Delay Tolerant Networking Research Group
Internet-Draft
Intended status: Experimental
Expires: March 1, 2013

M. Demmer
UC Berkeley
J. Ott
Helsinki University of Technology
S. Perreault
Viagenie

August 28, 2012

Delay Tolerant Networking TCP Convergence Layer Protocol
draft-irtf-dtnrg-tcp-clayer-04.txt

Abstract

This document describes the protocol for the TCP-based Convergence Layer for Delay Tolerant Networking (DTN).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on March 1, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Definitions 4
2.1. Definitions Relating to the Bundle Protocol 4
2.2. Definitions specific to the TCPCL Protocol 5
3. General Protocol Description 6
3.1. Example message exchange 7
4. Connection Establishment 8
4.1. Contact Header 9
4.2. Validation and parameter negotiation 11
5. Established Connection Operation 12
5.1. Message Type Codes 12
5.2. Bundle Data Transmission 13
5.3. Bundle Acknowledgments 14
5.4. Bundle Refusal 15
5.5. Bundle Length 16
5.6. Keepalive Messages 16
6. Connection Termination 17
6.1. Shutdown Message 17
6.2. Idle Connection Shutdown 19
7. Requirements notation 19
8. Security Considerations 19
9. IANA Considerations 20
10. References 20
Authors' Addresses 20

1. Introduction

This document describes the TCP-based convergence layer protocol for Delay Tolerant Networking (TCPCL). 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 the Delay-Tolerant Network Architecture [refs.dtnarch] RFC.

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 Bundling Protocol (BP) [refs.bundleproto], an application-layer protocol that is used to construct a store-and-forward overlay network. As described in the bundle protocol specification, 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.

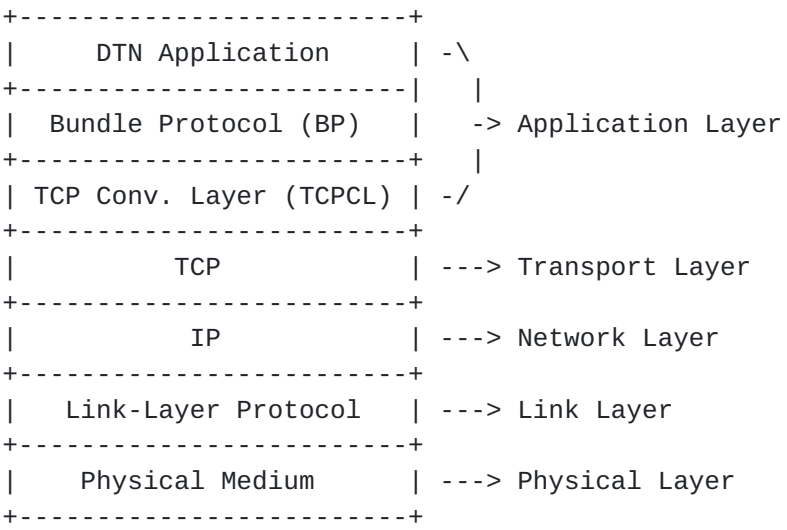


Figure 1: The locations of the bundle protocol and the TCP convergence layer protocol in the Internet protocol stack

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 bundling protocol, as those are defined elsewhere [refs.bundleproto].

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 any Internet Draft, RFC, or any other public document. These prior versions of the protocol were, however, implemented in the DTN reference implementation [refs.dtnimpl], in prior releases, hence the current version number reflects the existence of those prior versions.

2. Definitions

2.1. Definitions Relating to the Bundle Protocol

The following set of definitions are abbreviated versions of those which appear in the Bundle Protocol Specification [refs.bundleproto]. To the extent in which terms appear in both documents, they are intended to have the same meaning.

Bundle -- A bundle is a protocol data unit of the DTN bundle protocol.

Bundle payload -- A bundle payload (or simply "payload") is the application data whose conveyance to the bundle's destination is the purpose for the transmission of a given bundle.

Fragment -- A fragment is a bundle whose payload contains a contiguous subset of bytes from another bundle's payload.

Bundle node -- A bundle node (or simply a "node") is any entity that can send and/or receive bundles. The particular instantiation of this entity is deliberately unconstrained, allowing for implementations in software libraries, long-running processes, or even hardware. One component of the bundle node is the implementation of a convergence layer adapter.

Convergence layer adapter -- A convergence layer adapter (CLA) sends and receives bundles utilizing the services of some 'native' link, network, or internet protocol. This document describes the manner in which a CLA sends and receives bundles when using the TCP protocol for inter-node communication.

Self Describing Numeric Value -- A self describing numeric value (SDNV) is a variable length encoding for integer values, defined in the bundle protocol specification.

2.2. 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 one-to-one with 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 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

This protocol provides bundle conveyance over a TCP connection and specifies the encapsulation of bundles as well as procedures for TCP connection setup and teardown. 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 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, an SDNV 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.

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. Note that in the case of concurrent bidirectional transmission, then ack segments may be interleaved with data segments.

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. 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.

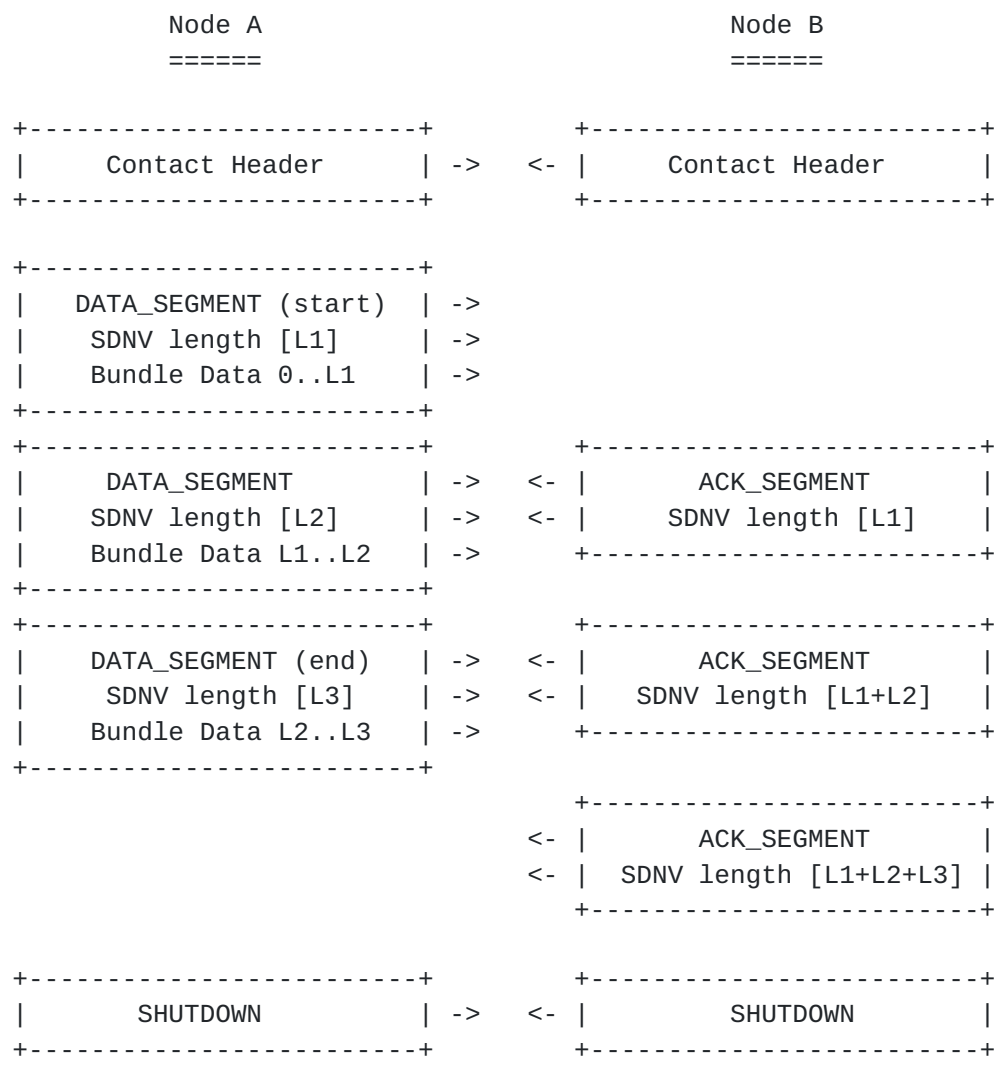


Figure 2: A simple visual example of the flow of protocol messages on a single TCP session between two nodes (A and B)

4. Connection Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL connection must first be established between communicating nodes. The manner in which a bundle node makes the decision to establish such a connection is implementation-dependent. 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, perhaps. 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 and it SHOULD use a (binary) exponential backoff mechanism to increase this delay in case of repeated failures.

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:

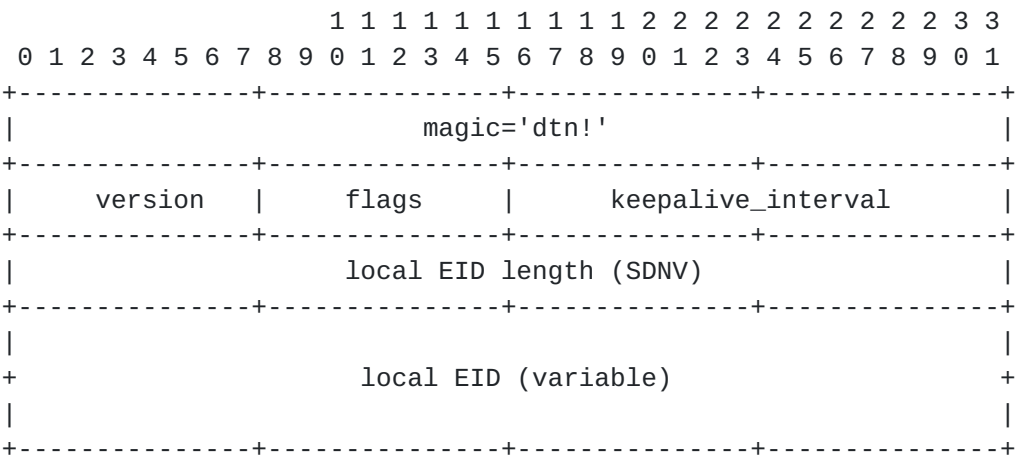


Figure 3: Contact Header Format

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!".

version: A one byte field value containing the 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 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: An octet 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>. A eight byte EID is shown the 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 for ensuring 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 segment acknowledgments enabled parameter is set to true iff the corresponding flag is set in both contact headers.

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

The bundle refusal capability may only be used iff both peers indicate support for it in their contact header.

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.

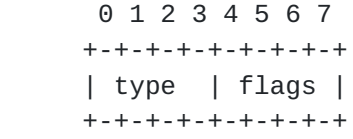
Once this process of parameter negotiation is completed, the protocol defines no additional mechanism to change the parameters of an established connection; to effect such a change, the connection MUST be terminated and a new connection established.

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 octet header with the following structure:



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

flags: Optional flags defined on a per message type basis.

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

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

Table 2: TCPCL Header Types

5.2. Bundle Data Transmission

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

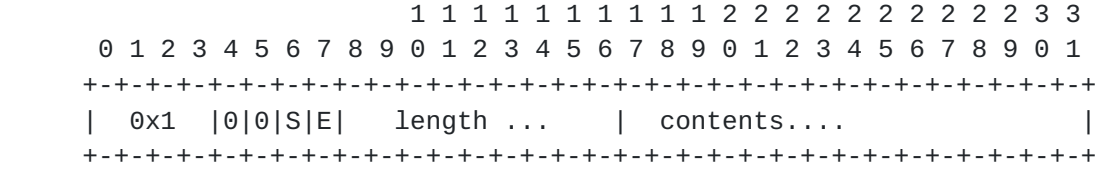


Figure 4: Format of bundle data segment messages

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

The flags portion of the message header octet contains two optional values in the two low-order bits, denoted 'S' and 'E' above. The 'S' bit MUST be set to one iff 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.

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. However, a node MUST be able to receive a bundle that has been transmitted in any segment size.

In the bundle protocol specification, 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.

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.

5.3. Bundle Acknowledgments

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 header when it successfully receives each data segment.

The format of a bundle acknowledgment is as follows:

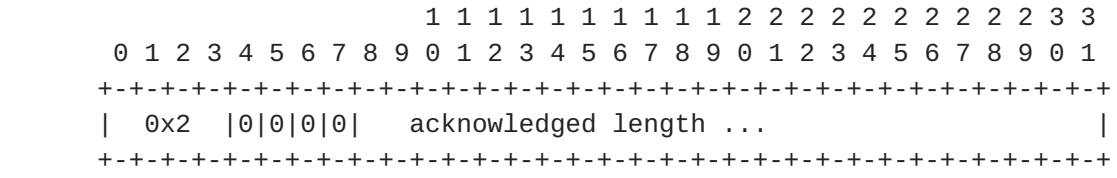


Figure 5: Format of bundle acknowledgement messages

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 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

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 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 be refused with a REFUSE_BUNDLE message, too.

Note: If a bundle transmission if 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

The format of the LENGTH message is as follows:

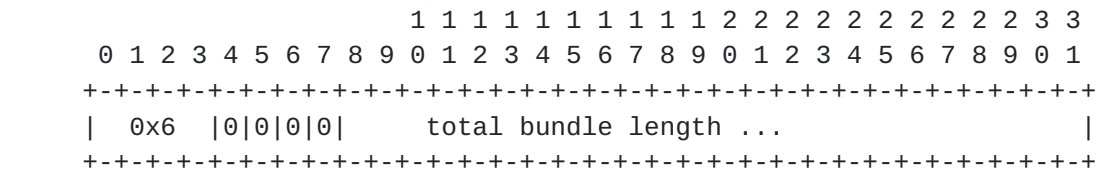


Figure 6: Format of LENGTH messages

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.

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 receiver MUST be prepared for this and MUST associate the refusal with the right bundle.

5.6. Keepalive Messages

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 message type code of SHUTDOWN (as described in Table 2) and 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 RTO) which is dependent on the measured RTT for the TCP connection so that keepalive message may experience noticeable latency.

6. Connection Termination

This section describes the procedures for ending a TCPCL connection.

6.1. Shutdown Message

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, it is advisable to acknowledge all received data segments first and then shut down the connection.

The format of the shutdown message is as follows:

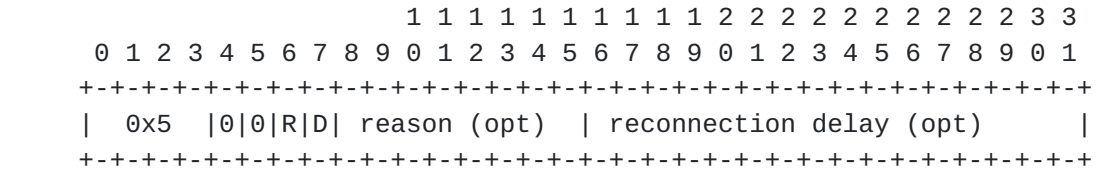


Figure 7: Format of bundle shutdown messages

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	Comment
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.

Table 3: 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, 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 delay field (and set the 'D' bit to zero). Note that in the figure above, a two octet SDNV is shown for convenience of the presentation.

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 capable of or willing to communicate. However, a node MUST always send the contact header to its peer first.

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 re-open 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, 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, then 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 above). After receiving a SHUTDOWN message in response, both sides may close the TCP connection.

7. Requirements notation

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].

8. 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.

These concerns may be mitigated through the use of the Bundle Security Protocols [refs.dtnsecurity]. In particular, the Bundle Authentication Header 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.

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 either 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.

9. IANA Considerations

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

10. References

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

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

[refs.dtnarch] Cerf et al, V., "Delay-Tolerant Network Architecture", RFC 4838, April 2007.

[refs.dtnimpl] DTNRG, "Delay Tolerant Networking Reference Implementation", <http://www.dtnrg.org/Code>.

[refs.dtnsecurity] Symington, S., Farrell, S., and H. Weiss, "Bundle Security Protocol Specification", Internet Draft, work in progress draft-irtf-dtnrg-bundle-security-03.txt, April 2007.

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
Helsinki University of Technology
Department of Communications and Networking
PO Box 3000
TKK 02015
Finland

Email: jo@netlab.tkk.fi

Simon Perreault
Viagenie
246 Aberdeen
Quebec, QC G1R 2E1
Canada

Phone: +1 418 656 9254
Email: simon.perreault@viagenie.ca