

dtnrg
Internet-Draft
Intended status: Informational
Expires: March 8, 2013

K. Scott
The MITRE Corporation
M. Blanchet
Viagenie
September 04, 2012

Licklider Transmission Protocol (LTP), Compressed Bundle Header Encoding
(CBHE), and Bundle Protocol IANA Registries
draft-dtnrg-ltp-cbhe-registries-03

Abstract

The DTNRG research group has defined the experimental Licklider Transmission Protocol (LTP) [RFC5326] and the Compressed Bundle Header Encoding (CBHE) [RFC6260] mechanism for the 'ipn' URI scheme. Finally, RFC5050 [RFC5050] defines values for the Bundle Administrative Record Type. All of these describe fields that are subject to a registry. For the purpose of its research work, the group has created ad-hoc registries. As the specifications are stable and have multiple interoperable implementations, the group would like to hand off the registries to IANA for official custody. This document describes the actions needed to be executed by IANA.

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 8, 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. Licklider Transmission Protocol | 3 |
| 2.1. LTP Cancel Segment Reason Codes | 3 |
| 2.2. LTP Engine ID | 4 |
| 2.3. LTP Client Service ID | 5 |
| 3. Compressed Bundle Header Encoding | 6 |
| 3.1. CBHE Node Numbers | 6 |
| 3.2. CBHE Service Numbers | 7 |
| 4. Bundle Administrative Record Types | 8 |
| 5. Security Considerations | 9 |
| 6. IANA Considerations | 9 |
| 7. Acknowledgements | 9 |
| 8. References | 9 |
| 8.1. Normative References | 9 |
| 8.2. Informative References | 10 |
| Authors' Addresses | 10 |

1. Introduction

The DTNRG research group has defined the Licklider Transmission Protocol (LTP)[RFC5326]. LTP contains certain fields that are subject to a registry. For the purpose of its research work, the group has created ad-hoc registries. As the specifications are stable and have multiple interoperable implementations, the group would like to hand off the registries to IANA for official custody. This document describes the actions needed to be executed by IANA [RFC5226].

The Compressed Bundle Header Encoding (CBHE) [RFC6260] specification defines the concepts of Node Number and Service Number in the 'ipn' URI scheme. In this document we request formation of an IANA registry for the Node Number field in the 'ipn' scheme.

Because of its association with space communication and the Consultative Committee on Space Data Systems [CCSDS], a portion of the CBHE Node Number space and a corresponding portion of the LTP Engine ID space is delegated by this document to the CCSDS Space Assigned Numbers Authority [SANA]. SANA functions similarly to IANA in that it maintains registries of managed values, with a focus on values used by protocols used by CCSDS member agencies.

2. Licklider Transmission Protocol

The Licklider Transmission Protocol has fields requiring registries managed by IANA. This document requests the creation of the three registries in this section and that they be associated with the other registries for the Licklider Transmission protocol.

2.1. LTP Cancel Segment Reason Codes

Section 3.2.4 of [RFC5326] defines the reason codes that may be present in Cancel Segments in the LTP protocol.

The registration policy for this registry is: RFC Required

The initial values(as defined by RFC5326) for the LTP Cancel Segment Reason Codes registry shall be:

LTP Cancel Segment Reason Codes Registry

| Value | Description | Reference |
|-----------|---------------------------------|---------------|
| 0 | Client service canceled session | [RFC5326] |
| 1 | Unreachable client service | [RFC5326] |
| 2 | Retransmission limit exceeded | [RFC5326] |
| 3 | Miscolored data received | [RFC5326] |
| 4 | System error caused termination | [RFC5326] |
| 5 | Retransmission limit exceeded | [RFC5326] |
| 0x06-0xFF | Unassigned | This document |

2.2. LTP Engine ID

The Licklider Transmission Protocol has an LTP Engine ID field (section 2 of [RFC5326]). An IANA registry shall be set up as follows.

The registration policy for this registry is:

- 1 -- (2**14)-1 Expert review required. The designated experts for the review are the chairs of the IRTF DTN Research Group (dtnrg) if the dtnrg is extant, or as determined by the IRSG.
- (2**14) -- (2**21)-1 Allocated to the Space Assigned Numbers Authority ([SANA]) for use by Consultative Committee for Space Data Systems (CCSDS) missions.
- (2**21) -- (2**27)-1 Private or experimental use. No assignment by IANA.
- (2**27) -- (2**42)-1 First-come, First-Served basis for requests for less than or equal to 2**14 values to a single entity or organization. Expert review for requests of more than 2**14 values to a single entity or organization. The designated experts for the review are the chairs of the IRTF DTN Research Group (dtnrg) if the dtnrg is extant, or as determined by the IRSG.

The LTP Engine ID is expressed as a Self-Delimiting Numeric Value (SDNV) in the LTP protocol and no maximum is specified in the protocol definition. SDNVs were first described in relation to the Bundle Protocol in Section 4.1 of [RFC5050] and are also described in the stand-alone document [RFC6256]. The initial values for the LTP Engine Numbers registry shall be:

LTP Engine Numbers Registry

| Value | Description | Reference |
|--------------------|---------------------------|---------------|
| 0 | Reserved | This document |
| 1--(2**14)-1 | Unassigned | This document |
| (2**14)--(2**21)-1 | Allocated to CCSDS (SANA) | This document |
| (2**21)--(2**27)-1 | Private/Experimental Use | This document |
| (2**27)--(2**42)-1 | Unassigned | This document |
| ≥(2**42) | Reserved | This document |

2.3. LTP Client Service ID

The Licklider Transmission Protocol has a client service ID number field (section 3.2.1 of [RFC5326]). An IANA registry shall be set up as follows.

The registration policy for this registry is:

4 -- (2**14)-1 Allocated to the Space Assigned Numbers Authority ([SANA]) for use by Consultative Committee for Space Data Systems (CCSDS) missions.

2**14 -- 32,767 Private or experimental use; no assignment by IANA.

≥ 32,768 Expert review required. The designated experts for the review are the chairs of the IRTF DTN Research Group (dtnrg) if the dtnrg is extant, or as determined by the IRSG.

The LTP Client Service ID is expressed as a Self-Delimiting Numeric Value (SDNV) in the LTP protocol and no maximum value is specified in the protocol definition. The initial values for the LTP Client Service Identifiers registry shall be:

LTP Client Service Identifiers Registry

| Value | Description | Reference |
|-----------------|------------------------------|---------------|
| 0 | Reserved | [RFC5326] |
| 1 | Bundle Protocol | This document |
| 2 | LTP Service Data Aggregation | This document |
| 3 | CCSDS File Delivery Service | This document |
| 4--(2**14)-1 | Allocated to CCSDS (SANA) | This document |
| (2**14)--32,767 | Private / Experimental Use | This document |
| >=32,768 | Unassigned | This document |

3. Compressed Bundle Header Encoding

The CBHE specification defines concepts of 'Node Number' and 'Service Number' that require registries managed by IANA.

3.1. CBHE Node Numbers

The CBHE specification defines a Node Number (node-nbr) field (section 2.1 of [RFC6260]). An IANA registry shall be set up as follows.

The registration policy for this registry is:

- 1 -- (2**14)-1 Allocatable by IANA; expert review required. The designated experts for the review are the chairs of the IRTF DTN Research Group (dtnrg) if the dtnrg is extant, or as determined by the IRSG.
- (2**14) -- (2**21)-1 Allocated to the Space Assigned Numbers Authority ([SANA]) for use by Consultative Committee for Space Data Systems (CCSDS) missions.
- (2**21) -- (2**27)-1 Private or experimental use. No assignment by IANA.
- (2**27) -- (2**42)-1 Allocatable by IANA on a First-come, First-Served basis for requests for less than or equal to 2**14 values to a single entity or organization. Expert review for requests of more than 2**14 values to a single entity or organization. The designated experts for the review are the chairs of the IRTF DTN Research Group (dtnrg) if the dtnrg is extant, or as determined by the IRSG.

$\geq (2^{42})$ Reserved

The CBHE Node Number is expressed as a Self-Delimiting Numeric Value (SDNV) in the CBHE specification. Allowable values for the Node Number range from 1 -- $2^{64}-1$. The initial values for the CBHE Node Number registry shall be:

CBHE Node Number Registry

| Value | Description | Reference |
|------------------------|---------------------------|---------------|
| 0 | Reserved | This document |
| $1--(2^{14})-1$ | Unassigned | This document |
| $(2^{14})--(2^{21})-1$ | Allocated to CCSDS (SANA) | This document |
| $(2^{21})--(2^{27})-1$ | Private/Experimental Use | This document |
| $(2^{27})--(2^{42})-1$ | Unassigned | This document |
| $\geq (2^{42})$ | Reserved | This document |

3.2. CBHE Service Numbers

The Compressed Bundle Header Encoding specification defines a Service Number (service-nbr) field (section 2.1 of [RFC6260]). An IANA registry shall be set up as follows.

The registration policy for this registry is:

0-63 RFC Required

64-127 Allocated to the Space Assigned Numbers Authority ([SANA]) for use by Consultative Committee for Space Data Systems (CCSDS) missions.

128 - $2^{16}-1$ Specification Required

$\geq 2^{16}$ Private / Experimental Use

The CBHE Service Number is expressed as a Self-Delimiting Numeric Value (SDNV) in the CBHE specification. Allowable values for the Node Number range from 1 -- $2^{64}-1$. The initial values for the CBHE Node Number registry shall be:

CBHE Service Number Registry

| Value | Description | Reference |
|------------------|---------------------------------------|---------------|
| 0 | Bundle Protocol Administrative Record | [RFC6260] |
| 1 | CCSDS File Delivery Service | [CFDP] |
| 2-63 | Unassigned | This document |
| 64-127 | Allocated to CCSDS (SANA) | This document |
| 128 - $2^{16}-1$ | Unassigned | This document |
| $\geq 2^{16}$ | Private/Experimental Use | This document |

4. Bundle Administrative Record Types

Section 6.1 of the Bundle Protocol specification[RFC5050] specifies a 4-bit Administrative Record type code. An IANA registry shall be set up as follows to manage these record types. This document requests the addition of an additional registry titled 'Bundle Administrative Record Type' be added to the list of registries associated with the Bundle Protocol.

The registration policy for this registry is:RFC required

The initial values for the Bundle Administrative Record Type registry shall be:

Bundle Protocol Administrative Record Type Registry

| Value | Description | Reference |
|-------|----------------------|---------------|
| 0 | Reserved | This document |
| 1 | Bundle status report | [RFC5050] |
| 2 | Custody signal | [RFC5050] |
| 3-15 | Unassigned | This document |

5. Security Considerations

This document requests the creation of registries managed by IANA. There are no security issues involved. Refer to the Security Considerations section of [RFC5326] for security issues with the LTP protocol.

6. IANA Considerations

IANA is requested to create the registries as described in Sections 2, 3, and 4 of this document.

7. Acknowledgements

The editors would like to thank the following people, in no specific order: Scott Burleigh, Stephen Farrell.

8. References

8.1. Normative References

- [CFDP] Consultative Committee for Space Data Systems (<http://www.ccsds.org>), "CCSDS File Delivery Protocol Version 4 (CCSDS 727.0-B-4)", January 2007.
- [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.
- [RFC5326] Ramadas, M., Burleigh, S., and S. Farrell, "Licklider Transmission Protocol - Specification", RFC 5326, September 2008.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.
- [RFC6260] Burleigh, S., "Compressed Bundle Header Encoding (CBHE)", RFC 6260, May 2011.

8.2. Informative References

- [CCSDS] "The Consultative Committee for Space Data Systems,
<http://www.ccsds.org>".
- [SANA] "The CCSDS SANA Registry page at <http://sanaregistry.org>".

Authors' Addresses

Keith Scott
The MITRE Corporation
7515 Colshire Drive
McLean, VA, California 22102
USA

Phone: +1-703-983-6547
Fax: +1-703-983-7142
Email: kscott@mitre.org

Marc Blanchet
Viagenie
246 Aberdeen
Quebec, Quebec G1R 2E1
Canada

Phone: +1-418-656-9254
Email: marc.blanchet@viagenie.ca

DTNRG
Internet-Draft
Intended status: Experimental
Expires: March 5, 2013

H. Kruse
S. Jero
S. Ostermann
Ohio University
Sep 2012

Datagram Convergence Layers for the DTN Bundle and LTP Protocols
draft-irtf-dtnrg-dgram-clayer-00

Abstract

This document specifies the preferred method for transporting DTN protocol data over the Internet using datagrams. The specification covers convergence layers for the Bundle Protocol as well as the transportation of LTP segments. UDP and DCCP are the candidate datagram protocols discussed. UDP can only be used on a local network, or in cases where the DTN node implements explicit congestion control. DCCP does address the congestion control problem; however, the availability of implementations is limited.

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 5, 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 |
| 1.1. Requirements Language | 3 |
| 2. General Recommendation | 3 |
| 3. Recommendations for Implementers | 4 |
| 3.1. How and Where to Deal with Fragmentation | 4 |
| 3.1.1. DCCP | 5 |
| 3.1.2. UDP | 5 |
| 3.2. Bundle Protocol over a Datagram Convergence Layer | 5 |
| 3.2.1. DCCP | 5 |
| 3.2.2. UDP | 6 |
| 3.3. LTP over a Datagram Convergence Layer | 6 |
| 3.3.1. DCCP | 6 |
| 3.3.2. UDP | 6 |
| 3.4. Keep Alive Option | 6 |
| 3.5. Checksums | 6 |
| 3.5.1. DCCP | 7 |
| 3.5.2. UDP | 7 |
| 3.6. DCCP Availability | 7 |
| 3.7. DCCP Congestion Control Modules | 7 |
| 4. Acknowledgements | 8 |
| 5. IANA Considerations | 8 |
| 6. Security Considerations | 8 |
| 7. References | 8 |
| 7.1. Normative References | 8 |
| 7.2. Informative References | 9 |
| Authors' Addresses | 10 |

1. Introduction

Delay/Disruption Tolerant Network (DTN) communication protocols include the Bundle Protocol described in RFC 5050 [RFC5050], which provides reliable transmission of application data blocks (bundles) through optional intermediate custody transfer, and the Licklider Transmission Protocol (LTP), RFCs 5325 [RFC5325], 5326 [RFC5326], and 5327 [RFC5327] which can be used to transmit bundles reliably and efficiently over a point to point link. It is often desirable to test these protocols over Internet Protocol links. draft-irtf-dtnrg-tcp-clayer [I-D.irtf-dtnrg-tcp-clayer] defines a method for transporting bundles over TCP. This draft specifies the preferred method for transmitting either bundles or LTP blocks across the Internet using datagrams in place of TCP.

1.1. 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 RFC 2119 [RFC2119].

2. General Recommendation

In order to utilize DTN protocols across the Internet, whether for testing purposes or as part of a larger network path, it is necessary to encapsulate them into a standard Internet protocol so that they travel easily across the Internet. This is particularly true for LTP, which provides no endpoint addressing. This encapsulation choice needs to be made carefully in order to avoid redundancy, since DTN protocols may provide their own reliability mechanisms.

TCP, a logical choice, guarantees reliability and provides congestion control. Congestion control is vital to the continued functioning of the Internet, particularly for situations where data will be sent at arbitrarily fast data rates. Because the Bundle Protocol offers neither congestion control nor reliability, TCP is the RECOMMENDED choice for its encapsulation. draft-irtf-dtnrg-tcp-clayer [I-D.irtf-dtnrg-tcp-clayer] defines the method for transporting bundles over TCP.

LTP, on the other hand, offers it's own form of reliability. Particularly for testing purposes, it makes no sense to run LTP over a protocol, like TCP, that offers reliability already. In addition, running LTP over TCP would reduce the flexibility available to users, since LTP offers more control over what data is delivered reliably and what data is delivered best effort, a feature that TCP lacks. As such, it would be better to run LTP over an unreliable protocol.

One solution would be to use UDP. UDP provides no reliability, allowing LTP to manage that itself. However, UDP does not provide congestion control. Because LTP is designed to run over fixed rate radio links it does provide rate control, but not congestion control. Lack of congestion control in network connections is a major problem that can cause artificially high loss rates and/or serious fairness issues. Previous standards documents are unanimous in recommending congestion control for protocols to be used on the Internet, see RFCs 2914 [RFC2914], 5405 [RFC5405], and 2309 [RFC2309], among others. RFC 5405 [RFC5405], in particular, calls congestion control "vital" for "applications that can operate at higher, potentially unbounded data rates". Therefore, any application using UDP to transport LTP segments or Bundles MUST implement congestion control consistent with RFC 5405.

Alternatively, the Datagram Congestion Control Protocol (DCCP) [RFC4340] was designed specifically to provide congestion control without reliability for those applications that traverse the Internet but do not desire to retransmit lost data. As such, it is RECOMMENDED that, if possible, DCCP be used to transport LTP segments across the Internet.

3. Recommendations for Implementers

3.1. How and Where to Deal with Fragmentation

The Bundle Protocol allows bundles with sizes limited only by node resource constraints. In IPv4, the maximum size of a UDP datagram is nearly 64KB. In IPv6, when using jumbograms [RFC2675], UDP datagrams can be up to 4GB in size [RFC2147]. It is well understood that sending large IP datagrams that must be fragmented by the network has enormous efficiency penalties [Kent88]. The primary efficiency penalty is increased loss probability. When a large datagram is broken into a number of fragments, the original datagram can only be recreated if all the fragments arrive at the ultimate destination for reassembly. When transmitted over a network with a packet loss probability of 2%, for example, a single, unfragmented datagram will arrive with probability 98%; a large datagram fragmented into 10 fragments will have all of its fragments arrive with probability $98\%^{10}$, giving a datagram arrival probability of only 81.7%. The higher-level protocol using UDP for delivery can retransmit lost UDP datagrams, but cannot retransmit lost IP datagram fragments. Therefore, retransmitting large, lost datagrams because of a small number of missing fragments can require many more packets than retransmitting a number of smaller, unfragmented datagrams because only the missing pieces need to be retransmitted. The other efficiency penalty paid by fragmentation that would be significant

for DTN is the resources (time, complexity, and memory) required for IP reassembly. If the Bundle Protocol is being encapsulated in DCCP or UDP, the bundle protocol specification provides a bundle fragmentation concept [RFC5050] that allows a large bundle to be divided into bundle fragments, each of which SHOULD be created of sufficiently small size that it can then be encapsulated into a datagram that will not need to be fragmented.

3.1.1. DCCP

Because DCCP implementations are not required to support IP fragmentation and are not allowed to enable it by default, a DCCP CL MUST NOT accept data segments that cannot be sent as one MTU sized datagram.

3.1.2. UDP

When an LTP CL is using UDP for datagram delivery, it SHOULD NOT create segments that will result in UDP datagrams that will need to be fragmented, as discussed above.

Without information from elsewhere in the networking stack about path MTU, the protocol can assume a minimum path MTU that would allow 512 bytes of UDP data [RFC0791] over IPv4 or (1280-(UDP and IP header sizes)) bytes [RFC1883] over IPv6.

3.2. Bundle Protocol over a Datagram Convergence Layer

In general, the use of the bundle protocol over a datagram CL is discouraged. Bundles can be of (almost) arbitrary length, and the bundle protocol does not include an effective retransmission mechanism. Whenever possible the bundle protocol SHOULD be operated over the TCP Convergence Layer or over LTP.

If a datagram CL is used for transmission of bundles, every packet MUST contain exactly one bundle or four zero octets as a keep-alive. The CL SHOULD use available operating system services to obtain the largest supported packet size, and MAY use the default packet size limit if path-specific information is not available. For bundles that are too large for the supported packet size, the bundle protocol fragmentation process SHOULD be used to transmit the large bundle.

3.2.1. DCCP

The DCCP CL for bundle protocol use SHOULD use the IANA assigned port 4556/DCCP and service code 1685351985; the use of other port numbers and service codes is implementation specific.

3.2.2. UDP

The UDP CL for bundle protocol use SHOULD use the IANA assigned port 4556/UDP; the use of other port numbers is implementation specific.

3.3. LTP over a Datagram Convergence Layer

LTP is designed as a point to point protocol within DTN, and it provides intrinsic acknowledgement and retransmission facilities. Transmission of LTP over a datagram CL is therefore the most appropriate choice. When a datagram CL is used to transmit LTP data, every packet MUST contain exactly one LTP segment or four zero octets as a keep-alive. The CL SHOULD use available operating system services to obtain the largest supported packet size, and MAY use the default packet size limit if path-specific information is not available. LTP MUST perform segmentation in such a way as to insure that every LTP segments fits into a single packet.

3.3.1. DCCP

The DCCP CL for LTP SHOULD use the IANA assigned port 1113/DCCP and service code 7107696; the use of other port numbers and service codes is implementation specific.

3.3.2. UDP

The UDP CL for LTP SHOULD use the IANA assigned port 1113/UDP; the use of other port numbers is implementation specific.

3.4. Keep Alive Option

It may be desirable for a UDP or DCCP CL to send "keep-alive" packets during extended idle periods. This may be needed to refresh a contact table entry at the destination, or to maintain an address mapping in a NAT or a dynamic access rule in a firewall. Therefore, the CL MAY send a packet containing exactly 4 octets of zero bits. The CL receiving such a packet MUST discard this packet; the receiving CL may then perform local maintenance of its state tables, these maintenance functions are not covered in this draft. Note that "real" CL packets will always contain more than 4 octets of information (either the bundle or the LTP header); keep-alive packets will therefore never be mistaken for actual data packets.

3.5. Checksums

Both the core bundle protocol specification and core LTP specification assume that they are transmitting over an erasure channel, i.e. a channel that either delivers packets correctly or not

at all.

3.5.1. DCCP

A DCCP CL transmitter **MUST**, therefore, ensure that the entire packet is checksummed by setting the Checksum Coverage to 0. Likewise, the DCCP CL receiver **MUST** ignore all packets with partial checksum coverage.

3.5.2. UDP

A UDP CL transmitter therefore **MUST NOT** disable UDP checksums, and the UDP CL receiver **MUST NOT** disable checking of received UDP checksums.

Even when UDP checksums are enabled a small probability of UDP packet corruption remains. In some environments it may be acceptable for LTP or the bundle protocol to occasionally receive corrupted input. In general, however, a UDP CL implementation **SHOULD** use optional security extensions available in the bundle protocol or LTP to protect against message corruption.

3.6. DCCP Availability

As of this writing, the most mature DCCP implementation seems to be the one in the Linux Kernel. DCCP has, unfortunately, been slow in making it's way into most of the major platforms. As a result, if no DCCP implementation is available for a target platform, tunneling LTP over UDP is acceptable. In such a case, the UDP CL either **MUST NOT** be used outside an isolated network for the transmission of any non-trivial amounts of data, or it **MUST** implement congestion control procedures as outlined in RFC 5405 [RFC5405].

3.7. DCCP Congestion Control Modules

DCCP supports pluggable congestion control modules in order to optimize it's behavior to particular environments. The two most common congestion control modules (CCIDs) are TCP-like Congestion Control (CCID2) [RFC4341] and TCP-Friendly Rate Control (CCID3) [RFC4342]. TCP-like Congestion Control is designed to emulate TCP's congestion control as much as possible. It is recommended for applications that want to send data as quickly as possible, while TCP-Friendly Rate Control is aimed at applications that want to avoid sudden changes in sending rate. DTN use cases seem to fit more into the first case so DCCP CL's **SHOULD** use TCP-like Congestion Control (CCID2) by default.

4. Acknowledgements

5. IANA Considerations

Port number assignments 1113/UDP and 4556/UDP have been registered with IANA. Port numbers 1113/DCCP for the transport of LTP, and 4556/DCCP for the transport of bundles have been requested. DCCP Service Codes 7107696 for tunneling LTP and 1685351985 for tunneling Bundle Protocol have been requested.

6. Security Considerations

This memo describes the use of datagrams to transport DTN application data. Hosts may be in the position of having to accept and process packets from unknown sources; the DTN Endpoint ID can be discovered only after the bundle has been retrieved from the DCCP or UDP packet. Hosts SHOULD use authentication methods available in the DTN specifications to prevent malicious hosts from inserting unknown data into the application.

Hosts need to listen for and process DCCP or UDP data on the known LTP or bundle protocol ports. A denial of service scenario exists where a malicious host sends datagrams at a high rate, forcing the receiving hosts to use its resources to process and attempt to authenticate this data. Whenever possible, hosts SHOULD use IP address filtering to limit the origin of packets to known hosts.

7. References

7.1. Normative References

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.
- [RFC1883] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 1883, December 1995.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2147] Borman, D., "TCP and UDP over IPv6 Jumbograms", RFC 2147, May 1997.
- [RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.

- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, March 2006.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.
- [RFC5325] Burleigh, S., Ramadas, M., and S. Farrell, "Licklider Transmission Protocol - Motivation", RFC 5325, September 2008.
- [RFC5326] Ramadas, M., Burleigh, S., and S. Farrell, "Licklider Transmission Protocol - Specification", RFC 5326, September 2008.
- [RFC5327] Farrell, S., Ramadas, M., and S. Burleigh, "Licklider Transmission Protocol - Security Extensions", RFC 5327, September 2008.

7.2. Informative References

- [I-D.irtf-dtnrg-tcp-clayer] Demmer, M., Ott, J., and S. Perreault, "Delay Tolerant Networking TCP Convergence Layer Protocol", draft-irtf-dtnrg-tcp-clayer-04 (work in progress), August 2012.
- [Kent88] Kent, C. and J. Mogul, "Fragmentation considered harmful.", 1988, <<http://doi.acm.org/10.1145/55482.55524>>.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

Authors' Addresses

Hans Kruse
Ohio University
292 Lindley Hall
Athens, OH 45701
United States

Phone: +1 740 593 4891
Email: kruse@ohiou.edu

Samuel Jero
Ohio University
Athens, Ohio 45701
United States

Email: sj323707@ohio.edu

Shawn Ostermann
Ohio University
Stocker Engineering Center
Athens, OH 45701
United States

Phone: +1 740 593 1566
Email: ostermann@eecs.ohiou.edu

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.

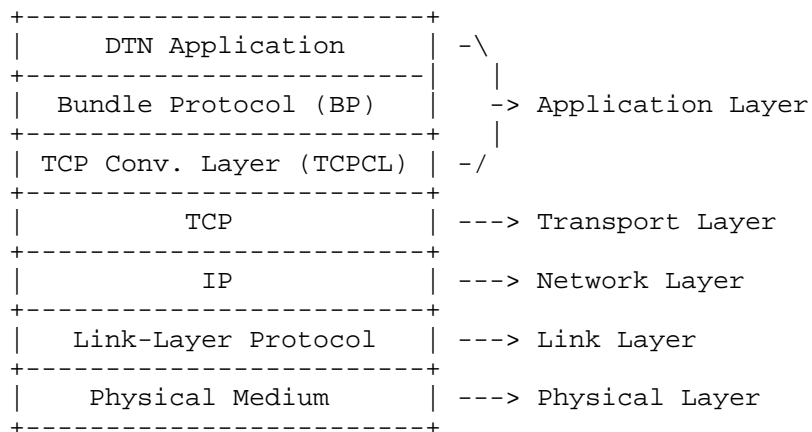


Figure 1: The locations of the bundle protocol and the TCP convergence layer protocol in the Internet protocol stack

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.

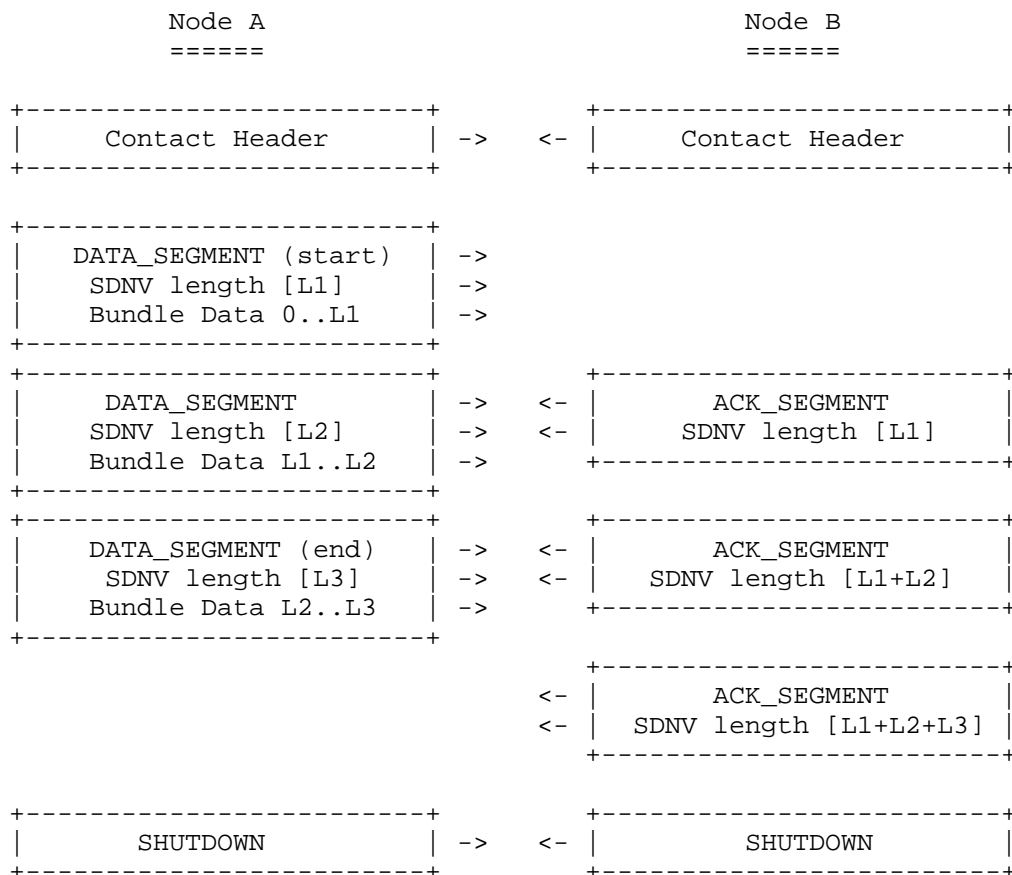


Figure 2: A simple visual example of the flow of protocol messages on a single TCP session between two nodes (A and B)

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:

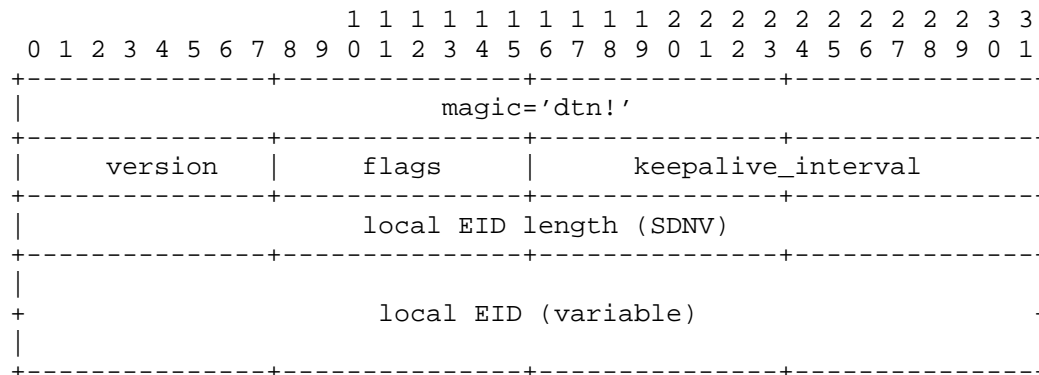


Figure 3: Contact Header Format

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:

```

  0 1 2 3 4 5 6 7
+---+---+---+---+---+
| type | flags |
+---+---+---+---+---+

```

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:

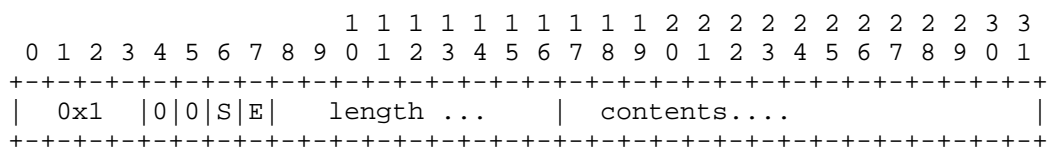


Figure 4: Format of bundle data segment messages

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:

```

          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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 0x2  |0|0|0|0|   acknowledged length ...                      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

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 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

The format of the LENGTH message is as follows:

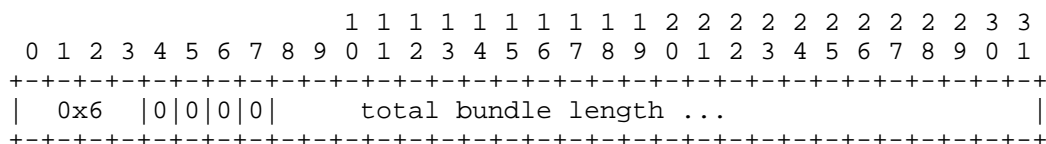


Figure 6: Format of LENGTH messages

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

KEEPAALIVE (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:

```

          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
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0x5   | 0|0|R|D| reason (opt) | reconnection delay (opt) |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

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

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: February 24, 2013

J. Zinky
A. Caro
Raytheon BBN Technologies
G. Stein
Laboratory for
Telecommunications Sciences
August 23, 2012

Bundle Protocol Erasure Coding Basic Objects
draft-zinky-dtnrg-erasure-coding-objects-00

Abstract

This document defines the Basic Data Objects formats for the Erasure Coding Extension [ErasureCoding] to the Delay and Disruption Tolerant Network (DTN) Bundle Protocol [RFC5050]. The File Data Object format is used to store a binary file and includes metadata for the file name and path name. The Bundle Data Object format is used to store a large DTN Bundle and to map its implicit Transfer Specification to the headers of the Encoding Bundles.

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 February 24, 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. Terminology | 4 |
| 2.1. Definitions | 4 |
| 2.2. Abbreviations | 4 |
| 2.3. Requirements Notation | 4 |
| 3. File Data Object Format Type = 1 | 5 |
| 4. Bundle Data Object Format Type = 2 | 8 |
| 4.1. Steps for Encoding a Bundle | 8 |
| 4.2. Steps for Decoding a Bundle | 9 |
| 4.3. Bundle Transfer Specification | 9 |
| 5. Security Considerations | 11 |
| 6. IANA Considerations | 12 |
| 7. References | 13 |
| 7.1. Normative References | 13 |
| 7.2. Informative References | 13 |
| Authors' Addresses | 14 |

1. Introduction

Data Object formats define how an Application Layer data structure will be stored as an array of octets that will be transmitted using the Erasure Coding Extension to the Bundle protocol. The octet array will be divided into equal length Chunks that are the input and output of the Coding Layer. The Coding Layer does not offer any fields for storing the Data Object Length in its headers. Data Object formats have the responsibility for storing the Data Object Length, the Data Object itself, associated metadata, and padding within the octet array. The Coding Layer offers a service that MAY deliver Chunks as they are decoded instead of waiting for all chunks to be decoded. But both Data Object types defined in this document can not make use this feature and MUST have every Chunk delivered all together.

The Data Object Format MAY put requirements and constraints into the Data Object Layer's Transfer Specification. The File Data Object format does not define any restrictions. The Bundle Data Object format defines an actionable Transfer Specification that is based on the implicit transfer specification in the original Bundle Header.

This document defines two Data Object formats; a File Data Object format and a Bundle Data Object format. The File Data Object format is used to transfer a binary file between two applications. The Bundle Data Object format is used by DTN Bundle Protocol Agents (BPAs) to divide large Bundles into Chunks to take advantage of Forward Error Correction services. Each format is described in its own section. Future documents could define, additional Data Object formats, such as mime [RFC2045], zip, or video. This document ends with discussions on Security and IANA considerations.

2. Terminology

The terminology used in this document follows the terminology of the Erasure Coding Extension to the Bundle Protocol [ErasureCoding]. Only terms specific to the Basic Data Objects are described in this section.

2.1. Definitions

Data Object Format Type is a field in the Erasure Coding Extension Block [ErasureCoding]. It specifies the format for the array of octets that hold the Data Object and its meta data. This document defines two Data Object Format Types and their type number.

Data Object Chunks are ordered equal length pieces of the octet array that store the Data Object, metadata, and padding. Chunks are created in the Data Object Layer and passed to the Coding Layer where they are encoded, transferred, and decoded.

2.2. Abbreviations

BPA: Bundle Protocol Agent [RFC5050]

DTN: Delay/Disruption Tolerant Network [RFC5050]

SDNV: Self-Delimiting Numeric Values, see [RFC6256]

2.3. 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].

3. File Data Object Format Type = 1

The File Data Object format stores a binary data file and the associated metadata about its name and intended storage path at the destination. The file format contains three sections, the file header, the file data, and the padding. The First Chunk MUST contain the whole file header, which constrains the minimum Chunk Length to be at least as long as the file header. Note that the file header is variable length, thus this constraint is specific to the Data Object. Only the last Chunk MAY contain padding, all other Chunks MUST NOT contain Padding. Except for the constraint above, the File Data Object Format does not restrict the Transfer Specification, that is Application Layer is responsibility for creating the Transfer Specification. The encoding and decoding process follows the procedures in the Erasure Coding Extension [ErasureCoding].

The octet array has the following format:

| Field | Type | Description |
|------------------|----------|---|
| Type | 4 Octets | 0xECECECEC: File Data Object Format Type constant. A magic number used to check that the decode process was successful. |
| Version | 4 Octets | 0x00000001: Version number of header which increments with newer versions. |
| Format: | 4 Octets | 0x00000001: Format of the File Data Object content. A value of one (1) specifies the 8-bit binary format. Other formats could be defined in the future, such as compressed or radix-64. |
| File UUID | 128 bits | Must match Data Object UUID in Erasure Coding Extension Block. |
| File Data Length | 8 Octets | Length of file in octets. |
| File Name | String | Name and extension of the File Data. |
| | 4 Octets | File Name Length, not including null terminator. |
| | Octets | Array of Octets for File Name, whose length is File Name Length. |
| | Octet | x00: Null Terminator. |
| Path Name | String | Path For the File Data. |
| | 4 Octets | Path Name Length, not including null terminator. |
| | Octets | Array of Octets for Path Name, whose length is Path Name Length. |
| | Octet | x00: Null Terminator. |
| File Data | Octets | Octet array for the File Data, whose length is File Data Length. |

4. Bundle Data Object Format Type = 2

The Bundle Data Object Format is used to divide a large Bundle into many smaller Chunks and to transfer those Chunks as Encoding Bundles using the Forward Error Correcting (FEC) services of the Erasure Coding Extension. The bundle format contains only two sections, the binary format of the bundle and padding. The Bundle is stored into the octet array using its over-the-wire representation. This allows for easy capture and reinsertion into the DTN. The octet array has the following format.

| Field | Type | Description |
|-------------|--------|--|
| Bundle Data | Octets | Octet array that is the over-the-wire binary format for the large Bundle. The destination MUST parse the large Bundle itself to obtain the Data Object Length of the large Bundle. |
| Padding | Octets | Extra Octets needed to pad the last Chunk to be the full Chunk Length. The sender MUST pad with x00. The receiver MUST ignore these octets. |

Table 2: Bundle Data Object Format

4.1. Steps for Encoding a Bundle

The Source Encoder in the Sending BPA performs the following steps to encode a large Bundle.

Step 1: The Sending BPA receives a large-bundle with a source and destination EIDs addressed as:

```
From: dtn://SourceBPA/SourceApp
To: dtn://DestBPA/DestApp
```

Step 2: The Source Encoder processes Bundles addressed to EIDs with the "dtn" scheme. The Transfer Specification for the large Bundle is derived from the large Bundle header, see Section 4.3. Note that the destination EID for the large Bundle is registered at the BPA, whose address is "DestBPA".

Step 3: Encoding Bundles are sent to the Destination Decoder at the "DestBPA" BPA using the Transfer Specification and EID addresses:

From: ebr://SourceBPA/ebr
To: ebr://DestBPA/ebr

Step 4: The Source Encoder MAY delete the original large Bundle before its expiration time once the Encoding Bundles are sent.

4.2. Steps for Decoding a Bundle

The following steps are performed by the Destination Decoder to decode a group of Encoding Bundles back into the original large Bundle.

Step 1: The Destination Decoder acts as an DTN application and uses the "ebr" extension to the base EID for the destination BPA.

Step 2: When Encoding Bundles arrive at the destination Decoder, they are sorted by UUID and stored in the corresponding Encoding Sets.

Step 3: When enough Encoding Bundles are in an Encoding Set, the Encoding Set is decoded into a large bundle.

Step 4: Destination Decoder injects the large Bundle back into the DTN routing layer, which determines further routing of the large Bundle.

Step 5: The Destination Decoder MAY delete the Encoding Set and its Encoding Bundles once the large Bundle is delivered to the routing layer.

Step 6: The Destination Decoder MAY send a "Stop" and/or a "Purge" end-to-end acknowledgement messages back to the Source Encoder using the EID, "ebr://SourceBPA/ebr"

4.3. Bundle Transfer Specification

The Transfer Specification is derived from the header of the large Bundle. Fields are extracted from the bundle header and are copied into the primary block and extension blocks of the Encoding Bundles. The following fields in the large Bundle are used in the Transfer Specification:

Source EID is changed to ebr://SourceBPA/ebr. Where SourceBPA is the BPA of the Source Encoder.

Destination EID is changed to ebr://DestBPA/ebr. Where DestBPA is the BPA of the Destination Decoder.

Creation Timestamp is changed to the time the Encoding Bundle was created, not to the original large Bundle creation time.

Life Time is changed to expire at the same time as the original large Bundle.

Age Extension Block is processed as-if the original large Bundle is being fragmented.

Class of Service bits from the Processing Control Flags is copied to the Encoding Bundles.

Singleton Destination bit from the Processing Control Flags is copied to the Encoding Bundles.

Request reporting bits from the Processing Control Flags MUST NOT be set in Encoding Bundles.

Extension Blocks The Bundle fragmentation rules guide which extension blocks to include in the Encoding Bundles. If the "replicate" bit is set in the Block Processing Control Flags field of the extension block, then the Extension block MUST be put into the Encoding Bundles. If the replicate bit is zero, the Extension Block MUST NOT be put into the Encoding Bundles, but will still be part of the large Bundle sent as the Data Object octet array.

5. Security Considerations

No additional security considerations have been identified beyond those described in [ErasureCoding]

6. IANA Considerations

The Basic Data Object Formats define two types. The assigned IDs should be less than 128 in order to fit into one octet using SDNV values. The reference implementation uses the following Data Object Format Types:

File = 1

Bundle = 2

7. References

7.1. Normative References

- [ErasureCoding] Zinky, J., Caro, A., and G. Stein, "Bundle Protocol Erasure Coding Extension", draft-zinky-dtnrg-erasure-coding-extension-00 (work in progress), Aug 2012.
- [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.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.

7.2. Informative References

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

Authors' Addresses

John Zinky
Raytheon BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: jzinky@bbn.com

Armando Caro
Raytheon BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: acaro@bbn.com

Gregory Stein
Laboratory for Telecommunications Sciences
8080 Greenmead Drive
College Park, MD 20740
US

Email: gstein@ece.umd.edu

