows:

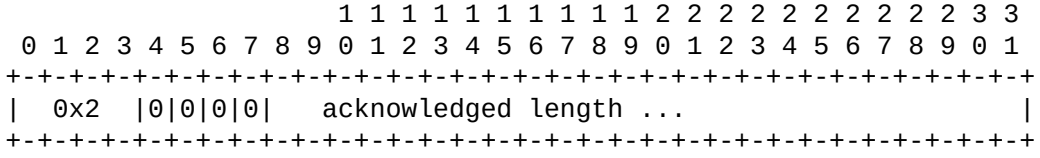

Figure 6: Format of ACK_SEGMENT Messages

To transmit an acknowledgment, a node first transmits a message header with the ACK_SEGMENT type code and all flags set to zero, then transmits an SDNV containing the cumulative length in bytes of the received segment(s) of the current bundle. The length MUST fall on a segment boundary. That is, only full segments can be acknowledged.

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.

5.4. Bundle Refusal (REFUSE_BUNDLE)

As bundles may be large, the TCPCL supports an optional mechanisms by which a receiving node may indicate to the sender that it does not want to receive the corresponding bundle.

To do so, upon receiving a DATA_SEGMENT message, the node MAY transmit a REFUSE_BUNDLE message. As data segments and acknowledgments may cross on the wire, the bundle that is being refused is implicitly identified by the sequence in which acknowledgements and refusals are received.

The format of the REFUSE_BUNDLE message is as follows:

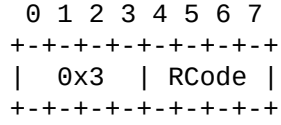

Figure 7: Format of REFUSE_BUNDLE Messages

The RCode field, which stands for "reason code", contains a value indicating why the bundle was refused. The following table contains semantics for some values. Other values may be registered with IANA, as defined in Section 8.

RCode	Semantics
0x0	Reason for refusal is unknown or not specified.
0x1	The receiver now has the complete bundle. The sender may now consider the bundle as completely received.
0x2	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
0x3	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.
0x4-0x7	Unassigned.
0x8-0xf	Reserved for future usage.

Table 3: REFUSE_BUNDLE Reason Codes

The receiver MUST, for each bundle preceding the one to be refused, have either acknowledged all DATA_SEGMENTs or refused the bundle. This allows the sender to identify the bundles accepted and refused by means of a simple FIFO list of segments and acknowledgments.

The bundle refusal MAY be sent before the entire data segment is received. If a sender receives a REFUSE_BUNDLE message, the sender MUST complete the transmission of any partially sent DATA_SEGMENT message (so that the receiver stays in sync). The sender MUST NOT commence transmission of any further segments of the rejected bundle subsequently. Note, however, that this requirement does not ensure that a node will not receive another DATA_SEGMENT for the same bundle after transmitting a REFUSE_BUNDLE message since messages may cross on the wire; if this happens, subsequent segments of the bundle SHOULD also be refused with a REFUSE_BUNDLE message.

Note: If a bundle transmission is aborted in this way, the receiver may not receive a segment with the 'E' flag set to '1' for the aborted bundle. The beginning of the next bundle is identified by the 'S' bit set to '1', indicating the start of a new bundle.

5.5. Bundle Length (LENGTH)

The LENGTH message contains the total length, in bytes, of the next bundle, formatted as an SDNV. Its purpose is to allow nodes to preemptively refuse bundles that would exceed their resources. It is an optimization.

The format of the LENGTH message is as follows:

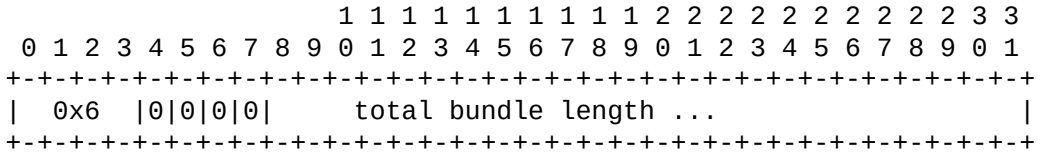

Figure 8: Format of LENGTH Messages

LENGTH messages MUST NOT be sent unless the corresponding flag bit is set in the contact header. If the flag bit is set, LENGTH messages MAY be sent at the sender's discretion. LENGTH messages MUST NOT be sent unless the next DATA_SEGMENT message has the 'S' bit set to "1" (i.e., just before the start of a new bundle).

A receiver MAY send a BUNDLE_REFUSE message as soon as it receives a LENGTH message without waiting for the next DATA_SEGMENT message. The sender MUST be prepared for this and MUST associate the refusal with the right bundle.

5.6. KEEPALIVE Feature (KEEPALIVE)

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

As described in Section 4.1, one of the parameters in the contact header is the keepalive_interval. Both sides populate this field with their requested intervals (in seconds) between KEEPALIVE messages.

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

If no message (KEEPALIVE or other) has been received for at least twice the keepalive_interval, then either party MAY terminate the session by transmitting a one-byte SHUTDOWN message (as described in Table 2) and by closing the TCP connection.

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.

6. Connection Termination

This section describes the procedures for ending a TCPCL connection.

6.1. Shutdown Message (SHUTDOWN)

To cleanly shut down a connection, a SHUTDOWN message MUST be transmitted by either node at any point following complete transmission of any other message. In case acknowledgments have been negotiated, a node SHOULD acknowledge all received data segments first and then shut down the connection.

The format of the SHUTDOWN message is as follows:

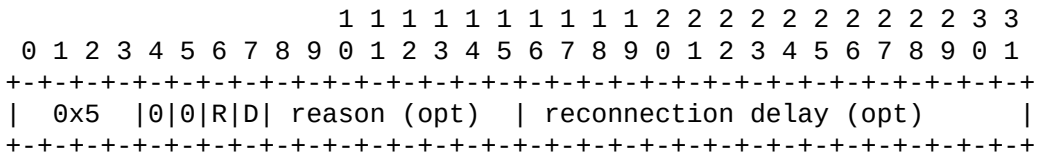

Figure 9: Format of Bundle SHUTDOWN Messages

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

Code	Meaning	Description
0x00	Idle timeout	The connection is being closed due to idleness.
0x01	Version mismatch	The node cannot conform to the specified TCPCL protocol version.
0x02	Busy	The node is too busy to handle the current connection.
0x03-0xff		Unassigned.

Table 4: SHUTDOWN Reason Codes

It is also possible to convey a requested reconnection delay to indicate how long the other node must wait before attempting connection re-establishment. To do so, the node sets the 'D' bit in

the message header flags and then transmits an SDNV specifying the requested delay, in seconds, following the message header (and optionally, the SHUTDOWN reason code). The value 0 SHALL be interpreted as an infinite delay, i.e., that the connecting node MUST NOT re-establish the connection. In contrast, if the node does not wish to request a delay, it SHOULD omit the reconnection delay field (and set the 'D' bit to zero). Note that in the figure above, the reconnection delay SDNV is represented as a two-byte field for convenience.

A connection shutdown MAY occur immediately after TCP connection establishment or reception of a contact header (and prior to any further data exchange). 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 either node terminates a connection prematurely in this manner, it SHOULD send a SHUTDOWN message and MUST indicate a reason code unless the incoming connection did not include the magic string. If a node does not want its peer to reopen the connection immediately, it SHOULD set the 'D' bit in the flags and include a reconnection delay to indicate when the peer is allowed to attempt another connection setup.

If a connection is to be terminated before another protocol message has completed, then the node MUST NOT transmit the SHUTDOWN message but still SHOULD close the TCP connection. In particular, if the connection is to be closed (for whatever reason) while a node is in the process of transmitting a bundle data segment, the receiving node is still expecting segment data and might erroneously interpret the SHUTDOWN message to be part of the data segment.

6.2. Idle Connection Shutdown

The protocol includes a provision for clean shutdown of idle TCP connections. Determining the length of time to wait before closing idle connections, 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 bundle data (other than KEEPALIVE messages) has been received for at least that amount of time, then either node MAY terminate the connection by transmitting a SHUTDOWN message indicating the reason code of 'Idle timeout' (as described in Table 4). After receiving a SHUTDOWN message in response, both sides may close the TCP connection.

7. Security Considerations

One security consideration for this protocol relates to the fact that nodes present their endpoint identifier as part of the connection 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 without further verification as a means to determine which bundles are transmitted over the connection, then the node that has falsified its identity may be able to obtain bundles that it should not have. Therefore, a node SHALL NOT use the endpoint identifier conveyed in the TCPCL connection message to derive a peer node's identity unless it can ascertain it via other means.

These concerns may be mitigated through the use of the Bundle Security Protocol [RFC6257]. In particular, the Bundle Authentication Block defines mechanism for secure exchange of bundles between DTN nodes. Thus, an implementation could delay trusting the presented endpoint identifier until the node can securely validate that its peer is in fact the only member of the given singleton endpoint.

In general, TCPCL does not provide any security services. The mechanisms defined in [RFC6257] are to be used instead.

Nothing in TCPCL prevents the use of the Transport Layer Security (TLS) protocol [RFC5246] to secure a connection.

Another consideration for this protocol relates to denial-of-service attacks. A node may send a large amount of data over a TCP connection, requiring the receiving node to handle the data, attempt to stop the flood of data by sending a REFUSE_BUNDLE message, or forcibly terminate the connection. This burden could cause denial of service on other, well-behaving connections. There is also nothing to prevent a malicious node from continually establishing connections 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].

8.1. Port Number

Port number 4556 has been assigned as the default port for the TCP convergence layer.

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 initialized it with the following:

Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4-255	Unassigned	[RFC7242]

The registration procedure is RFC Required.

8.3. Message Types

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer Message Types" and initialized it with the contents of Table 2. The registration procedure is RFC Required.

8.4. REFUSE_BUNDLE Reason Codes

IANA has created, under the "Bundle Protocol" registry, a sub-registry titled "Bundle Protocol TCP Convergence-Layer REFUSE_BUNDLE Reason Codes" and initialized it with the contents of Table 3. The registration procedure is RFC Required.

8.5. SHUTDOWN Reason Codes

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

9. Acknowledgments

The authors would like to thank the following individuals who have participated in the drafting, review, and discussion of this memo: Alex McMahon, Brenton Walker, Darren Long, Dirk Kutscher, Elwyn Davies, Jean-Philippe Dionne, Joseph Ishac, Keith Scott, Kevin Fall, Lloyd Wood, Marc Blanchet, Peter Lovell, Scott Burleigh, Stephen Farrell, Vint Cerf, and William Ivancic.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

10.2. Informative References

[DTNIMPL] DTNRG, "Delay-Tolerant Networking Reference Implementation", <https://sites.google.com/site/dtnresgroup/home/code>.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.

[RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, May 2011.

Authors' Addresses

Michael J. 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