Delay-Tolerant Networking Working Group S. Burleigh
Internet Draft JPL, Calif. Inst. Of Technology
Intended status: Standards Track September 14, 2018
Expires: March 2019

Simple TCP Convergence-Layer Protocol
draft-burleigh-dtn-stcp-00.txt

Abstract

This document describes a Simple TCP (STCP) "convergence-layer" protocol for the Delay-Tolerant Networking (DTN) Bundle Protocol (BP). STCP uses Transmission Control Protocol (TCP) to transmit BP "bundles" from one BP node to another node to which it is topologically adjacent in the BP network. The services provided by the STCP convergence-layer protocol adapter utilize a standard TCP connection for the purposes of bundle transmission.

Table of Contents

1. Introduction

This document describes the Simple TCP (STCP) protocol, a Delay-Tolerant Networking (DTN) Bundle Protocol (BP) [RFC5050] "convergence layer" protocol that uses a standard TCP connection to transmit bundles from one BP node to another node to which it is topologically adjacent in the BP network.

Conformance to the STCP convergence-layer protocol specification is OPTIONAL for BP nodes.

Each BP node that conforms to the STCP specification includes an STCP convergence-layer adapter (SCLA). Every SCLA engages in communication via the Transmission Control Protocol [RFC0793].

Like any convergence-layer adapter, the STCP CLA provides:

- A transmission service that sends an outbound bundle (from the bundle protocol agent) to a peer CLA via the STCP convergence layer protocol.
- A reception service that delivers to the bundle protocol agent an inbound bundle that was sent by a peer CLA via the STCP convergence layer protocol.

Transmission of bundles via STCP is "reliable" to the extent that TCP itself is reliable. STCP provides no supplementary error detection and recovery procedures. In particular, STCP does not provide to the sender any intermediate reporting of reception progress.

2. Conventions used in this document

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].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

3. STCP Design Elements

3.1. STCP Sessions

An STCP "session" is formed when a TCP connection is established by the matching of an active TCP OPEN request issued by some SCLA, termed the session's "sender", with a passive TCP OPEN request issued by some SCLA, termed the session's "receiver". That portion of the state of a session that is exposed to the session's sender is termed the "transmission element" of the session. That portion of the state of a session that is exposed to the session's receiver is termed the "reception element" of the session.

The values of the parameters constraining STCP's TCP connection establishment, including the establishment of Transport Layer Security (TLS; [RFC8446]) sessions within the connections, are assumed to be provided by management. At some point a discovery protocol may be developed that enables these values to be declared automatically; such protocol is beyond the scope of this specification.

STCP sessions are unidirectional; that is, bundles transmitted via an STCP session are transmitted only from the session's sender to its receiver. When bidirectional exchange of bundles between SCLAs via STCP is required, two sessions are formed, one in each direction.

Closure of either element of a session MAY occur either upon request of the bundle protocol agent or upon detection of any error. Closure of either element of an STCP session SHALL cause the corresponding TCP connection to be terminated (unless termination of that connection was in fact the cause of the closure of that session element). Since termination of the associated TCP connection will result in errors at the other element of the session, termination of either element of the session will effectively terminate the session.

3.2. STCP Protocol Data Units

An STCP protocol data unit (SPDU) is simply a serialized bundle preceded by an integer indicating the length of that serialized bundle. An SPDU is constructed as follows.

Each SPDU SHALL be represented as a CBOR array. The number of items in the array SHALL be 2.

The first item of the SPDU array SHALL be the length of the serialized bundle that is encapsulated in the SPDU, represented as a CBOR unsigned integer.

The second item of the SPDU array SHALL be a single serialized BP bundle, termed the "encapsulated bundle", represented as a CBOR byte string.

4. STCP Procedures

4.1. SPDU Transmission

When an SCLA is requested by the bundle protocol agent to send a bundle to a peer SCLA identified by some IP address and port number:

- If no STCP session enabling transmission to that SCLA has been formed, the SCLA SHALL attempt to form that session. If this attempt is unsuccessful, the SCLA SHALL inform the bundle protocol agent that its data sending procedures with regard to this bundle have concluded and transmission of the bundle was unsuccessful; no further steps of this procedure will be attempted.

- The SCLA SHALL form an SPDU from the subject bundle.
- The SCLA SHALL attempt to send this SPDU to the peer SCLA by TCP via the transmission element of the session formed for this purpose.
  - If that transmission is completed without error, the SCLA SHALL inform the bundle protocol agent that its data sending procedures with regard to this bundle have concluded and transmission of the bundle was successful.
  - Otherwise:
    - The transmission element SHALL be closed.
    - The SCLA SHALL inform the bundle protocol agent that its data sending procedures with regard to this bundle have concluded and transmission of the bundle was unsuccessful.

4.2. Reception Session Formation

An SCLA that is required to receive (rather than only transmit) bundles SHALL issue a passive TCP OPEN. Whenever TCP matches that passive OPEN with an active TCP OPEN issued by some SCLA, an STCP session is formed as noted earlier; SPDUs may be received via the reception element of such session.

4.3. SPDU Reception

From the moment at which an STCP session reception element is first exposed to the moment at which it is closed, in a continuous cycle, the corresponding session’s receiver SHALL:

- Attempt to receive, by TCP via the corresponding session, the length of the next bundle sent via this session. If this attempt fails for any reason, the reception element SHALL be closed and no further steps of this procedure will be attempted.
- Attempt to receive, by TCP via the corresponding session, a serialized bundle of the indicated length. If this attempt fails for any reason, the reception element SHALL be closed and no further steps of this procedure will be attempted.
- Deliver the received serialized bundle to the bundle protocol agent.

5. Security Considerations

Because STCP constitutes a nearly negligible extension of TCP, it introduces virtually no security considerations beyond the well-known TCP security considerations.

An adversary could mount a denial-of-service attack by repeatedly establishing and terminating STCP sessions; well-understood DOS attack mitigations would apply.

Maliciously formed bundle lengths could disrupt the operation of STCP session receivers, but STCP implementations need to be robust against incorrect bundle lengths in any case.

Maliciously crafted serialized bundles could be received and delivered to the bundle protocol agent, but that is not an STCP-specific security consideration: all bundles delivered to the BPA by all convergence-layer adapters need to be processed in awareness of this possibility.

6. IANA Considerations

No new IANA considerations apply.

7. References

7.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018.

7.2. Informative References

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

8. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Appendix A. For More Information

Please refer comments to dtn@ietf.org. The Delay Tolerant Networking Research Group (DTNRG) Web site is located at http://www.dtnrg.org.

Authors' Address

Scott Burleigh
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
US
Phone: +1 818 393 3353
Email: Scott.Burleigh@jpl.nasa.gov

Delay-Tolerant Networking   E. Birrane
Internet-Draft   K. McKeever
Intended status: Standards Track   JHU/APL
Expires: April 25, 2019   October 22, 2018

Bundle Protocol Security Specification
draft-ietf-dtn-bpsec-08

Abstract

This document defines a security protocol providing end to end data integrity and confidentiality services for the Bundle Protocol.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines security features for the Bundle Protocol (BP) [I-D.ietf-dtn-bpbis] and is intended for use in Delay Tolerant Networks (DTNs) to provide end-to-end security services.

The Bundle Protocol specification [I-D.ietf-dtn-bpbis] defines DTN as referring to "a networking architecture providing communications in and/or through highly stressed environments" where "BP may be viewed as sitting at the application layer of some number of constituent networks, forming a store-carry-forward overlay network". The term "stressed" environment refers to multiple challenging conditions including intermittent connectivity, large and/or variable delays, asymmetric data rates, and high bit error rates.

The BP might be deployed such that portions of the network cannot be trusted, posing the usual security challenges related to confidentiality and integrity. However, the stressed nature of the BP operating environment imposes unique conditions where usual transport security mechanisms may not be sufficient. For example, the store-carry-forward nature of the network may require protecting data at rest, preventing unauthorized consumption of critical resources such as storage space, and operating without regular contact with a centralized security oracle (such as a certificate authority).

An end-to-end security service is needed that operates in all of the environments where the BP operates.

1.1. Supported Security Services

BPSec provides end-to-end integrity and confidentiality services for BP bundles, as defined in this section.

Integrity services ensure that target data within a bundle are not changed from the time they are provided to the network to the time they are delivered at their destination. Data changes may be caused by processing errors, environmental conditions, or intentional manipulation. In the context of BPSec, integrity services apply to plain-text in the bundle.

Confidentiality services ensure that target data is unintelligible to nodes in the DTN, except for authorized nodes possessing special information. This generally means producing cipher-text from plain-text and generating authentication information for that cipher-text. Confidentiality, in this context, applies to the contents of target data and does not extend to hiding the fact that confidentiality exists in the bundle.

NOTE: Hop-by-hop authentication is NOT a supported security service in this specification, for three reasons.

1. The term "hop-by-hop" is ambiguous in a BP overlay, as nodes that are adjacent in the overlay may not be adjacent in physical connectivity. This condition is difficult or impossible to detect and therefore hop-by-hop authentication is difficult or impossible to enforce.

2. Networks in which BPSec may be deployed may have a mixture of security-aware and not-security-aware nodes. Hop-by-hop authentication cannot be deployed in a network if adjacent nodes in the network have different security capabilities.

3. Hop-by-hop authentication is a special case of data integrity and can be achieved with the integrity mechanisms defined in this specification. Therefore, a separate authentication service is not necessary.

1.2. Specification Scope

This document defines the security services provided by the BPSec. This includes the data specification for representing these services as BP extension blocks, and the rules for adding, removing, and processing these blocks at various points during the bundle’s traversal of the DTN.

BPSec applies only to those nodes that implement it, known as "security-aware" nodes. There might be other nodes in the DTN that do not implement BPSec. While all nodes in a BP overlay can exchange bundles, BPSec security operations can only happen at BPSec security-aware nodes.

BPSec addresses only the security of data traveling over the DTN, not the underlying DTN itself. Furthermore, while the BPSec protocol can provide security-at-rest in a store-carry-forward network, it does not address threats which share computing resources with the DTN and/or BPSec software implementations. These threats may be malicious software or compromised libraries which intend to intercept data or recover cryptographic material. Here, it is the responsibility of the BPSec implementer to ensure that any cryptographic material, including shared secret or private keys, is protected against access within both memory and storage devices.

This specification addresses neither the fitness of externally-defined cryptographic methods nor the security of their implementation. Different networking conditions and operational considerations require varying strengths of security mechanism such that mandating a cipher suite in this specification may result in too much security for some networks and too little security in others. It is expected that separate documents will be standardized to define cipher suites compatible with BPSec, to include operational cipher suites and interoperability cipher suites.

This specification does not address the implementation of security policy and does not provide a security policy for the BPSec. Similar to cipher suites, security policies are based on the nature and capabilities of individual networks and network operational concepts. This specification does provide policy considerations when building a security policy.

With the exception of the Bundle Protocol, this specification does not address how to combine the BPSec security blocks with other protocols, other BP extension blocks, or other best practices to achieve security in any particular network implementation.

1.3. Related Documents

This document is best read and understood within the context of the following other DTN documents:

"Delay-Tolerant Networking Architecture" [RFC4838] defines the architecture for DTNs and identifies certain security assumptions made by existing Internet protocols that are not valid in a DTN.

The Bundle Protocol [I-D.ietf-dtn-bpbis] defines the format and processing of bundles, defines the extension block format used to represent BPSec security blocks, and defines the canonicalization algorithms used by this specification.

The Bundle Security Protocol [RFC6257] and Streamlined Bundle Security Protocol [I-D.birrane-dtn-sbsp] documents introduced the concepts of using BP extension blocks for security services in a DTN. The BPSec is a continuation and refinement of these documents.

1.4. Terminology

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

This section defines terminology either unique to the BPSec or otherwise necessary for understanding the concepts defined in this specification.

- Bundle Source - the node which originates a bundle. The Node ID of the BPA originating the bundle.

- Forwarder - any node that transmits a bundle in the DTN. The Node ID of the Bundle Protocol Agent (BPA) that sent the bundle on its most recent hop.

- Intermediate Receiver, Waypoint, or "Next Hop" - any node that receives a bundle from a Forwarder that is not the Destination. The Node ID of the BPA at any such node.

- Path - the ordered sequence of nodes through which a bundle passes on its way from Source to Destination. The path is not necessarily known in advance by the bundle or any BPAs in the DTN.

- Security Block - a BPSec extension block in a bundle.

- Security Operation - the application of a security service to a security target, notated as OP(security service, security target). For example, OP(confidentiality, payload). Every security operation in a bundle MUST be unique, meaning that a security service can only be applied to a security target once in a bundle. A security operation is implemented by a security block.

- Security Service - the security features supported by this specification: integrity and confidentiality.

- Security Source - a bundle node that adds a security block to a bundle. The Node ID of that node.

- Security Target - the block within a bundle that receives a security-service as part of a security-operation.

2. Design Decisions

The application of security services in a DTN is a complex endeavor that must consider physical properties of the network, policies at each node, and various application security requirements. This section identifies those desirable properties that guide design decisions for this specification and are necessary for understanding the format and behavior of the BPSec protocol.

2.1. Block-Level Granularity

Security services within this specification must allow different blocks within a bundle to have different security services applied to them.

Blocks within a bundle represent different types of information. The primary block contains identification and routing information. The payload block carries application data. Extension blocks carry a variety of data that may augment or annotate the payload, or otherwise provide information necessary for the proper processing of a bundle along a path. Therefore, applying a single level and type of security across an entire bundle fails to recognize that blocks in a bundle represent different types of information with different security needs.

For example, a payload block might be encrypted to protect its contents and an extension block containing summary information related to the payload might be integrity signed but unencrypted to provide waypoints access to payload-related data without providing access to the payload.

2.2. Multiple Security Sources

A bundle can have multiple security blocks and these blocks can have different security sources. BPSec implementations MUST NOT assume that all blocks in a bundle have the same security operations and/or security sources.

The Bundle Protocol allows extension blocks to be added to a bundle at any time during its existence in the DTN. When a waypoint adds a new extension block to a bundle, that extension block MAY have security services applied to it by that waypoint. Similarly, a waypoint MAY add a security service to an existing extension block, consistent with its security policy.

When a waypoint adds a security service to the bundle, the waypoint is the security source for that service. The security block(s) which represent that service in the bundle may need to record this security source as the bundle destination might need this information for processing. For example, a destination node might interpret policy as it related to security blocks as a function of the security source for that block.

For example, a bundle source may choose to apply an integrity service to its plain-text payload. Later a waypoint node, representing a gateway to an insecure portion of the DTN, may receive the bundle and choose to apply a confidentiality service. In this case, the integrity security source is the bundle source and the confidentiality security source is the waypoint node.

2.3. Mixed Security Policy

The security policy enforced by nodes in the DTN may differ.

Some waypoints might not be security aware and will not be able to process security blocks. Therefore, security blocks must have their processing flags set such that the block will be treated appropriately by non-security-aware waypoints.

Some waypoints will have security policies that require evaluating security services even if they are not the bundle destination or the final intended destination of the service. For example, a waypoint could choose to verify an integrity service even though the waypoint is not the bundle destination and the integrity service will be needed by other nodes along the bundle's path.

Some waypoints will determine, through policy, that they are the intended recipient of the security service and terminate the security service in the bundle. For example, a gateway node could determine that, even though it is not the destination of the bundle, it should verify and remove a particular integrity service or attempt to decrypt a confidentiality service, before forwarding the bundle along its path.

Some waypoints could understand security blocks but refuse to process them unless they are the bundle destination.

2.4. User-Selected Cipher Suites

The security services defined in this specification rely on a variety of cipher suites providing integrity signatures, cipher-text, and other information necessary to populate security blocks. Users may select different cipher suites to implement security services. For example, some users might prefer a SHA2 hash function for integrity whereas other users might prefer a SHA3 hash function instead. The security services defined in this specification must provide a mechanism for identifying what cipher suite has been used to populate a security block.

2.5. Deterministic Processing

Whenever a node determines that it must process more than one security block in a received bundle (either because the policy at a waypoint states that it should process security blocks or because the node is the bundle destination) the order in which security blocks are processed must be deterministic. All nodes must impose this same deterministic processing order for all security blocks. This specification provides determinism in the application and evaluation of security services, even when doing so results in a loss of flexibility.

3. Security Blocks

3.1. Block Definitions

This specification defines three types of security block: the Security Association Block (SAB), the Block Integrity Block (BIB) and the Block Confidentiality Block (BCB).

The SAB is used to define security associations between two messaging endpoints. In this sense, they are similar to security associations used in other security protocols such as IPSec, with the exception that these associations may be pre-negotiated as a matter of policy, parameterized as part of their definition, or otherwise made fit for use in a challenged networking scenario.

The BIB is used to ensure the integrity of its plain-text security target(s). The integrity information in the BIB MAY be verified by any node along the bundle path from the BIB security source to the bundle destination. Security-aware waypoints add or remove BIBs from bundles in accordance with their security policy. BIBs are never used to sign the cipher-text provided by a BCB.

The BCB indicates that the security target(s) have been encrypted at the BCB security source in order to protect their content while in transit. The BCB is decrypted by security-aware nodes in the network, up to and including the bundle destination, as a matter of security policy. BCBs additionally provide authentication mechanisms for the cipher-text they generate.

3.2. Uniqueness

Security operations in a bundle MUST be unique; the same security service MUST NOT be applied to a security target more than once in a bundle. Since a security operation is represented as a security block, this limits what security blocks may be added to a bundle: if adding a security block to a bundle would cause some other security block to no longer represent a unique security operation then the new block MUST NOT be added. It is important to note that any cipher-text integrity mechanism supplied by the BCB is considered part of the confidentiality service and, therefore, unique from the plain-text integrity service provided by the BIB.

If multiple security blocks representing the same security operation were allowed in a bundle at the same time, there would exist ambiguity regarding block processing order and the property of deterministic processing blocks would be lost.

Using the notation OP(service, target), several examples illustrate this uniqueness requirement.

- Signing the payload twice: The two operations OP(integrity, payload) and OP(integrity, payload) are redundant and MUST NOT both be present in the same bundle at the same time.

- Signing different blocks: The two operations OP(integrity, payload) and OP(integrity, extension_block_1) are not redundant and both may be present in the same bundle at the same time. Similarly, the two operations OP(integrity, extension_block_1) and OP(integrity,extension_block_2) are also not redundant and may both be present in the bundle at the same time.

- Different Services on same block: The two operations OP(integrity, payload) and OP(confidentiality, payload) are not inherently redundant and may both be present in the bundle at the same time, pursuant to other processing rules in this specification.

3.3. Target Multiplicity

Under special circumstances, a single security block MAY represent multiple security operations as a way of reducing the overall number of security blocks present in a bundle. In these circumstances, reducing the number of security blocks in the bundle reduces the amount of redundant information in the bundle.

A set of security operations can be represented by a single security block when all of the following conditions are true.

- o The security operations apply the same security service. For example, they are all integrity operations or all confidentiality operations.

- o The security association parameters and key information for the security operations are identical.

- o The security source for the security operations is the same. Meaning the set of operations are being added/removed by the same node.

- o No security operations have the same security target, as that would violate the need for security operations to be unique.

- o None of the security operations conflict with security operations already present in the bundle.

When representing multiple security operations in a single security block, the information that is common across all operations is represented once in the security block, and the information which is different (e.g., the security targets) are represented individually. When the security block is processed all security operations represented by the security block MUST be applied/evaluated at that time.

3.4. Target Identification

A security target is a block in the bundle to which a security service applies. This target must be uniquely and unambiguously identifiable when processing a security block. The definition of the extension block header from [I-D.ietf-dtn-bpbis] provides a "Block Number" field suitable for this purpose. Therefore, a security target in a security block MUST be represented as the Block Number of the target block.

3.5. Block Representation

Each security block uses the Canonical Bundle Block Format as defined in [I-D.ietf-dtn-bpbis]. That is, each security block is comprised of the following elements:

- o Block Type Code

- o Block Number

- o Block Processing Control Flags

- o CRC Type and CRC Field (if present)

- Block Data Length
- Block Type Specific Data Fields

Security-specific information for a security block is captured in the "Block Type Specific Data Fields".

3.6. Security Association Block

The SAB defines a security association (SA) between bundle messaging endpoints. This association captures the set of parameterized cipher suite information, key information, and other annotative information necessary to configure security services in the network.

In deployments where data communications are challenged, the SAB block may be omitted in favor of negotiating SAs using out-of-band mechanisms.

The Block Type Code of an SAB is as specified in Section 11.1.

The Block number, Block Processing Control Flags, CRC Type and CRC Field, and Block Data Length may be set in any way that conforms with security policy and in compliance with [I-D.ietf-dtn-bpbis].

The Block Type Specific Data Fields of the SAB MUST be encoded as a CBOR array, with each element of the array defining a unique SA.

An individual security association (SA) MUST be encoded as a CBOR array comprising the following fields, listed in the order in which they must appear.

Security Association Id:
: This field identifies the identifier for the SA. This field SHALL be represented by a CBOR unsigned integer.

Security Association Flags:
: This field identifies which optional fields are present in the security block. This field SHALL be represented as a CBOR unsigned integer containing a bit field of 5 bits indicating the presence or absence of other fields, as follows.

  Bit 1 (the most-significant bit, 0x10): EID Scope Flag.

  Bit 2 (0x08): Block Type Scope Flag.

  Bit 3 (0x04): Cipher Suite Id Present Flag.

  Bit 4 (0x02): Security Source Present Flag.

Bit 5 (the least-significant bit, 0x01): Security Association Parameters Present Flag.

In this field, a value of 1 indicates that the associated security block field MUST be included in the security block. A value of 0 indicates that the associated security block field MUST NOT be in the security block.

EID Scope (Optional Field):
This field identifies the message destinations (as a series of Endpoints) for which this SA should be applied. If this field is not present, the SA may be applied to any message endpoints or may be filtered in some other way in accordance with security policy. This field SHALL be represented by a CBOR array with each element containing an EID encoded in accordance with [I-D.ietf-dtn-bpbis] rules for representing Endpoint Identifiers (EIDs).

Block Type Scope (Optional Field):
This field identifies the block types for which this SA should be applied. If this field is not present, the SA may be applied to any block type or may be filtered in some other way in accordance with security policy. This field SHALL be represented by a CBOR array with each element containing a block type encoded in accordance with [I-D.ietf-dtn-bpbis] rules for representing block types.

Cipher Suite Id (Optional Field):
This field identifies the cipher suite used by this SA. If this field is not present, the cipher suite associated with this SA MUST be known through some alternative mechanisms, such as local security policy or out-of-band configuration. The cipher suite Id SHALL be presented by a CBOR unsigned integer.

Security Source (Optional Field):
This field identifies the Endpoint that inserted the security block in the bundle. If the security source field is not present then the source MUST be inferred from other information, such as the bundle source, previous hop, or other values defined by security policy. This field SHALL be represented by a CBOR array in accordance with [I-D.ietf-dtn-bpbis] rules for representing Endpoint Identifiers (EIDs).

Security Association Parameters (Optional Field):
This field captures one or more security association parameters that should be provided to security-aware nodes when processing the security service described by this security block. This

field SHALL be represented by a CBOR array. Each entry in this array is a single SA parameter. A single SA parameter SHALL also be represented as a CBOR array comprising a 2-tuple of the id and value of the parameter, as follows.

* Parameter Id. This field identifies which SA parameter is being specified. This field SHALL be represented as a CBOR unsigned integer. Parameter ids are selected as described in Section 3.11.

* Parameter Value. This field captures the value associated with this parameter. This field SHALL be represented by the applicable CBOR representation of the parameter, in accordance with Section 3.11.

The logical layout of the security association parameters array is illustrated in Figure 1.

```
+----------------+----------------+     +----------------+
|  Parameter 1   |  Parameter 2   | ... |  Parameter N   |
+------+---------+------+---------+     +------+---------+
|  Id  |  Value  |  Id  |  Value  |     |  Id  |  Value  |
+------+---------+------+---------+     +------+---------+
```

Figure 1: Security Association Parameters

Notes:

o It is RECOMMENDED that security association designers carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

3.7. Abstract Security Block

The structure of the security-specific portions of a security block is identical for both the BIB and BCB Block Types. Therefore, this section defines an Abstract Security Block (ASB) data structure and discusses the definition, processing, and other constraints for using this structure. An ASB is never directly instantiated within a bundle, it is only a mechanism for discussing the common aspects of BIB and BCB security blocks.

The fields of the ASB SHALL be as follows, listed in the order in which they must appear.

Security Targets:

This field identifies the block(s) targeted by the security operation(s) represented by this security block. Each target block is represented by its unique Block Number. This field SHALL be represented by a CBOR array of data items. Each target within this CBOR array SHALL be represented by a CBOR unsigned integer. This array MUST have at least 1 entry and each entry MUST represent the Block Number of a block that exists in the bundle. There MUST NOT be duplicate entries in this array.

Security Association Id:
This field identifies the cipher suite used to implement the security service represented by this block and applied to each security target. This field SHALL be represented by a CBOR unsigned integer.

Security Association Flags:
This field identifies which optional fields are present in the security block. This field SHALL be represented as a CBOR unsigned integer containing a bit field of 5 bits indicating the presence or absence of other security block fields, as follows.

Bit 1 (the most-significant bit, 0x10): reserved.

Bit 2 (0x08): reserved.

Bit 3 (0x04): reserved.

Bit 4 (0x02): Security Source Present Flag.

Bit 5 (the least-significant bit, 0x01): reserved.

In this field, a value of 1 indicates that the associated security block field MUST be included in the security block. A value of 0 indicates that the associated security block field MUST NOT be in the security block.

Security Source (Optional Field):
This field identifies the Endpoint that inserted the security block in the bundle. If the security source field is not present then the source MUST be inferred from other information, such as the bundle source, previous hop, or other values defined by security policy. This field SHALL be represented by a CBOR array in accordance with [I-D.ietf-dtn-bpbis] rules for representing Endpoint Identifiers (EIDs).

Security Results:

This field captures the results of applying a security service to the security targets of the security block. This field SHALL be represented as a CBOR array of target results. Each entry in this array represents the set of security results for a specific security target. The target results MUST be ordered identically to the Security Targets field of the security block. This means that the first set of target results in this array corresponds to the first entry in the Security Targets field of the security block, and so on. There MUST be one entry in this array for each entry in the Security Targets field of the security block.

The set of security results for a target is also represented as a CBOR array of individual results. An individual result is represented as a 2-tuple of a result id and a result value, defined as follows.

* Result Id. This field identifies which security result is being specified. Some security results capture the primary output of a cipher suite. Other security results contain additional annotative information from cipher suite processing. This field SHALL be represented as a CBOR unsigned integer. Security result ids will be as specified in Section 3.11.

* Result Value. This field captures the value associated with the result. This field SHALL be represented by the applicable CBOR representation of the result value, in accordance with Section 3.11.

The logical layout of the security results array is illustrated in Figure 2. In this figure there are N security targets for this security block. The first security target contains M results and the Nth security target contains K results.

```
+------------------------------+     +------------------------------+
|           Target 1           |     |           Target N           |
+------------+----+------------+     +------------------------------+
|  Result 1  |    |  Result M  | ... |  Result 1  |    |  Result K  |
+----+-------+ .. +----+-------+     +----+-------+ .. +----+-------+
| Id | Value |    | Id | Value |     | Id | Value |    | Id | Value |
+----+-------+    +----+-------+     +----+-------+    +----+-------+
```

Figure 2: Security Results

3.8. Block Integrity Block

A BIB is a bundle extension block with the following characteristics.

- The Block Type Code value is as specified in Section 11.1.

- The Block Type Specific Data Fields follow the structure of the ASB.

- A security target listed in the Security Targets field MUST NOT reference a security block defined in this specification (e.g., a BIB or a BCB).

- The Security Association Id MUST refer to a known SA that supports an end-to-end authentication-cipher suite or as an end-to-end error-detection-cipher suite.

- An EID-reference to the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source MAY be specified as part of key information described in Section 3.11.

Notes:

- It is RECOMMENDED that SA designers carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

- Since OP(integrity, target) is allowed only once in a bundle per target, it is RECOMMENDED that users wishing to support multiple integrity signatures for the same target define a multi-signature SA.

- For some SAs, (e.g., those using asymmetric keying to produce signatures or those using symmetric keying with a group key), the security information MAY be checked at any hop on the way to the destination that has access to the required keying information, in accordance with Section 3.10.

- The use of a generally available key is RECOMMENDED if custodial transfer is employed and all nodes SHOULD verify the bundle before accepting custody.

3.9. Block Confidentiality Block

A BCB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The Block Processing Control flags value can be set to whatever values are required by local policy, except that this block MUST have the "replicate in every fragment" flag set if the target of the BCB is the Payload Block. Having that BCB in each fragment indicates to a receiving node that the payload portion of each fragment represents cipher-text.

The Block Type Specific Data Fields follow the structure of the ASB.

A security target listed in the Security Targets field can reference the payload block, a non-security extension block, or a BIB. A BCB MUST NOT include another BCB as a security target. A BCB MUST NOT target the primary block.

The Security Association Id MUST refer to a known SA that supports a confidentiality cipher suite that supports authenticated encryption with associated data (AEAD).

Additional information created by the SA (such as additional authenticated data) can be placed either in a security result field or in the generated cipher-text. The determination of where to place these data is a function of the cipher suite used.

An EID-reference to the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source MAY be specified as part of key information described in Section 3.11.

The BCB modifies the contents of its security target(s). When a BCB is applied, the security target body data are encrypted "in-place". Following encryption, the security target Block Type Specific Data field contains cipher-text, not plain-text. Other block fields remain unmodified, with the exception of the Block Data Length field, which MUST be updated to reflect the new length of the Block Type Specific Data field.

Notes:

- It is RECOMMENDED that SA designers carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

- The BCB block processing control flags can be set independently from the processing control flags of the security target(s). The setting of such flags SHOULD be an implementation/policy decision for the encrypting node.

3.10. Block Interactions

The security block types defined in this specification are designed to be as independent as possible. However, there are some cases where security blocks may share a security target creating processing dependencies.

If a security target of a BCB is also a security target of a BIB, an undesirable condition occurs where a security aware waypoint would be unable to validate the BIB because one of its security target's contents have been encrypted by a BCB. To address this situation the following processing rules MUST be followed.

- When adding a BCB to a bundle, if some (or all) of the security targets of the BCB also match all of the security targets of an existing BIB, then the existing BIB MUST also be encrypted. This can be accomplished by either adding a new BCB that targets the existing BIB, or by adding the BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.

- When adding a BCB to a bundle, if some (or all) of the security targets of the BCB match some (but not all) of the security targets of a BIB, then a new BIB MUST be created and all entries relating to those BCB security targets MUST be moved from the original BIB to the newly created BIB. The newly created BIB MUST then be encrypted. This can be accomplished by either adding a new BCB that targets the new BIB, or by adding the new BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.

- A BIB MUST NOT be added for a security target that is already the security target of a BCB. In this instance, the BCB is already providing authentication and integrity of the security target and the BIB would be redundant, insecure, and cause ambiguity in block processing order.

- o A BIB integrity value MUST NOT be evaluated if the BIB is the security target of an existing BCB. In this case, the BIB data is encrypted.

- o A BIB integrity value MUST NOT be evaluated if the security target of the BIB is also the security target of a BCB. In such a case, the security target data contains cipher-text as it has been encrypted.

- o As mentioned in Section 3.8, a BIB MUST NOT have a BCB as its security target.

These restrictions on block interactions impose a necessary ordering when applying security operations within a bundle. Specifically, for a given security target, BIBs MUST be added before BCBs. This ordering MUST be preserved in cases where the current BPA is adding all of the security blocks for the bundle or whether the BPA is a waypoint adding new security blocks to a bundle that already contains security blocks.

NOTE: Since any cipher suite used with a BCB MUST be an AEAD cipher suite, it is inefficient and possibly insecure for a single security source to add both a BIB and a BCB for the same security target. In cases where a security source wishes to calculate both a plain-text integrity mechanism and encrypt a security target, a BCB with a cipher suite that generates such signatures as additional security results SHOULD be used instead.

3.11. SA Parameters and Result Identification

SA parameters and security results each represent multiple distinct pieces of information in a security block. Each piece of information is assigned an identifier and a CBOR encoding. Identifiers MUST be unique for a given SA but do not need to be unique across all SAs. Therefore, parameter ids and security result ids are specified in the context of an SA definition.

Individual BPSec SAs SHOULD use existing registries of identifiers and CBOR encodings, such as those defined in [RFC8152], whenever possible. SAs SHOULD define their own identifiers and CBOR encodings when necessary.

A SA can include multiple instances of the same identifier for a parameter or result in the SAB. Parameters and results are represented using CBOR, and any identification of a new parameter or result must include how the value will be represented using the CBOR specification. Ids themselves are always represented as a CBOR unsigned integer.

3.12. BSP Block Examples

This section provides two examples of BPSec blocks applied to a bundle. In the first example, a single node adds several security operations to a bundle. In the second example, a waypoint node received the bundle created in the first example and adds additional security operations. In both examples, the first column represents blocks within a bundle and the second column represents the Block Number for the block, using the terminology B1...Bn for the purpose of illustration.

3.12.1. Example 1: Constructing a Bundle with Security

In this example a bundle has four non-security-related blocks: the primary block (B1), two extension blocks (B4,B5), and a payload block (B6). The bundle source wishes to provide an integrity signature of the plain-text associated with the primary block, one of the extension blocks, and the payload. The resultant bundle is illustrated in Figure 3 and the security actions are described below.

```
                 Block in Bundle                ID
  +======================================+====+
  |          Primary Block               | B1 |
  +--------------------------------------+----+
  |              BIB                     | B2 |
  | OP(integrity, targets=B1, B5, B6)    |    |
  +--------------------------------------+----+
  |              BCB                     | B3 |
  |  OP(confidentiality, target=B4)      |    |
  +--------------------------------------+----+
  |       Extension Block     (encrypted) | B4 |
  +--------------------------------------+----+
  |       Extension Block                | B5 |
  +--------------------------------------+----+
  |          Payload Block               | B6 |
  +--------------------------------------+----+
```

Figure 3: Security at Bundle Creation

The following security actions were applied to this bundle at its time of creation.

o An integrity signature applied to the canonicalized primary block (B1), the second extension block (B5) and the payload block (B6). This is accomplished by a single BIB (B2) with multiple targets. A single BIB is used in this case because all three targets share a security source and policy has them share the same cipher suite, key, and cipher suite parameters. Had this not been the case, multiple BIBs could have been added instead.

- Confidentiality for the first extension block (B4). This is accomplished by a BCB (B3). Once applied, the contents of extension block B4 are encrypted. The BCB MUST hold an authentication signature for the cipher-text either in the cipher-text that now populated the first extension block or as a security result in the BCB itself, depending on which cipher suite is used to form the BCB. A plain-text integrity signature may also exist as a security result in the BCB if one is provided by the selected confidentiality cipher suite.

3.12.2. Example 2: Adding More Security At A New Node

Consider that the bundle as it is illustrated in Figure 3 is now received by a waypoint node that wishes to encrypt the first extension block and the bundle payload. The waypoint security policy is to allow existing BIBs for these blocks to persist, as they may be required as part of the security policy at the bundle destination.

The resultant bundle is illustrated in Figure 4 and the security actions are described below. Note that block IDs provided here are ordered solely for the purpose of this example and not meant to impose an ordering for block creation. The ordering of blocks added to a bundle MUST always be in compliance with [I-D.ietf-dtn-bpbis].

```
                   Block in Bundle               ID
      +======================================+====+
      |          Primary Block               | B1 |
      +--------------------------------------+----+
      |              BIB                     | B2 |
      | OP(integrity, targets=B1)            |    |
      +--------------------------------------+----+
      |              BIB         (encrypted) | B7 |
      | OP(integrity, targets=B5, B6)        |    |
      +--------------------------------------+----+
      |              BCB                     | B8 |
      | OP(confidentiality, target=B4,B6,B7) |    |
      +--------------------------------------+----+
      |              BCB                     | B3 |
      |  OP(confidentiality, target=B4)      |    |
      +--------------------------------------+----+
      |       Extension Block    (encrypted) | B4 |
      +--------------------------------------+----+
      |       Extension Block    (encrypted) | B5 |
      +--------------------------------------+----+
      |          Payload Block   (encrypted) | B6 |
      +--------------------------------------+----+
```

Figure 4: Security At Bundle Forwarding

The following security actions were applied to this bundle prior to its forwarding from the waypoint node.

- o Since the waypoint node wishes to encrypt blocks B5 and B6, it MUST also encrypt the BIBs providing plain-text integrity over those blocks. However, BIB B2 could not be encrypted in its entirety because it also held a signature for the primary block (B1). Therefore, a new BIB (B7) is created and security results associated with B5 and B6 are moved out of BIB B2 and into BIB B7.

- o Now that there is no longer confusion of which plain-text integrity signatures must be encrypted, a BCB is added to the bundle with the security targets being the second extension block (B5) and the payload (B6) as well as the newly created BIB holding their plain-text integrity signatures (B7). A single new BCB is used in this case because all three targets share a security source and policy has them share the same cipher suite, key, and cipher suite parameters. Had this not been the case, multiple BCBs could have been added instead.

4. Canonical Forms

Security services require consistency and determinism in how information is presented to cipher suites at the security source and at a receiving node. For example, integrity services require that the same target information (e.g., the same bits in the same order) is provided to the cipher suite when generating an original signature and when generating a comparison signature. Canonicalization algorithms are used to construct a stable, end-to-end bit representation of a target block.

Canonical forms are not transmitted, they are used to generate input to a cipher suite for security processing at a security-aware node.

The canonicalization of the primary block is as specified in [I-D.ietf-dtn-bpbis].

All non-primary blocks share the same block structure and are canonicalized as specified in [I-D.ietf-dtn-bpbis] with the following exceptions.

- If the service being applied is a confidentiality service, then the Block Type Code, Block Number, Block Processing Control Flags, CRC Type and CRC Field (if present), and Block Data Length fields MUST NOT be included in the canonicalization. Confidentiality services are used solely to convert the Block Type Specific Data Fields from plain-text to cipher-text.

- Reserved flags MUST NOT be included in any canonicalization as it is not known if those flags will change in transit.

These canonicalization algorithms assume that Endpoint IDs do not change from the time at which a security source adds a security block to a bundle and the time at which a node processes that security block.

Cipher suites used by SAs MAY define their own canonicalization algorithms and require the use of those algorithms over the ones provided in this specification. In the event of conflicting canonicalization algorithms, cipher suite algorithms take precedence over this specification.

5. Security Processing

This section describes the security aspects of bundle processing.

5.1. Bundles Received from Other Nodes

Security blocks must be processed in a specific order when received by a security-aware node. The processing order is as follows.

- o When BIBs and BCBs share a security target, BCBs MUST be evaluated first and BIBs second.

5.1.1. Receiving BCBs

If a received bundle contains a BCB, the receiving node MUST determine whether it has the responsibility of decrypting the BCB security target and removing the BCB prior to delivering data to an application at the node or forwarding the bundle.

If the receiving node is the destination of the bundle, the node MUST decrypt any BCBs remaining in the bundle. If the receiving node is not the destination of the bundle, the node MUST decrypt the BCB if directed to do so as a matter of security policy.

If the security policy of a security-aware node specifies that a bundle should have applied confidentiality to a specific security target and no such BCB is present in the bundle, then the node MUST process this security target in accordance with the security policy. This may involve removing the security target from the bundle. If the removed security target is the payload block, the bundle MUST be discarded.

If an encrypted payload block cannot be decrypted (i.e., the cipher-text cannot be authenticated), then the bundle MUST be discarded and processed no further. If an encrypted security target other than the payload block cannot be decrypted then the associated security target and all security blocks associated with that target MUST be discarded and processed no further. In both cases, requested status reports (see [I-D.ietf-dtn-bpbis]) MAY be generated to reflect bundle or block deletion.

When a BCB is decrypted, the recovered plain-text MUST replace the cipher-text in the security target Block Type Specific Data Fields. If the Block Data Length field was modified at the time of encryption it MUST be updated to reflect the decrypted block length.

If a BCB contains multiple security targets, all security targets MUST be processed when the BCB is processed. Errors and other processing steps SHALL be made as if each security target had been represented by an individual BCB with a single security target.

5.1.2. Receiving BIBs

If a received bundle contains a BIB, the receiving node MUST determine whether it has the final responsibility of verifying the BIB security target and removing it prior to delivering data to an application at the node or forwarding the bundle. If a BIB check fails, the security target has failed to authenticate and the security target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. Otherwise, if the BIB verifies, the security target is ready to be processed for delivery.

A BIB MUST NOT be processed if the security target of the BIB is also the security target of a BCB in the bundle. Given the order of operations mandated by this specification, when both a BIB and a BCB share a security target, it means that the security target must have been encrypted after it was integrity signed and, therefore, the BIB cannot be verified until the security target has been decrypted by processing the BCB.

If the security policy of a security-aware node specifies that a bundle should have applied integrity to a specific security target and no such BIB is present in the bundle, then the node MUST process this security target in accordance with the security policy. This may involve removing the security target from the bundle. If the removed security target is the payload or primary block, the bundle MAY be discarded. This action can occur at any node that has the ability to verify an integrity signature, not just the bundle destination.

If a receiving node does not have the final responsibility of verifying the BIB it MAY attempt to verify the BIB to prevent the needless forwarding of corrupt data. If the check fails, the node SHALL process the security target in accordance to local security policy. It is RECOMMENDED that if a payload integrity check fails at a waypoint that it is processed in the same way as if the check fails at the destination. If the check passes, the node MUST NOT remove the BIB prior to forwarding.

If a BIB contains multiple security targets, all security targets MUST be processed if the BIB is processed by the Node. Errors and other processing steps SHALL be made as if each security target had been represented by an individual BIB with a single security target.

5.2. Bundle Fragmentation and Reassembly

If it is necessary for a node to fragment a bundle payload, and security services have been applied to that bundle, the fragmentation rules described in [I-D.ietf-dtn-bpbis] MUST be followed. As defined there and summarized here for completeness, only the payload block can be fragmented; security blocks, like all extension blocks, can never be fragmented.

Due to the complexity of payload block fragmentation, including the possibility of fragmenting payload block fragments, integrity and confidentiality operations are not to be applied to a bundle representing a fragment. Specifically, a BCB or BIB MUST NOT be added to a bundle if the "Bundle is a Fragment" flag is set in the Bundle Processing Control Flags field.

Security processing in the presence of payload block fragmentation may be handled by other mechanisms outside of the BPSec protocol or by applying BPSec blocks in coordination with an encapsulation mechanism.

6. Key Management

There exist a myriad of ways to establish, communicate, and otherwise manage key information in a DTN. Certain DTN deployments might follow established protocols for key management whereas other DTN deployments might require new and novel approaches. BPSec assumes that key management is handled as a separate part of network management and this specification neither defines nor requires a specific key management strategy.

7. Security Policy Considerations

When implementing BPSec, several policy decisions must be considered. This section describes key policies that affect the generation, forwarding, and receipt of bundles that are secured using this specification. No single set of policy decisions is envisioned to work for all secure DTN deployments.

- If a bundle is received that contains more than one security operation, in violation of BPSec, then the BPA must determine how to handle this bundle. The bundle may be discarded, the block affected by the security operation may be discarded, or one security operation may be favored over another.

- BPAs in the network must understand what security operations they should apply to bundles. This decision may be based on the source of the bundle, the destination of the bundle, or some other information related to the bundle.

- If a waypoint has been configured to add a security operation to a bundle, and the received bundle already has the security operation applied, then the receiver must understand what to do. The receiver may discard the bundle, discard the security target and associated BPSec blocks, replace the security operation, or some other action.

- It is recommended that security operations only be applied to the blocks that absolutely need them. If a BPA were to apply security operations such as integrity or confidentiality to every block in the bundle, regardless of need, there could be downstream errors processing blocks whose contents must be inspected or changed at every hop along the path.

- It is recommended that BCBs be allowed to alter the size of extension blocks and the payload block. However, care must be taken to ensure that changing the size of the payload block while the bundle is in transit do not negatively affect bundle processing (e.g., calculating storage needs, scheduling transmission times, caching block byte offsets).

- Adding a BIB to a security target that has already been encrypted by a BCB is not allowed. If this condition is likely to be encountered, there are (at least) three possible policies that could handle this situation.

    1. At the time of encryption, a plain-text integrity signature may be generated and added to the BCB for the security target as additional information in the security result field.

    2. The encrypted block may be replicated as a new block and integrity signed.

    3. An encapsulation scheme may be applied to encapsulate the security target (or the entire bundle) such that the encapsulating structure is, itself, no longer the security target of a BCB and may therefore be the security target of a BIB.

- It is recommended that security policy address whether cipher suites whose cipher-text is larger (or smaller) than the initial plain-text are permitted and, if so, for what types of blocks. Changing the size of a block may cause processing difficulties for networks that calculate block offsets into bundles or predict transmission times or storage availability as a function of bundle size. In other cases, changing the size of a payload as part of encryption has no significant impact.

8. Security Considerations

Given the nature of DTN applications, it is expected that bundles may traverse a variety of environments and devices which each pose unique security risks and requirements on the implementation of security within BPSec. For these reasons, it is important to introduce key threat models and describe the roles and responsibilities of the BPSec protocol in protecting the confidentiality and integrity of the data against those threats. This section provides additional discussion on security threats that BPSec will face and describes how BPSec security mechanisms operate to mitigate these threats.

The threat model described here is assumed to have a set of capabilities identical to those described by the Internet Threat Model in [RFC3552], but the BPSec threat model is scoped to illustrate threats specific to BPSec operating within DTN environments and therefore focuses on man-in-the-middle (MITM) attackers. In doing so, it is assumed that the DTN (or significant portions of the DTN) are completely under the control of an attacker.

8.1. Attacker Capabilities and Objectives

BPSec was designed to protect against MITM threats which may have access to a bundle during transit from its source, Alice, to its destination, Bob. A MITM node, Mallory, is a non-cooperative node operating on the DTN between Alice and Bob that has the ability to receive bundles, examine bundles, modify bundles, forward bundles, and generate bundles at will in order to compromise the confidentiality or integrity of data within the DTN. For the purposes of this section, any MITM node is assumed to effectively be security-aware even if it does not implement the BPSec protocol. There are three classes of MITM nodes which are differentiated based on their access to cryptographic material:

- o Unprivileged Node: Mallory has not been provisioned within the secure environment and only has access to cryptographic material which has been publicly-shared.

- o Legitimate Node: Mallory is within the secure environment and therefore has access to cryptographic material which has been provisioned to Mallory (i.e., K_M) as well as material which has been publicly-shared.

- o Privileged Node: Mallory is a privileged node within the secure environment and therefore has access to cryptographic material which has been provisioned to Mallory, Alice and/or Bob (i.e. K_M, K_A, and/or K_B) as well as material which has been publicly-shared.

If Mallory is operating as a privileged node, this is tantamount to compromise; BPSec does not provide mechanisms to detect or remove Mallory from the DTN or BPSec secure environment. It is up to the BPSec implementer or the underlying cryptographic mechanisms to provide appropriate capabilities if they are needed. It should also be noted that if the implementation of BPSec uses a single set of shared cryptographic material for all nodes, a legitimate node is equivalent to a privileged node because K_M == K_A == K_B.

A special case of the legitimate node is when Mallory is either Alice or Bob (i.e., K_M == K_A or K_M == K_B). In this case, Mallory is able to impersonate traffic as either Alice or Bob, which means that traffic to and from that node can be decrypted and encrypted, respectively. Additionally, messages may be signed as originating from one of the endpoints.

8.2. Attacker Behaviors and BPSec Mitigations

8.2.1. Eavesdropping Attacks

Once Mallory has received a bundle, she is able to examine the contents of that bundle and attempt to recover any protected data or cryptographic keying material from the blocks contained within. The protection mechanism that BPSec provides against this action is the BCB, which encrypts the contents of its security target, providing confidentiality of the data. Of course, it should be assumed that Mallory is able to attempt offline recovery of encrypted data, so the cryptographic mechanisms selected to protect the data should provide a suitable level of protection.

When evaluating the risk of eavesdropping attacks, it is important to consider the lifetime of bundles on a DTN. Depending on the network, bundles may persist for days or even years. Long-lived bundles imply that the data exists in the network for a longer period of time and, thus, there may be more opportunities to capture those bundles. Additionally, bundles that are long-lived imply that the information stored within them may remain relevant and sensitive for long enough that, once captured, there is sufficient time to crack encryption associated with the bundle. If a bundle does persist on the network for years and the cipher suite used for a BCB provides inadequate protection, Mallory may be able to recover the protected data either before that bundle reaches its intended destination or before the information in the bundle is no longer considered sensitive.

8.2.2. Modification Attacks

As a node participating in the DTN between Alice and Bob, Mallory will also be able to modify the received bundle, including non-BPSec data such as the primary block, payload blocks, or block processing control flags as defined in [I-D.ietf-dtn-bpbis]. Mallory will be able to undertake activities which include modification of data within the blocks, replacement of blocks, addition of blocks, or removal of blocks. Within BPSec, both the BIB and BCB provide integrity protection mechanisms to detect or prevent data manipulation attempts by Mallory.

The BIB provides that protection to another block which is its security target. The cryptographic mechanisms used to generate the BIB should be strong against collision attacks and Mallory should not have access to the cryptographic material used by the originating node to generate the BIB (e.g., K_A). If both of these conditions are true, Mallory will be unable to modify the security target or the BIB and lead Bob to validate the security target as originating from Alice.

Since BPSec security operations are implemented by placing blocks in a bundle, there is no in-band mechanism for detecting or correcting certain cases where Mallory removes blocks from a bundle. If Mallory removes a BCB, but keeps the security target, the security target remains encrypted and there is a possibility that there may no longer be sufficient information to decrypt the block at its destination. If Mallory removes both a BCB (or BIB) and its security target there is no evidence left in the bundle of the security operation. Similarly, if Mallory removes the BIB but not the security target there is no evidence left in the bundle of the security operation. In each of these cases, the implementation of BPSec must be combined with policy configuration at endpoints in the network which describe the expected and required security operations that must be applied on transmission and are expected to be present on receipt. This or other similar out-of-band information is required to correct for removal of security information in the bundle.

A limitation of the BIB may exist within the implementation of BIB validation at the destination node. If Mallory is a legitimate node within the DTN, the BIB generated by Alice with K_A can be replaced with a new BIB generated with K_M and forwarded to Bob. If Bob is only validating that the BIB was generated by a legitimate user, Bob will acknowledge the message as originating from Mallory instead of Alice. In order to provide verifiable integrity checks, both a BIB and BCB should be used and the BCB should require an IND-CCA2 encryption scheme. Such an encryption scheme will guard against signature substitution attempts by Mallory. In this case, Alice creates a BIB with the protected data block as the security target and then creates a BCB with both the BIB and protected data block as its security targets.

8.2.3. Topology Attacks

If Mallory is in a MITM position within the DTN, she is able to influence how any bundles that come to her may pass through the network. Upon receiving and processing a bundle that must be routed elsewhere in the network, Mallory has three options as to how to proceed: not forward the bundle, forward the bundle as intended, or forward the bundle to one or more specific nodes within the network.

Attacks that involve re-routing the packets throughout the network are essentially a special case of the modification attacks described in this section where the attacker is modifying fields within the primary block of the bundle. Given that BPSec cannot encrypt the contents of the primary block, alternate methods must be used to prevent this situation. These methods may include requiring BIBs for primary blocks, using encapsulation, or otherwise strategically manipulating primary block data. The specifics of any such mitigation technique are specific to the implementation of the deploying network and outside of the scope of this document.

Furthermore, routing rules and policies may be useful in enforcing particular traffic flows to prevent topology attacks. While these rules and policies may utilize some features provided by BPSec, their definition is beyond the scope of this specification.

8.2.4. Message Injection

Mallory is also able to generate new bundles and transmit them into the DTN at will. These bundles may either be copies or slight modifications of previously-observed bundles (i.e., a replay attack) or entirely new bundles generated based on the Bundle Protocol, BPSec, or other bundle-related protocols. With these attacks Mallory's objectives may vary, but may be targeting either the bundle protocol or application-layer protocols conveyed by the bundle protocol.

BPSec relies on cipher suite capabilities to prevent replay or forged message attacks. A BCB used with appropriate cryptographic mechanisms (e.g., a counter-based cipher mode) may provide replay protection under certain circumstances. Alternatively, application data itself may be augmented to include mechanisms to assert data uniqueness and then protected with a BIB, a BCB, or both along with other block data. In such a case, the receiving node would be able to validate the uniqueness of the data.

9. Cipher Suite Authorship Considerations

Cipher suite developers or implementers should consider the diverse performance and conditions of networks on which the Bundle Protocol (and therefore BPSec) will operate. Specifically, the delay and capacity of delay-tolerant networks can vary substantially. Cipher suite developers should consider these conditions to better describe the conditions when those suites will operate or exhibit vulnerability, and selection of these suites for implementation should be made with consideration to the reality. There are key differences that may limit the opportunity to leverage existing cipher suites and technologies that have been developed for use in traditional, more reliable networks:

- Data Lifetime: Depending on the application environment, bundles may persist on the network for extended periods of time, perhaps even years. Cryptographic algorithms should be selected to ensure protection of data against attacks for a length of time reasonable for the application.

- One-Way Traffic: Depending on the application environment, it is possible that only a one-way connection may exist between two endpoints, or if a two-way connection does exist, the round-trip time may be extremely large. This may limit the utility of session key generation mechanisms, such as Diffie-Hellman, as a two-way handshake may not be feasible or reliable.

- Opportunistic Access: Depending on the application environment, a given endpoint may not be guaranteed to be accessible within a certain amount of time. This may make asymmetric cryptographic architectures which rely on a key distribution center or other trust center impractical under certain conditions.

When developing new cipher suites for use with BPSec, the following information SHOULD be considered for inclusion in these specifications.

- Cipher Suite Parameters. Cipher suites MUST define their parameter ids, the data types of those parameters, and their CBOR encoding.

- Security Results. Cipher suites MUST define their security result ids, the data types of those results, and their CBOR encoding.

- New Canonicalizations. Cipher suites may define new canonicalization algorithms as necessary.

- Cipher-Text Size. Cipher suites MUST state whether they generate cipher-text (to include any included authentication information) that is of a different size than the input plain-text.

  If a cipher suite does not wish to alter the size of the plain-text, it should consider the following.

  * Place overflow bytes, authentication signatures, and any additional authenticated data in security result fields rather than in the cipher-text itself.

  * Pad the cipher-text in cases where the cipher-text is smaller than the plain-text.

- If a BCB cannot alter the size of the security target then differences in the size of the cipher-text and plain-text MUST be handled in the following way. If the cipher-text is shorter in length than the plain-text, padding MUST be used in accordance with the cipher suite policy. If the cipher-text is larger than the plain-text, overflow bytes MUST be placed in overflow parameters in the Security Result field. Any additional authentication information can be treated either as overflow cipher-text or represented separately in the BCB in a security result field, in accordance with cipher suite documentation and security policy.

10. Defining Other Security Blocks

Other security blocks (OSBs) may be defined and used in addition to the security blocks identified in this specification. Both the usage of BIB, BCB, and any future OSBs can co-exist within a bundle and can be considered in conformance with BPSec if each of the following requirements are met by any future identified security blocks.

- Other security blocks (OSBs) MUST NOT reuse any enumerations identified in this specification, to include the block type codes for BIB and BCB.

- An OSB definition MUST state whether it can be the target of a BIB or a BCB. The definition MUST also state whether the OSB can target a BIB or a BCB.

- An OSB definition MUST provide a deterministic processing order in the event that a bundle is received containing BIBs, BCBs, and OSBs. This processing order MUST NOT alter the BIB and BCB processing orders identified in this specification.

- An OSB definition MUST provide a canonicalization algorithm if the default non-primary-block canonicalization algorithm cannot be used to generate a deterministic input for a cipher suite. This requirement can be waived if the OSB is defined so as to never be the security target of a BIB or a BCB.

- An OSB definition MUST NOT require any behavior of a BPSEC-BPA that is in conflict with the behavior identified in this specification. In particular, the security processing requirements imposed by this specification must be consistent across all BPSEC-BPAs in a network.

- The behavior of an OSB when dealing with fragmentation must be specified and MUST NOT lead to ambiguous processing states. In particular, an OSB definition should address how to receive and process an OSB in a bundle fragment that may or may not also contain its security target. An OSB definition should also address whether an OSB may be added to a bundle marked as a fragment.

Additionally, policy considerations for the management, monitoring, and configuration associated with blocks SHOULD be included in any OSB definition.

NOTE: The burden of showing compliance with processing rules is placed upon the standards defining new security blocks and the identification of such blocks shall not, alone, require maintenance of this specification.

11. IANA Considerations

A registry of cipher suite identifiers will be required.

11.1. Bundle Block Types

This specification allocates three block types from the existing "Bundle Block Types" registry defined in [RFC6255].

Additional Entries for the Bundle Block-Type Codes Registry:

Value	Description	Reference
TBD	Security Association Block	This document
TBD	Block Integrity Block	This document
TBD	Block Confidentiality Block	This document

Table 1

12. References

12.1. Normative References

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

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

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

[RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, DOI 10.17487/RFC6255, May 2011, <https://www.rfc-editor.org/info/rfc6255>.

12.2. Informative References

[I-D.birrane-dtn-sbsp]
Birrane, E., Pierce-Mayer, J., and D. Iannicca, "Streamlined Bundle Security Protocol Specification", draft-birrane-dtn-sbsp-01 (work in progress), October 2015.

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

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

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

Appendix A. Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this security specification: Scott Burleigh of the Jet Propulsion Laboratory, Amy Alford and Angela Hennessy of the Laboratory for Telecommunications Sciences, and Angela Dalton and Cherita Corbett of the Johns Hopkins University Applied Physics Laboratory.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Kenneth McKeever
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 2237
Email: Ken.McKeever@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: June 4, 2021

E. Birrane
K. McKeever
JHU/APL

December 1, 2020

Bundle Protocol Security Specification
draft-ietf-dtn-bpsec-25

Abstract

This document defines a security protocol providing data integrity and confidentiality services for the Bundle Protocol.

Table of Contents

1. Introduction

This document defines security features for the Bundle Protocol (BP) [I-D.ietf-dtn-bpbis] and is intended for use in Delay Tolerant Networks (DTNs) to provide security services between a security source and a security acceptor. When the security source is the bundle source and when the security acceptor is the bundle destination, the security service provides end-to-end protection.

The Bundle Protocol specification [I-D.ietf-dtn-bpbis] defines DTN as referring to "a networking architecture providing communications in and/or through highly stressed environments" where "BP may be viewed as sitting at the application layer of some number of constituent networks, forming a store-carry-forward overlay network". The term "stressed" environment refers to multiple challenging conditions including intermittent connectivity, large and/or variable delays, asymmetric data rates, and high bit error rates.

It should be presumed that the BP will be deployed such that the network cannot be trusted, posing the usual security challenges related to confidentiality and integrity. However, the stressed nature of the BP operating environment imposes unique conditions where usual transport security mechanisms may not be sufficient. For example, the store-carry-forward nature of the network may require protecting data at rest, preventing unauthorized consumption of critical resources such as storage space, and operating without regular contact with a centralized security oracle (such as a certificate authority).

An end-to-end security service is needed that operates in all of the environments where the BP operates.

1.1. Supported Security Services

BPSec provides integrity and confidentiality services for BP bundles, as defined in this section.

Integrity services ensure that changes to target data within a bundle can be discovered. Data changes may be caused by processing errors, environmental conditions, or intentional manipulation. In the context of BPSec, integrity services apply to plain text in the bundle.

Confidentiality services ensure that target data is unintelligible to nodes in the DTN, except for authorized nodes possessing special information. This generally means producing cipher text from plain text and generating authentication information for that cipher text. Confidentiality, in this context, applies to the contents of target data and does not extend to hiding the fact that confidentiality exists in the bundle.

NOTE: Hop-by-hop authentication is NOT a supported security service in this specification, for two reasons.

1. The term "hop-by-hop" is ambiguous in a BP overlay, as nodes that are adjacent in the overlay may not be adjacent in physical connectivity. This condition is difficult or impossible to detect and therefore hop-by-hop authentication is difficult or impossible to enforce.

2. Hop-by-hop authentication cannot be deployed in a network if adjacent nodes in the network have incompatible security capabilities.

1.2. Specification Scope

This document defines the security services provided by the BPSec. This includes the data specification for representing these services as BP extension blocks, and the rules for adding, removing, and processing these blocks at various points during the bundle's traversal of the DTN.

BPSec addresses only the security of data traveling over the DTN, not the underlying DTN itself. Furthermore, while the BPSec protocol can provide security-at-rest in a store-carry-forward network, it does not address threats which share computing resources with the DTN and/ or BPSec software implementations. These threats may be malicious software or compromised libraries which intend to intercept data or recover cryptographic material. Here, it is the responsibility of the BPSec implementer to ensure that any cryptographic material, including shared secret or private keys, is protected against access within both memory and storage devices.

Completely trusted networks are extremely uncommon. Amongst untrusted networks, different networking conditions and operational considerations require varying strengths of security mechanism. Mandating a single security context may result in too much security for some networks and too little security in others. It is expected that separate documents define different security contexts for use in different networks. A set of default security contexts are defined in ([I-D.ietf-dtn-bpsec-interop-sc]) and provide basic security services for interoperability testing and for operational use on the terrestrial Internet.

This specification addresses neither the fitness of externally-defined cryptographic methods nor the security of their implementation.

This specification does not address the implementation of security policy and does not provide a security policy for the BPSec. Similar to cipher suites, security policies are based on the nature and capabilities of individual networks and network operational concepts. This specification does provide policy considerations when building a security policy.

With the exception of the Bundle Protocol, this specification does not address how to combine the BPSec security blocks with other protocols, other BP extension blocks, or other best practices to achieve security in any particular network implementation.

1.3. Related Documents

This document is best read and understood within the context of the following other DTN documents:

"Delay-Tolerant Networking Architecture" [RFC4838] defines the architecture for DTNs and identifies certain security assumptions made by existing Internet protocols that are not valid in a DTN.

The Bundle Protocol [I-D.ietf-dtn-bpbis] defines the format and processing of bundles, defines the extension block format used to represent BPSec security blocks, and defines the canonical block structure used by this specification.

The Concise Binary Object Representation (CBOR) format [RFC7049] defines a data format that allows for small code size, fairly small message size, and extensibility without version negotiation. The block-specific-data associated with BPSec security blocks are encoded in this data format.

The Bundle Security Protocol [RFC6257] and Streamlined Bundle Security Protocol [I-D.birrane-dtn-sbsp] documents introduced the concepts of using BP extension blocks for security services in a DTN. The BPSec is a continuation and refinement of these documents.

1.4. Terminology

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

This section defines terminology either unique to the BPSec or otherwise necessary for understanding the concepts defined in this specification.

- Bundle Destination - the node which receives a bundle and delivers the payload of the bundle to an application. Also, the Node ID of the Bundle Protocol Agent (BPA) receiving the bundle. The bundle destination acts as the security acceptor for every security target in every security block in every bundle it receives.

- Bundle Source - the node which originates a bundle. Also, the Node ID of the BPA originating the bundle.

- Cipher Suite - a set of one or more algorithms providing integrity and/or confidentiality services. Cipher suites may define user parameters (e.g. secret keys to use) but do not provide values for those parameters.

- Forwarder - any node that transmits a bundle in the DTN. Also, the Node ID of the BPA that sent the bundle on its most recent hop.

- Intermediate Receiver, Waypoint, or Next Hop - any node that receives a bundle from a Forwarder that is not the Bundle Destination. Also, the Node ID of the BPA at any such node.

- Path - the ordered sequence of nodes through which a bundle passes on its way from Source to Destination. The path is not necessarily known in advance by the bundle or any BPAs in the DTN.

- Security Acceptor - a bundle node that processes and dispositions one or more security blocks in a bundle. Also, the Node ID of that node.

- Security Block - a BPSec extension block in a bundle.

- Security Context - the set of assumptions, algorithms, configurations and policies used to implement security services.

- Security Operation - the application of a given security service to a security target, notated as OP(security service, security target). For example, OP(bcb-confidentiality, payload). Every security operation in a bundle MUST be unique, meaning that a given security service can only be applied to a security target once in a bundle. A security operation is implemented by a security block.

- Security Service - a process that gives some protection to a security target. For example, this specification defines security services for plain text integrity (bib-integrity), and authenticated plain text confidentiality with additional authenticated data (bcb-confidentiality).

- Security Source - a bundle node that adds a security block to a bundle. Also, the Node ID of that node.

- Security Target - the block within a bundle that receives a security service as part of a security operation.

2. Design Decisions

The application of security services in a DTN is a complex endeavor that must consider physical properties of the network (such as connectivity and propagation times), policies at each node, application security requirements, and current and future threat environments. This section identifies those desirable properties that guide design decisions for this specification and are necessary for understanding the format and behavior of the BPSec protocol.

2.1. Block-Level Granularity

Security services within this specification must allow different blocks within a bundle to have different security services applied to them.

Blocks within a bundle represent different types of information. The primary block contains identification and routing information. The payload block carries application data. Extension blocks carry a variety of data that may augment or annotate the payload, or otherwise provide information necessary for the proper processing of a bundle along a path. Therefore, applying a single level and type of security across an entire bundle fails to recognize that blocks in a bundle represent different types of information with different security needs.

For example, a payload block might be encrypted to protect its contents and an extension block containing summary information related to the payload might be integrity signed but unencrypted to provide waypoints access to payload-related data without providing access to the payload.

2.2. Multiple Security Sources

A bundle can have multiple security blocks and these blocks can have different security sources. BPSec implementations MUST NOT assume that all blocks in a bundle have the same security operations applied to them.

The Bundle Protocol allows extension blocks to be added to a bundle at any time during its existence in the DTN. When a waypoint adds a new extension block to a bundle, that extension block MAY have security services applied to it by that waypoint. Similarly, a waypoint MAY add a security service to an existing block, consistent with its security policy.

When a waypoint adds a security service to the bundle, the waypoint is the security source for that service. The security block(s) which represent that service in the bundle may need to record this security source as the bundle destination might need this information for processing.

For example, a bundle source may choose to apply an integrity service to its plain text payload. Later a waypoint node, representing a gateway to another portion of the DTN, may receive the bundle and choose to apply a confidentiality service. In this case, the integrity security source is the bundle source and the confidentiality security source is the waypoint node.

In cases where the security source and security acceptor are not the bundle source and bundle destination, it is possible that the bundle will reach the bundle destination prior to reaching a security acceptor. In cases where this may be a practical problem, it is recommended that solutions such as bundle encapsulation can be used to ensure that a bundle be delivered to a security acceptor prior to being delivered to the bundle destination. Generally, if a bundle reaches a waypoint that has the appropriate configuration and policy to act as a security acceptor for a security service in the bundle, then the waypoint should act as that security acceptor.

2.3. Mixed Security Policy

The security policy enforced by nodes in the DTN may differ.

Some waypoints will have security policies that require evaluating security services even if they are not the bundle destination or the final intended acceptor of the service. For example, a waypoint could choose to verify an integrity service even though the waypoint is not the bundle destination and the integrity service will be needed by other nodes along the bundle's path.

Some waypoints will determine, through policy, that they are the intended recipient of the security service and terminate the security service in the bundle. For example, a gateway node could determine that, even though it is not the destination of the bundle, it should verify and remove a particular integrity service or attempt to decrypt a confidentiality service, before forwarding the bundle along its path.

Some waypoints could understand security blocks but refuse to process them unless they are the bundle destination.

2.4. User-Defined Security Contexts

A security context is the union of security algorithms (cipher suites), policies associated with the use of those algorithms, and configuration values. Different contexts may specify different algorithms, different polices, or different configuration values used in the implementation of their security services. BPSec provides a mechanism to define security contexts. Users may select from registered security contexts and customize those contexts through security context parameters.

For example, some users might prefer a SHA2 hash function for integrity whereas other users might prefer a SHA3 hash function. Providing either separate security contexts or a single, parameterized security context allows users flexibility in applying the desired cipher suite, policy, and configuration when populating a security block.

2.5. Deterministic Processing

Whenever a node determines that it must process more than one security block in a received bundle (either because the policy at a waypoint states that it should process security blocks or because the node is the bundle destination) the order in which security blocks are processed must be deterministic. All nodes must impose this same deterministic processing order for all security blocks. This specification provides determinism in the application and evaluation of security services, even when doing so results in a loss of flexibility.

3. Security Blocks

3.1. Block Definitions

This specification defines two types of security block: the Block Integrity Block (BIB) and the Block Confidentiality Block (BCB).

The BIB is used to ensure the integrity of its plain text security target(s). The integrity information in the BIB MAY be verified by any node along the bundle path from the BIB security source to the bundle destination. Waypoints add or remove BIBs from bundles in accordance with their security policy. BIBs are never used for integrity protection of the cipher text provided by a BCB. Because security policy at BPSec nodes may differ regarding integrity verification, BIBs do not guarantee hop-by-hop authentication, as discussed in Section 1.1.

The BCB indicates that the security target(s) have been encrypted at the BCB security source in order to protect their content while in transit. The BCB is decrypted by security acceptor nodes in the network, up to and including the bundle destination, as a matter of security policy. BCBs additionally provide integrity protection mechanisms for the cipher text they generate.

3.2. Uniqueness

Security operations in a bundle MUST be unique; the same security service MUST NOT be applied to a security target more than once in a bundle. Since a security operation is represented by a security block, this means that multiple security blocks of the same type cannot share the same security targets. A new security block MUST NOT be added to a bundle if a pre-existing security block of the same type is already defined for the security target of the new security block.

This uniqueness requirement ensures that there is no ambiguity related to the order in which security blocks are processed or how security policy can be specified to require certain security services be present in a bundle.

Using the notation OP(service, target), several examples illustrate this uniqueness requirement.

- o Signing the payload twice: The two operations OP(bib-integrity, payload) and OP(bib-integrity, payload) are redundant and MUST NOT both be present in the same bundle at the same time.

- o Signing different blocks: The two operations OP(bib-integrity, payload) and OP(bib-integrity, extension_block_1) are not redundant and both may be present in the same bundle at the same time. Similarly, the two operations OP(bib-integrity, extension_block_1) and OP(bib-integrity, extension_block_2) are also not redundant and may both be present in the bundle at the same time.

- o Different Services on same block: The two operations OP(bib-integrity, payload) and OP(bcb-confidentiality, payload) are not inherently redundant and may both be present in the bundle at the same time, pursuant to other processing rules in this specification.

- o Different services from different block types: The notation OP(service, target) refers specifically to a security block, as the security block is the embodiment of a security service applied to a security target in a bundle. Were some Other Security Block (OSB) to be defined providing an integrity service, then the operations OP(bib-integrity, target) and OP(osb-integrity, target) MAY both be present in the same bundle if so allowed by the definition of the OSB, as discussed in Section 10.

NOTES:

A security block may be removed from a bundle as part of security processing at a waypoint node with a new security block being added to the bundle by that node. In this case, conflicting security blocks never co-exist in the bundle at the same time and the uniqueness requirement is not violated.

A cipher text integrity mechanism (such as associated authenticated data) calculated by a cipher suite and transported in a BCB is considered part of the confidentiality service and, therefore, unique from the plain text integrity service provided by a BIB.

The security blocks defined in this specification (BIB and BCB) are designed with the intention that the BPA adding these blocks is the authoritative source of the security service. If a BPA adds a BIB on a security target, then the BIB is expected to be the authoritative source of integrity for that security target. If a BPA adds a BCB to a security target, then the BCB is expected to be the authoritative source of confidentiality for that security target. More complex scenarios, such as having multiple nodes in a network sign the same security target, can be accommodated using the definition of custom security contexts (Section 9) and/or the definition of other security blocks (Section 10).

3.3. Target Multiplicity

A single security block MAY represent multiple security operations as a way of reducing the overall number of security blocks present in a bundle. In these circumstances, reducing the number of security blocks in the bundle reduces the amount of redundant information in the bundle.

A set of security operations can be represented by a single security block when all of the following conditions are true.

- The security operations apply the same security service. For example, they are all integrity operations or all confidentiality operations.

- The security context parameters for the security operations are identical.

- The security source for the security operations is the same, meaning the set of operations are being added by the same node.

- No security operations have the same security target, as that would violate the need for security operations to be unique.

- None of the security operations conflict with security operations already present in the bundle.

When representing multiple security operations in a single security block, the information that is common across all operations is represented once in the security block, and the information which is different (e.g., the security targets) are represented individually.

It is RECOMMENDED that if a node processes any security operation in a security block that it process all security operations in the security block. This allows security sources to assert that the set of security operations in a security block are expected to be processed by the same security acceptor. However, the determination of whether a node actually is a security acceptor or not is a matter of the policy of the node itself. In cases where a receiving node determines that it is the security acceptor of only a subset of the security operations in a security block, the node may choose to only process that subset of security operations.

3.4. Target Identification

A security target is a block in the bundle to which a security service applies. This target must be uniquely and unambiguously identifiable when processing a security block. The definition of the extension block header from [I-D.ietf-dtn-bpbis] provides a "Block Number" field suitable for this purpose. Therefore, a security target in a security block MUST be represented as the Block Number of the target block.

3.5. Block Representation

Each security block uses the Canonical Bundle Block Format as defined in [I-D.ietf-dtn-bpbis]. That is, each security block is comprised of the following elements:

- block type code
- block number
- block processing control flags
- CRC type
- block-type-specific-data
- CRC field (if present)

Security-specific information for a security block is captured in the block-type-specific-data field.

3.6. Abstract Security Block

The structure of the security-specific portions of a security block is identical for both the BIB and BCB Block Types. Therefore, this section defines an Abstract Security Block (ASB) data structure and discusses the definition, processing, and other constraints for using this structure. An ASB is never directly instantiated within a bundle, it is only a mechanism for discussing the common aspects of BIB and BCB security blocks.

The fields of the ASB SHALL be as follows, listed in the order in which they must appear.

Security Targets:
: This field identifies the block(s) targeted by the security operation(s) represented by this security block. Each target block is represented by its unique Block Number. This field

SHALL be represented by a CBOR array of data items. Each target within this CBOR array SHALL be represented by a CBOR unsigned integer. This array MUST have at least 1 entry and each entry MUST represent the Block Number of a block that exists in the bundle. There MUST NOT be duplicate entries in this array. The order of elements in this list has no semantic meaning outside of the context of this block. Within the block, the ordering of targets must match the ordering of results associated with these targets.

Security Context Id:
This field identifies the security context used to implement the security service represented by this block and applied to each security target. This field SHALL be represented by a CBOR unsigned integer. The values for this Id should come from the registry defined in Section 11.3

Security Context Flags:
This field identifies which optional fields are present in the security block. This field SHALL be represented as a CBOR unsigned integer whose contents shall be interpreted as a bit field. Each bit in this bit field indicates the presence (bit set to 1) or absence (bit set to 0) of optional data in the security block. The association of bits to security block data is defined as follows.

Bit 0 (the least-significant bit, 0x01): Security Context Parameters Present Flag.

Bit 1 (0x02): Security Source Present Flag.

Bit >1 Reserved

Implementations MUST set reserved bits to 0 when writing this field and MUST ignore the values of reserved bits when reading this field. For unreserved bits, a value of 1 indicates that the associated security block field MUST be included in the security block. A value of 0 indicates that the associated security block field MUST NOT be in the security block.

Security Source (Optional):
This field identifies the Endpoint that inserted the security block in the bundle. If the security source field is not present then the source MUST be inferred from other information, such as the bundle source, previous hop, or other values defined by security policy. This field SHALL be represented by a CBOR array in accordance with

[I-D.ietf-dtn-bpbis] rules for representing Endpoint Identifiers (EIDs).

Security Context Parameters (Optional):
This field captures one or more security context parameters that should be used when processing the security service described by this security block. This field SHALL be represented by a CBOR array. Each entry in this array is a single security context parameter. A single parameter SHALL also be represented as a CBOR array comprising a 2-tuple of the id and value of the parameter, as follows.

* Parameter Id. This field identifies which parameter is being specified. This field SHALL be represented as a CBOR unsigned integer. Parameter Ids are selected as described in Section 3.10.

* Parameter Value. This field captures the value associated with this parameter. This field SHALL be represented by the applicable CBOR representation of the parameter, in accordance with Section 3.10.

The logical layout of the parameters array is illustrated in Figure 1.

```
+----------------+----------------+     +----------------+
|  Parameter 1   |  Parameter 2   | ... |  Parameter N   |
+------+---------+------+---------+     +------+---------+
|  Id  |  Value  |  Id  |  Value  |     |  Id  |  Value  |
+------+---------+------+---------+     +------+---------+
```

Figure 1: Security Context Parameters

Security Results:
This field captures the results of applying a security service to the security targets of the security block. This field SHALL be represented as a CBOR array of target results. Each entry in this array represents the set of security results for a specific security target. The target results MUST be ordered identically to the Security Targets field of the security block. This means that the first set of target results in this array corresponds to the first entry in the Security Targets field of the security block, and so on. There MUST be one entry in this array for each entry in the Security Targets field of the security block.

The set of security results for a target is also represented as a CBOR array of individual results. An individual result is represented as a 2-tuple of a result id and a result value, defined as follows.

* Result Id. This field identifies which security result is being specified. Some security results capture the primary output of a cipher suite. Other security results contain additional annotative information from cipher suite processing. This field SHALL be represented as a CBOR unsigned integer. Security result Ids will be as specified in Section 3.10.

* Result Value. This field captures the value associated with the result. This field SHALL be represented by the applicable CBOR representation of the result value, in accordance with Section 3.10.

The logical layout of the security results array is illustrated in Figure 2. In this figure there are N security targets for this security block. The first security target contains M results and the Nth security target contains K results.

```
+------------------------------+     +------------------------------+
|            Target 1          |     |           Target N           |
+------------+----+------------+     +------------------------------+
| Result 1   |    | Result M   | ... | Result 1    |    | Result K  |
+----+-------+ .. +----+-------+     +----+-------+ .. +----+-------+
| Id | Value |    | Id | Value |     | Id | Value |    | Id | Value |
+----+-------+    +----+-------+     +----+-------+    +----+-------+
```

Figure 2: Security Results

3.7. Block Integrity Block

A BIB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The block-type-specific-data field follows the structure of the ASB.

A security target listed in the Security Targets field MUST NOT reference a security block defined in this specification (e.g., a BIB or a BCB).

The Security Context MUST utilize an authentication mechanism or an error detection mechanism.

The EID of the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source MAY be specified as part of security context parameters described in Section 3.10.

Notes:

- Designers SHOULD carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.

- Since OP(bib-integrity, target) is allowed only once in a bundle per target, it is RECOMMENDED that users wishing to support multiple integrity mechanisms for the same target define a multi-result security context. Such a context could generate multiple security results for the same security target using different integrity-protection mechanisms or different configurations for the same integrity-protection mechanism.

- A BIB is used to verify the plain text integrity of its security target. However, a single BIB MAY include security results for blocks other than its security target when doing so establishes a needed relationship between the BIB security target and other blocks in the bundle (such as the primary block).

- Security information MAY be checked at any hop on the way to the bundle destination that has access to the required keying information, in accordance with Section 3.9.

3.8. Block Confidentiality Block

A BCB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The Block Processing Control flags value can be set to whatever values are required by local policy with the following exceptions. BCB blocks MUST have the "block must be replicated in every fragment" flag set if one of the targets is the payload block. Having that BCB in each fragment indicates to a receiving node that the payload portion of each fragment represents cipher text. BCB blocks MUST NOT have the "block must be removed from bundle if it can’t be processed" flag set. Removing a BCB from a bundle without decrypting its security targets removes information from the bundle necessary for their later decryption.

The block-type-specific-data fields follow the structure of the ASB.

A security target listed in the Security Targets field can reference the payload block, a non-security extension block, or a BIB. A BCB MUST NOT include another BCB as a security target. A BCB MUST NOT target the primary block. A BCB MUST NOT target a BIB block unless it shares a security target with that BIB block.

Any Security Context used by a BCB MUST utilize a confidentiality cipher that provides authenticated encryption with associated data (AEAD).

Additional information created by a cipher suite (such as an authentication tag) can be placed either in a security result field or in the generated cipher text. The determination of where to place this information is a function of the cipher suite and security context used.

The EID of the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source MAY be specified as part of security context parameters described in Section 3.10.

The BCB modifies the contents of its security target(s). When a BCB is applied, the security target body data are encrypted "in-place". Following encryption, the security target block-type-specific-data field contains cipher text, not plain text.

Notes:

- It is RECOMMENDED that designers carefully consider the effect of setting flags that delete the bundle in the event that this block cannot be processed.

- The BCB block processing control flags can be set independently from the processing control flags of the security target(s). The setting of such flags should be an implementation/policy decision for the encrypting node.

3.9. Block Interactions

The security block types defined in this specification are designed to be as independent as possible. However, there are some cases where security blocks may share a security target creating processing dependencies.

If a security target of a BCB is also a security target of a BIB, an undesirable condition occurs where a waypoint would be unable to validate the BIB because one of its security target's contents have been encrypted by a BCB. To address this situation the following processing rules MUST be followed.

- o When adding a BCB to a bundle, if some (or all) of the security targets of the BCB also match all of the security targets of an existing BIB, then the existing BIB MUST also be encrypted. This can be accomplished by either adding a new BCB that targets the existing BIB, or by adding the BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.

- o When adding a BCB to a bundle, if some (or all) of the security targets of the BCB match some (but not all) of the security targets of a BIB then that BIB MUST be altered in the following way. Any security results in the BIB associated with the BCB security targets MUST be removed from the BIB and placed in a new BIB. This newly created BIB MUST then be encrypted. The encryption of the new BIB can be accomplished by either adding a new BCB that targets the new BIB, or by adding the new BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.

- o A BIB MUST NOT be added for a security target that is already the security target of a BCB as this would cause ambiguity in block processing order.

- o A BIB integrity value MUST NOT be checked if the BIB is the security target of an existing BCB. In this case, the BIB data is encrypted.

- o A BIB integrity value MUST NOT be checked if the security target associated with that value is also the security target of a BCB. In such a case, the security target data contains cipher text as it has been encrypted.

- o As mentioned in Section 3.7, a BIB MUST NOT have a BCB as its security target.

These restrictions on block interactions impose a necessary ordering when applying security operations within a bundle. Specifically, for a given security target, BIBs MUST be added before BCBs. This ordering MUST be preserved in cases where the current BPA is adding all of the security blocks for the bundle or whether the BPA is a waypoint adding new security blocks to a bundle that already contains security blocks.

In cases where a security source wishes to calculate both a plain text integrity mechanism and encrypt a security target, a BCB with a security context that generates an integrity-protection mechanism as one or more additional security results MUST be used instead of adding both a BIB and then a BCB for the security target at the security source.

3.10. Parameter and Result Identification

Each security context MUST define its own context parameters and results. Each defined parameter and result is represented as the tuple of an identifier and a value. Identifiers are always represented as a CBOR unsigned integer. The CBOR encoding of values is as defined by the security context specification.

Identifiers MUST be unique for a given security context but do not need to be unique amongst all security contexts.

An example of a security context can be found at [I-D.ietf-dtn-bpsec-interop-sc].

3.11. BSP Block Examples

This section provides two examples of BPSec blocks applied to a bundle. In the first example, a single node adds several security operations to a bundle. In the second example, a waypoint node received the bundle created in the first example and adds additional security operations. In both examples, the first column represents blocks within a bundle and the second column represents the Block Number for the block, using the terminology B1...Bn for the purpose of illustration.

3.11.1. Example 1: Constructing a Bundle with Security

In this example a bundle has four non-security-related blocks: the primary block (B1), two extension blocks (B4,B5), and a payload block (B6). The bundle source wishes to provide an integrity signature of the plain text associated with the primary block, the second extension block, and the payload. The bundle source also wishes to provide confidentiality for the first extension block. The resultant bundle is illustrated in Figure 3 and the security actions are described below.

```
                  Block in Bundle                ID
+==========================================+====+
|              Primary Block               | B1 |
+------------------------------------------+----+
|                   BIB                    | B2 |
|   OP(bib-integrity, targets=B1, B5, B6)  |    |
+------------------------------------------+----+
|                   BCB                    | B3 |
|    OP(bcb-confidentiality, target=B4)    |    |
+------------------------------------------+----+
|       Extension Block (encrypted)        | B4 |
+------------------------------------------+----+
|              Extension Block             | B5 |
+------------------------------------------+----+
|               Payload Block              | B6 |
+------------------------------------------+----+
```

Figure 3: Security at Bundle Creation

The following security actions were applied to this bundle at its time of creation.

o An integrity signature applied to the canonical form of the primary block (B1), the canonical form of the block-type-specific-data field of the second extension block (B5) and the canonical form of the payload block (B6). This is accomplished by a single BIB (B2) with multiple targets. A single BIB is used in this case because all three targets share a security source, security context, and security context parameters. Had this not been the case, multiple BIBs could have been added instead.

o Confidentiality for the first extension block (B4). This is accomplished by a BCB (B3). Once applied, the block-type-specific-data field of extension block B4 is encrypted. The BCB MUST hold an authentication tag for the cipher text either in the cipher text that now populates the first extension block or as a security result in the BCB itself, depending on which security context is used to form the BCB. A plain text integrity signature may also exist as a security result in the BCB if one is provided by the selected confidentiality security context.

3.11.2. Example 2: Adding More Security At A New Node

Consider that the bundle as it is illustrated in Figure 3 is now received by a waypoint node that wishes to encrypt the second extension block and the bundle payload. The waypoint security policy is to allow existing BIBs for these blocks to persist, as they may be required as part of the security policy at the bundle destination.

The resultant bundle is illustrated in Figure 4 and the security actions are described below. Note that block IDs provided here are ordered solely for the purpose of this example and not meant to impose an ordering for block creation. The ordering of blocks added to a bundle MUST always be in compliance with [I-D.ietf-dtn-bpbis].

```
                   Block in Bundle                 ID
+=========================================+====+
|              Primary Block              | B1 |
+-----------------------------------------+----+
|                   BIB                   | B2 |
|      OP(bib-integrity, targets=B1)      |    |
+-----------------------------------------+----+
|                   BIB (encrypted)       | B7 |
|      OP(bib-integrity, targets=B5, B6)  |    |
+-----------------------------------------+----+
|                   BCB                   | B8 |
| OP(bcb-confidentiality,targets=B5,B6,B7)|    |
+-----------------------------------------+----+
|                   BCB                   | B3 |
|   OP(bcb-confidentiality, target=B4)    |    |
+-----------------------------------------+----+
|       Extension Block (encrypted)       | B4 |
+-----------------------------------------+----+
|       Extension Block (encrypted)       | B5 |
+-----------------------------------------+----+
|         Payload Block (encrypted)       | B6 |
+-----------------------------------------+----+
```

Figure 4: Security At Bundle Forwarding

The following security actions were applied to this bundle prior to its forwarding from the waypoint node.

- o Since the waypoint node wishes to encrypt the block-type-specific-data field of blocks B5 and B6, it MUST also encrypt the block-type-specific-data field of the BIBs providing plain text integrity over those blocks. However, BIB B2 could not be encrypted in its entirety because it also held a signature for the primary block (B1). Therefore, a new BIB (B7) is created and security results associated with B5 and B6 are moved out of BIB B2 and into BIB B7.

- o Now that there is no longer confusion of which plain text integrity signatures must be encrypted, a BCB is added to the bundle with the security targets being the second extension block (B5) and the payload (B6) as well as the newly created BIB holding their plain text integrity signatures (B7). A single new BCB is used in this case because all three targets share a security source, security context, and security context parameters. Had this not been the case, multiple BCBs could have been added instead.

4. Canonical Forms

Security services require consistency and determinism in how information is presented to cipher suites at security sources, verifiers, and acceptors. For example, integrity services require that the same target information (e.g., the same bits in the same order) is provided to the cipher suite when generating an original signature and when validating a signature. Canonicalization algorithms transcode the contents of a security target into a canonical form.

Canonical forms are used to generate input to a security context for security processing at a BP node. If the values of a security target are unchanged, then the canonical form of that target will be the same even if the encoding of those values for wire transmission is different.

BPSec operates on data fields within bundle blocks (e.g., the block-type-specific-data field). In their canonical form, these fields MUST include their own CBOR encoding and MUST NOT include any other encapsulating CBOR encoding. For example, the canonical form of the block-type-specific-data field is a CBOR byte string existing within the CBOR array containing the fields of the extension block. The entire CBOR byte string is considered the canonical block-type-specific-data field. The CBOR array framing is not considered part of the field.

The canonical form of the primary block is as specified in [I-D.ietf-dtn-bpbis] with the following constraint.

- o CBOR values from the primary block MUST be canonicalized using the rules for Canonical CBOR, as specified in [RFC7049]. Canonical CBOR generally requires that integers and type lengths are encoded to be as small as possible, that map values be sorted, and that indefinite-length items be made into definite-length items.

All non-primary blocks share the same block structure and are canonicalized as specified in [I-D.ietf-dtn-bpbis] with the following constraints.

- o CBOR values from the non-primary block MUST be canonicalized using the rules for Canonical CBOR, as specified in [RFC7049].

- Only the block-type-specific-data field may be provided to a cipher suite for encryption as part of a confidentiality security service. Other fields within a non-primary-block MUST NOT be encrypted or decrypted and MUST NOT be included in the canonical form used by the cipher suite for encryption and decryption. These other fields MAY have an integrity protection mechanism applied to them by treating them as associated authenticated data.

- Reserved and unassigned flags in the block processing control flags field MUST be set to 0 in a canonical form as it is not known if those flags will change in transit.

Security contexts MAY define their own canonicalization algorithms and require the use of those algorithms over the ones provided in this specification. In the event of conflicting canonicalization algorithms, algorithms defined in a security context take precedence over this specification when constructing canonical forms for that security context.

5. Security Processing

This section describes the security aspects of bundle processing.

5.1. Bundles Received from Other Nodes

Security blocks must be processed in a specific order when received by a BP node. The processing order is as follows.

- When BIBs and BCBs share a security target, BCBs MUST be evaluated first and BIBs second.

5.1.1. Receiving BCBs

If a received bundle contains a BCB, the receiving node MUST determine whether it is the security acceptor for any of the security operations in the BCB. If so, the node MUST process those operations and remove any operation-specific information from the BCB prior to delivering data to an application at the node or forwarding the bundle. If processing a security operation fails, the target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. When all security operations for a BCB have been removed from the BCB, the BCB MUST be removed from the bundle.

If the receiving node is the destination of the bundle, the node MUST decrypt any BCBs remaining in the bundle. If the receiving node is not the destination of the bundle, the node MUST process the BCB if directed to do so as a matter of security policy.

If the security policy of a node specifies that a node should have applied confidentiality to a specific security target and no such BCB is present in the bundle, then the node MUST process this security target in accordance with the security policy. It is RECOMMENDED that the node remove the security target from the bundle because the confidentiality (and possibly the integrity) of the security target cannot be guaranteed. If the removed security target is the payload block, the bundle MUST be discarded.

If an encrypted payload block cannot be decrypted (i.e., the cipher text cannot be authenticated), then the bundle MUST be discarded and processed no further. If an encrypted security target other than the payload block cannot be decrypted then the associated security target and all security blocks associated with that target MUST be discarded and processed no further. In both cases, requested status reports (see [I-D.ietf-dtn-bpbis]) MAY be generated to reflect bundle or block deletion.

When a BCB is decrypted, the recovered plain text for each security target MUST replace the cipher text in each of the security targets' block-type-specific-data fields. If the plain text is of different size than the cipher text, the CBOR byte string framing of this field must be updated to ensure this field remains a valid CBOR byte string. The length of the recovered plain text is known by the decrypting security context.

If a BCB contains multiple security operations, each operation processed by the node MUST be treated as if the security operation has been represented by a single BCB with a single security operation for the purposes of report generation and policy processing.

5.1.2. Receiving BIBs

If a received bundle contains a BIB, the receiving node MUST determine whether it is the security acceptor for any of the security operations in the BIB. If so, the node MUST process those operations and remove any operation-specific information from the BIB prior to delivering data to an application at the node or forwarding the bundle. If processing a security operation fails, the target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. When all security operations for a BIB have been removed from the BIB, the BIB MUST be removed from the bundle.

A BIB MUST NOT be processed if the security target of the BIB is also the security target of a BCB in the bundle. Given the order of operations mandated by this specification, when both a BIB and a BCB share a security target, it means that the security target must have been encrypted after it was integrity signed and, therefore, the BIB cannot be verified until the security target has been decrypted by processing the BCB.

If the security policy of a node specifies that a node should have applied integrity to a specific security target and no such BIB is present in the bundle, then the node MUST process this security target in accordance with the security policy. It is RECOMMENDED that the node remove the security target from the bundle if the security target is not the payload or primary block. If the security target is the payload or primary block, the bundle MAY be discarded. This action can occur at any node that has the ability to verify an integrity signature, not just the bundle destination.

If a receiving node is not the security acceptor of a security operation in a BIB it MAY attempt to verify the security operation anyway to prevent forwarding corrupt data. If the verification fails, the node SHALL process the security target in accordance to local security policy. It is RECOMMENDED that if a payload integrity check fails at a waypoint that it is processed in the same way as if the check fails at the bundle destination. If the check passes, the node MUST NOT remove the security operation from the BIB prior to forwarding.

If a BIB contains multiple security operations, each operation processed by the node MUST be treated as if the security operation has been represented by a single BIB with a single security operation for the purposes of report generation and policy processing.

5.2. Bundle Fragmentation and Reassembly

If it is necessary for a node to fragment a bundle payload, and security services have been applied to that bundle, the fragmentation rules described in [I-D.ietf-dtn-bpbis] MUST be followed. As defined there and summarized here for completeness, only the payload block can be fragmented; security blocks, like all extension blocks, can never be fragmented.

Due to the complexity of payload block fragmentation, including the possibility of fragmenting payload block fragments, integrity and confidentiality operations are not to be applied to a bundle representing a fragment. Specifically, a BCB or BIB MUST NOT be added to a bundle if the "Bundle is a Fragment" flag is set in the Bundle Processing Control Flags field.

Security processing in the presence of payload block fragmentation may be handled by other mechanisms outside of the BPSec protocol or by applying BPSec blocks in coordination with an encapsulation mechanism. A node should apply any confidentiality protection prior to performing any fragmentation.

6. Key Management

There exist a myriad of ways to establish, communicate, and otherwise manage key information in a DTN. Certain DTN deployments might follow established protocols for key management whereas other DTN deployments might require new and novel approaches. BPSec assumes that key management is handled as a separate part of network management and this specification neither defines nor requires a specific key management strategy.

7. Security Policy Considerations

When implementing BPSec, several policy decisions must be considered. This section describes key policies that affect the generation, forwarding, and receipt of bundles that are secured using this specification. No single set of policy decisions is envisioned to work for all secure DTN deployments.

- If a bundle is received that contains combinations of security operations that are disallowed by this specification the BPA must determine how to handle the bundle. The bundle may be discarded, the block affected by the security operation may be discarded, or one security operation may be favored over another.

- BPAs in the network must understand what security operations they should apply to bundles. This decision may be based on the source of the bundle, the destination of the bundle, or some other information related to the bundle.

- If a waypoint has been configured to add a security operation to a bundle, and the received bundle already has the security operation applied, then the receiver must understand what to do. The receiver may discard the bundle, discard the security target and associated BPSec blocks, replace the security operation, or some other action.

- It is RECOMMENDED that security operations be applied to every block in a bundle and that the default behavior of a bundle agent is to use the security services defined in this specification. Designers should only deviate from the use of security operations when the deviation can be justified - such as when doing so causes downstream errors when processing blocks whose contents must be inspected or changed at one or more hops along the path.

- BCB security contexts can alter the size of extension blocks and the payload block. Security policy SHOULD consider how changes to the size of a block could negatively effect bundle processing (e.g., calculating storage needs and scheduling transmission times).

- Adding a BIB to a security target that has already been encrypted by a BCB is not allowed. If this condition is likely to be encountered, there are (at least) three possible policies that could handle this situation.

    1. At the time of encryption, a security context can be selected which computes a plain text integrity-protection mechanism that is included as a security context result field.

    2. The encrypted block may be replicated as a new block with a new block number and given integrity protection.

    3. An encapsulation scheme may be applied to encapsulate the security target (or the entire bundle) such that the encapsulating structure is, itself, no longer the security target of a BCB and may therefore be the security target of a BIB.

- Security policy SHOULD address whether cipher suites whose cipher text is larger than the initial plain text are permitted and, if so, for what types of blocks. Changing the size of a block may cause processing difficulties for networks that calculate block offsets into bundles or predict transmission times or storage availability as a function of bundle size. In other cases, changing the size of a payload as part of encryption has no significant impact.

7.1. Security Reason Codes

Bundle protocol agents (BPAs) must process blocks and bundles in accordance with both BP policy and BPSec policy. The decision to receive, forward, deliver, or delete a bundle may be communicated to the report-to address of the bundle, in the form of a status report, as a method of tracking the progress of the bundle through the network. The status report for a bundle may be augmented with a "reason code" explaining why the particular action was taken on the bundle.

This section describes a set of reason codes associated with the security processing of a bundle. Security reason codes are assigned in accordance with Section 11.2.

Missing Security Operation:
This reason code indicates that a bundle was missing one or more required security operations. This reason code is typically used by a security verifier or security acceptor.

Unknown Security Operation:
This reason code indicates that one or more security operations present in a bundle cannot be understood by the security verifier or security acceptor for the operation. For example, this reason code may be used if a security block references an unknown security context identifier or security context parameter. This reason code should not be used for security operations for which the node is not a security verifier or security acceptor; there is no requirement that all nodes in a network understand all security contexts, security context parameters, and security services for every bundle in a network.

Unexpected Security Operation:
This reason code indicates that a receiving node is neither a security verifier nor a security acceptor for at least one security operation in a bundle. This reason code should not be seen as an error condition; not every node is a security verifier or security acceptor for every security operation in every bundle. In certain networks, this reason code may be useful in identifying misconfigurations of security policy.

Failed Security Operation:
This reason code indicates that one or more security operations in a bundle failed to process as expected for reasons other than misconfiguration. This may occur when a security-source is unable to add a security block to a bundle. This may occur if the target of a security operation fails to verify using the defined security context at a security verifier. This may also occur if a security operation fails to be processed without error at a security acceptor.

Conflicting Security Operations:
This reason code indicates that two or more security operations in a bundle are not conformant with the BPSec specification and that security processing was unable to proceed because of a BPSec protocol violation.

8. Security Considerations

Given the nature of DTN applications, it is expected that bundles may traverse a variety of environments and devices which each pose unique security risks and requirements on the implementation of security within BPSec. For these reasons, it is important to introduce key threat models and describe the roles and responsibilities of the BPSec protocol in protecting the confidentiality and integrity of the data against those threats. This section provides additional discussion on security threats that BPSec will face and describes how BPSec security mechanisms operate to mitigate these threats.

The threat model described here is assumed to have a set of capabilities identical to those described by the Internet Threat Model in [RFC3552], but the BPSec threat model is scoped to illustrate threats specific to BPSec operating within DTN environments and therefore focuses on on-path-attackers (OPAs). In doing so, it is assumed that the DTN (or significant portions of the DTN) are completely under the control of an attacker.

8.1. Attacker Capabilities and Objectives

BPSec was designed to protect against OPA threats which may have access to a bundle during transit from its source, Alice, to its destination, Bob. An OPA node, Olive, is a non-cooperative node operating on the DTN between Alice and Bob that has the ability to receive bundles, examine bundles, modify bundles, forward bundles, and generate bundles at will in order to compromise the confidentiality or integrity of data within the DTN. There are three classes of OPA nodes which are differentiated based on their access to cryptographic material:

- Unprivileged Node: Olive has not been provisioned within the secure environment and only has access to cryptographic material which has been publicly-shared.

- Legitimate Node: Olive is within the secure environment and therefore has access to cryptographic material which has been provisioned to Olive (i.e., K_M) as well as material which has been publicly-shared.

- Privileged Node: Olive is a privileged node within the secure environment and therefore has access to cryptographic material which has been provisioned to Olive, Alice and/or Bob (i.e. K_M, K_A, and/or K_B) as well as material which has been publicly-shared.

If Olive is operating as a privileged node, this is tantamount to compromise; BPSec does not provide mechanisms to detect or remove Olive from the DTN or BPSec secure environment. It is up to the BPSec implementer or the underlying cryptographic mechanisms to provide appropriate capabilities if they are needed. It should also be noted that if the implementation of BPSec uses a single set of shared cryptographic material for all nodes, a legitimate node is equivalent to a privileged node because K_M == K_A == K_B. For this reason, sharing cryptographic material in this way is not recommended.

A special case of the legitimate node is when Olive is either Alice or Bob (i.e., K_M == K_A or K_M == K_B). In this case, Olive is able to impersonate traffic as either Alice or Bob, respectively, which means that traffic to and from that node can be decrypted and encrypted, respectively. Additionally, messages may be signed as originating from one of the endpoints.

8.2. Attacker Behaviors and BPSec Mitigations

8.2.1. Eavesdropping Attacks

Once Olive has received a bundle, she is able to examine the contents of that bundle and attempt to recover any protected data or cryptographic keying material from the blocks contained within. The protection mechanism that BPSec provides against this action is the BCB, which encrypts the contents of its security target, providing confidentiality of the data. Of course, it should be assumed that Olive is able to attempt offline recovery of encrypted data, so the cryptographic mechanisms selected to protect the data should provide a suitable level of protection.

When evaluating the risk of eavesdropping attacks, it is important to consider the lifetime of bundles on a DTN. Depending on the network, bundles may persist for days or even years. Long-lived bundles imply that the data exists in the network for a longer period of time and, thus, there may be more opportunities to capture those bundles. Additionally, bundles that are long-lived imply that the information stored within them may remain relevant and sensitive for long enough that, once captured, there is sufficient time to crack encryption associated with the bundle. If a bundle does persist on the network for years and the cipher suite used for a BCB provides inadequate protection, Olive may be able to recover the protected data either before that bundle reaches its intended destination or before the information in the bundle is no longer considered sensitive.

NOTE: Olive is not limited by the bundle lifetime and may retain a given bundle indefinitely.

NOTE: Irrespective of whether BPSec is used, traffic analysis will be possible.

8.2.2. Modification Attacks

As a node participating in the DTN between Alice and Bob, Olive will also be able to modify the received bundle, including non-BPSec data such as the primary block, payload blocks, or block processing control flags as defined in [I-D.ietf-dtn-bpbis]. Olive will be able to undertake activities which include modification of data within the blocks, replacement of blocks, addition of blocks, or removal of blocks. Within BPSec, both the BIB and BCB provide integrity protection mechanisms to detect or prevent data manipulation attempts by Olive.

The BIB provides that protection to another block which is its security target. The cryptographic mechanisms used to generate the BIB should be strong against collision attacks and Olive should not have access to the cryptographic material used by the originating node to generate the BIB (e.g., K_A). If both of these conditions are true, Olive will be unable to modify the security target or the BIB and lead Bob to validate the security target as originating from Alice.

Since BPSec security operations are implemented by placing blocks in a bundle, there is no in-band mechanism for detecting or correcting certain cases where Olive removes blocks from a bundle. If Olive removes a BCB, but keeps the security target, the security target remains encrypted and there is a possibility that there may no longer be sufficient information to decrypt the block at its destination. If Olive removes both a BCB (or BIB) and its security target there is no evidence left in the bundle of the security operation. Similarly, if Olive removes the BIB but not the security target there is no evidence left in the bundle of the security operation. In each of these cases, the implementation of BPSec must be combined with policy configuration at endpoints in the network which describe the expected and required security operations that must be applied on transmission and are expected to be present on receipt. This or other similar out-of-band information is required to correct for removal of security information in the bundle.

A limitation of the BIB may exist within the implementation of BIB validation at the destination node. If Olive is a legitimate node within the DTN, the BIB generated by Alice with K_A can be replaced with a new BIB generated with K_M and forwarded to Bob. If Bob is only validating that the BIB was generated by a legitimate user, Bob will acknowledge the message as originating from Olive instead of Alice. Validating a BIB indicates only that the BIB was generated by a holder of the relevant key; it does not provide any guarantee that the bundle or block was created by the same entity. In order to provide verifiable integrity checks BCB should require an encryption scheme that is Indistinguishable under adaptive Chosen Ciphertext Attack (IND-CCA2) secure. Such an encryption scheme will guard against signature substitution attempts by Olive. In this case, Alice creates a BIB with the protected data block as the security target and then creates a BCB with both the BIB and protected data block as its security targets.

8.2.3. Topology Attacks

If Olive is in a OPA position within the DTN, she is able to influence how any bundles that come to her may pass through the network. Upon receiving and processing a bundle that must be routed elsewhere in the network, Olive has three options as to how to proceed: not forward the bundle, forward the bundle as intended, or forward the bundle to one or more specific nodes within the network.

Attacks that involve re-routing the packets throughout the network are essentially a special case of the modification attacks described in this section where the attacker is modifying fields within the primary block of the bundle. Given that BPSec cannot encrypt the contents of the primary block, alternate methods must be used to prevent this situation. These methods may include requiring BIBs for primary blocks, using encapsulation, or otherwise strategically manipulating primary block data. The specifics of any such mitigation technique are specific to the implementation of the deploying network and outside of the scope of this document.

Furthermore, routing rules and policies may be useful in enforcing particular traffic flows to prevent topology attacks. While these rules and policies may utilize some features provided by BPSec, their definition is beyond the scope of this specification.

8.2.4. Message Injection

Olive is also able to generate new bundles and transmit them into the DTN at will. These bundles may either be copies or slight modifications of previously-observed bundles (i.e., a replay attack) or entirely new bundles generated based on the Bundle Protocol, BPSec, or other bundle-related protocols. With these attacks Olive's objectives may vary, but may be targeting either the bundle protocol or application-layer protocols conveyed by the bundle protocol. The target could also be the storage and compute of the nodes running the bundle or application layer protocols (e.g., a denial of service to flood on the storage of the store-and-forward mechanism; or compute which would process the packets and perhaps prevent other activities).

BPSec relies on cipher suite capabilities to prevent replay or forged message attacks. A BCB used with appropriate cryptographic mechanisms may provide replay protection under certain circumstances. Alternatively, application data itself may be augmented to include mechanisms to assert data uniqueness and then protected with a BIB, a BCB, or both along with other block data. In such a case, the receiving node would be able to validate the uniqueness of the data.

For example, a BIB may be used to validate the integrity of a bundle's primary block, which includes a timestamp and lifetime for the bundle. If a bundle is replayed outside of its lifetime, then the replay attack will fail as the bundle will be discarded. Similarly, additional blocks such as the Bundle Age may be signed and validated to identify replay attacks. Finally, security context parameters within BIB and BCB blocks may include anti-replay mechanisms such as session identifiers, nonces, and dynamic passwords as supported by network characteristics.

9. Security Context Considerations

9.1. Mandating Security Contexts

Because of the diversity of networking scenarios and node capabilities that may utilize BPSec there is a risk that a single security context mandated for every possible BPSec implementation is not feasible. For example, a security context appropriate for a resource-constrained node with limited connectivity may be inappropriate for use in a well-resourced, well connected node.

This does not mean that the use of BPSec in a particular network is meant to be used without security contexts for interoperability and default behavior. Network designers must identify the minimal set of security contexts necessary for functions in their network. For example, a default set of security contexts could be created for use over the terrestrial Internet and required by any BPSec implementation communicating over the terrestrial Internet.

To ensure interoperability among various implementations, all BPSec implementations MUST support at least the current IETF standards-track mandatory security context(s). As of this writing, that BCP mandatory security context is specified in [I-D.ietf-dtn-bpsec-interop-sc], but the mandatory security context(s) might change over time in accordance with usual IETF processes. Such changes are likely to occur in the future if/when flaws are discovered in the applicable cryptographic algorithms, for example.

Additionally, BPsec implementations need to support the security contexts which are specified and/or used by the BP networks in which they are deployed.

If a node serves as a gateway amongst two or more networks, the BPSec implementation at that node needs to support the union of security contexts mandated in those networks.

BPSec has been designed to allow for a diversity of security contexts and for new contexts to be defined over time. The use of different security contexts does not change the BPSec protocol itself and the definition of new security contexts MUST adhere to the requirements of such contexts as presented in this section and generally in this specification.

Implementors should monitor the state of security context specifications to check for future updates and replacement.

9.2. Identification and Configuration

Security blocks uniquely identify the security context to be used in the processing of their security services. The security context for a security block MUST be uniquely identifiable and MAY use parameters for customization.

To reduce the number of security contexts used in a network, security context designers should make security contexts customizable through the definition of security context parameters. For example, a single security context could be associated with a single cipher suite and security context parameters could be used to configure the use of this security context with different key lengths and different key management options without needing to define separate security contexts for each possible option.

A single security context may be used in the application of more than one security service. This means that a security context identifier MAY be used with a BIB, with a BCB, or with any other BPSec-compliant security block. The definition of a security context MUST identify which security services may be used with the security context, how security context parameters are interpreted as a function of the security operation being supported, and which security results are produced for each security service.

Network operators must determine the number, type, and configuration of security contexts in a system. Networks with rapidly changing configurations may define relatively few security contexts with each context customized with multiple parameters. For networks with more stability, or an increased need for confidentiality, a larger number of contexts can be defined with each context supporting few, if any, parameters.

Security Context Examples

Context Type	Parameters	Definition
Key Exchange AES	Encrypted Key, IV	AES-GCM-256 cipher suite with provided ephemeral key encrypted with a predetermined key encryption key and clear text initialization vector.
Pre-shared Key AES	IV	AES-GCM-256 cipher suite with predetermined key and predetermined key rotation policy.
Out of Band AES	None	AES-GCM-256 cipher suite with all info predetermined.

Table 1

9.3. Authorship

Developers or implementers should consider the diverse performance and conditions of networks on which the Bundle Protocol (and therefore BPSec) will operate. Specifically, the delay and capacity of delay-tolerant networks can vary substantially. Developers should consider these conditions to better describe the conditions when those contexts will operate or exhibit vulnerability, and selection of these contexts for implementation should be made with consideration for this reality. There are key differences that may limit the opportunity for a security context to leverage existing cipher suites and technologies that have been developed for use in traditional, more reliable networks:

- o Data Lifetime: Depending on the application environment, bundles may persist on the network for extended periods of time, perhaps even years. Cryptographic algorithms should be selected to ensure protection of data against attacks for a length of time reasonable for the application.

- o One-Way Traffic: Depending on the application environment, it is possible that only a one-way connection may exist between two endpoints, or if a two-way connection does exist, the round- trip time may be extremely large. This may limit the utility of session key generation mechanisms, such as Diffie-Hellman, as a two-way handshake may not be feasible or reliable.

- Opportunistic Access: Depending on the application environment, a given endpoint may not be guaranteed to be accessible within a certain amount of time. This may make asymmetric cryptographic architectures which rely on a key distribution center or other trust center impractical under certain conditions.

When developing security contexts for use with BPSec, the following information SHOULD be considered for inclusion in these specifications.

- Security Context Parameters. Security contexts MUST define their parameter Ids, the data types of those parameters, and their CBOR encoding.

- Security Results. Security contexts MUST define their security result Ids, the data types of those results, and their CBOR encoding.

- New Canonicalizations. Security contexts may define new canonicalization algorithms as necessary.

- Cipher-Text Size. Security contexts MUST state whether their associated cipher suites generate cipher text (to include any authentication information) that is of a different size than the input plain text.

  If a security context does not wish to alter the size of the plain text it should place overflow bytes and authentication tags in security result fields.

- Block Header Information. Security contexts SHOULD include block header information that is considered to be immutable for the block. This information MAY include the block type code, block number, CRC Type and CRC field (if present or if missing and unlikely to be added later), and possibly certain block processing control flags. Designers should input these fields as additional data for integrity protection when these fields are expected to remain unchanged over the path the block will take from the security source to the security acceptor. Security contexts considering block header information MUST describe expected behavior when these fields fail their integrity verification.

10. Defining Other Security Blocks

Other security blocks (OSBs) may be defined and used in addition to the security blocks identified in this specification. Both the usage of BIB, BCB, and any future OSBs can co-exist within a bundle and can be considered in conformance with BPSec if each of the following requirements are met by any future identified security blocks.

- Other security blocks (OSBs) MUST NOT reuse any enumerations identified in this specification, to include the block type codes for BIB and BCB.

- An OSB definition MUST state whether it can be the target of a BIB or a BCB. The definition MUST also state whether the OSB can target a BIB or a BCB.

- An OSB definition MUST provide a deterministic processing order in the event that a bundle is received containing BIBs, BCBs, and OSBs. This processing order MUST NOT alter the BIB and BCB processing orders identified in this specification.

- An OSB definition MUST provide a canonicalization algorithm if the default non-primary-block canonicalization algorithm cannot be used to generate a deterministic input for a cipher suite. This requirement can be waived if the OSB is defined so as to never be the security target of a BIB or a BCB.

- An OSB definition MUST NOT require any behavior of a BPSEC-BPA that is in conflict with the behavior identified in this specification. In particular, the security processing requirements imposed by this specification must be consistent across all BPSEC-BPAs in a network.

- The behavior of an OSB when dealing with fragmentation must be specified and MUST NOT lead to ambiguous processing states. In particular, an OSB definition should address how to receive and process an OSB in a bundle fragment that may or may not also contain its security target. An OSB definition should also address whether an OSB may be added to a bundle marked as a fragment.

Additionally, policy considerations for the management, monitoring, and configuration associated with blocks SHOULD be included in any OSB definition.

NOTE: The burden of showing compliance with processing rules is placed upon the specifications defining new security blocks and the identification of such blocks shall not, alone, require maintenance of this specification.

11. IANA Considerations

This specification includes fields requiring registries managed by IANA.

11.1. Bundle Block Types

This specification allocates two block types from the existing "Bundle Block Types" registry defined in [RFC6255].

Additional Entries for the Bundle Block-Type Codes Registry:

Value	Description	Reference
TBA	Block Integrity Block	This document
TBA	Block Confidentiality Block	This document

Table 2

The Bundle Block Types namespace notes whether a block type is meant for use in BP version 6, BP version 7, or both. The two block types defined in this specification are meant for use with BP version 7.

11.2. Bundle Status Report Reason Codes

This specification allocates five reason codes from the existing "Bundle Status Report Reason Codes" registry defined in [RFC6255].

Additional Entries for the Bundle Status Report Reason Codes Registry:

BP Version	Value	Description	Reference
7	TBD	Missing Security Operation	This document, Section Section 7.1
7	TBD	Unknown Security Operation	This document, Section Section 7.1
7	TBD	Unexpected Security Operation	This document, Section Section 7.1
7	TBD	Failed Security Operation	This document, Section Section 7.1
7	TBD	Conflicting Security Operation	This document, Section Section 7.1

11.3. Security Context Identifiers

BPSec has a Security Context Identifier field for which IANA is requested to create and maintain a new registry named "BPSec Security Context Identifiers". Initial values for this registry are given below.

The registration policy for this registry is: Specification Required.

The value range is: signed 16-bit integer.

BPSec Security Context Identifier Registry

Value	Description	Reference
< 0	Reserved	This document
0	Reserved	This document

Table 3

Negative security context identifiers are reserved for local/site-specific uses. The use of 0 as a security context identifier is for non-operational testing purposes only.

12. References

12.1. Normative References

[I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-28 (work in progress), October 2020.

[I-D.ietf-dtn-bpsec-interop-sc]
Birrane, E., "BPSec Default Security Contexts", draft-ietf-dtn-bpsec-interop-sc-02 (work in progress), November 2020.

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

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

[RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, DOI 10.17487/RFC6255, May 2011, <https://www.rfc-editor.org/info/rfc6255>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

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

12.2. Informative References

[I-D.birrane-dtn-sbsp] Birrane, E., Pierce-Mayer, J., and D. Iannicca, "Streamlined Bundle Security Protocol Specification", draft-birrane-dtn-sbsp-01 (work in progress), October 2015.

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

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

Appendix A. Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this security specification: Scott Burleigh of the Jet Propulsion Laboratory, Angela Hennessy of the Laboratory for Telecommunications Sciences, and Amy Alford, Angela Dalton, and Cherita Corbett of the Johns Hopkins University Applied Physics Laboratory.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Kenneth McKeever
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 2237
Email: Ken.McKeever@jhuapl.edu

Delay Tolerant Networking
Internet-Draft
Obsoletes: 7242 (if approved)
Intended status: Standards Track
Expires: November 23, 2017

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

May 22, 2017

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4

draft-ietf-dtn-tcpclv4-02

Abstract

This document describes a revised protocol for the TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). The protocol revision is based on implementation issues in the original TCPCL Version 3 and updates to the Bundle Protocol contents, encodings, and convergence layer requirements in Bundle Protocl Version 7. Several new IANA registries are defined for TCPCLv4 which define some behaviors inherited from TCPCLv3 but with updated encodings and/or semantics.

Table of Contents

1. Introduction

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

An important goal of the DTN architecture is to accommodate a wide range of networking technologies and environments. The protocol used for DTN communications is the revised Bundle Protocol (BP) [I-D.ietf-dtn-bpbis], an application-layer protocol that is used to construct a store-and- forward overlay network. As described in the Bundle Protocol specification [I-D.ietf-dtn-bpbis], 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.

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

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

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

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

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

2. Requirements Language

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

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions 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 Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two bundle nodes. The lifetime of a TCPCL session is bound to the lifetime of an underlying TCP connection. Therefore, a TCPCL session is initiated after a bundle node establishes a TCP connection to for the purposes of bundle communication. A TCPCL session is terminated when the TCP connection ends, due either to one or both nodes actively terminating the TCP connection or due to network errors causing a failure of the TCP connection. For the remainder of this document, the term "session" without the prefix "TCPCL" refer to a TCPCL session.

Session parameters: The session parameters are a set of values used to affect the operation of the TCPCL for a given session. The manner in which these parameters are conveyed to the bundle node and thereby to the TCPCL is implementation dependent. However, the mechanism by which two bundle nodes exchange and negotiate the values to be used for a given session is described in Section 4.2.

Transfer Transfer refers to the procedures and mechanisms (described below) for conveyance of an individual bundle from one node to another. Each transfer within TCPCLv4 is identified by a Transfer ID number which is unique only to a single direction within a single Session.

3. General Protocol Description

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

First, one node establishes a TCPCL session 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 session and exchange a singleton endpoint identifier for each node (not the singleton Endpoint Identifier (EID) of any application running on the node) to denote the bundle-layer identity of each DTN node. This is used to assist in routing and forwarding messages, e.g., to prevent loops.

Once the TCPCL session is established and configured in this way, bundles can be transferred in either direction. Each transfer is performed in one or more logical segments of data. Each logical data segment consists of a XFER_SEGMENT message header and flags, a count of the length of the segment, and finally the octet range of the bundle data. The choice of the length to use for segments is an implementation matter (but must be within the Segment MRU size of

Section 4.1). The first segment for a bundle MUST set the 'START' flag, and the last one MUST set the 'end' flag in the XFER_SEGMENT message flags.

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

A feature of this protocol is for the receiving node to send acknowledgments as bundle data segments arrive (XFER_ACK). The rationale behind these acknowledgments is to enable the sender node to determine how much of the bundle has been received, so that in case the session is interrupted, it can perform reactive fragmentation to avoid re-sending the already transmitted part of the bundle. For each data segment that is received, the receiving node sends an XFER_ACK message containing the cumulative length of the bundle that has been received. The sending node MAY transmit multiple XFER_SEGMENT messages without necessarily waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a channel. In addition, there is no explicit flow control on the TCPCL layer.

Another feature is that a receiver MAY interrupt the transmission of a bundle at any point in time by replying with a XFER_REFUSE message, which causes the sender to stop transmission of the 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 sessions that are idle, a KEEPALIVE message is sent at a negotiated interval. This is used to convey liveness information.

Finally, before sessions close, a SHUTDOWN message is sent to the session peer. After sending a SHUTDOWN message, the sender of this message MAY send further acknowledgments (XFER_ACK or XFER_REFUSE) but no further data messages (XFER_SEGMENT). A SHUTDOWN message MAY also be used to refuse a session setup by a peer.

3.1. Bidirectional Use of TCPCL Sessions

There are specific messages for sending and receiving operations (in addition to session setup/teardown). TCPCL is symmetric, i.e., both sides can start sending data segments in a session, and one side's bundle transfer does not have to complete before the other side can start sending data segments on its own. Hence, the protocol allows for a bi-directional mode of communication.

Note that in the case of concurrent bidirectional transmission, acknowledgment segments MAY be interleaved with data segments.

3.2. Example Message Exchange

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

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

No errors or rejections are shown in this example.

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

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

                                           +-------------------------+
                                        <- |      XFER_ACK (end)     |
                                           |     Transfer ID [I1]    |
                                           |     Length   [L1+L2+L3] |
                                           +-------------------------+

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

Figure 2: A SL1e Visual Example of the Flow of Protocol Messages on a Single TCP Session between Two Nodes (A and B)

4. Session Establishment

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

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

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

If the node is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. A node MAY decide to re-attempt to establish the connection. If it does so, it MUST NOT overwhelm its target with repeated connection attempts. Therefore, the node MUST retry the connection setup only after some delay (a 1-second minimum is RECOMMENDED), and it SHOULD use a (binary) exponential backoff mechanism to increase this delay in case of repeated failures. In case a SHUTDOWN message specifying a reconnection delay is received, that delay is used as the initial delay. The default initial delay SHOULD be at least 1 second but SHOULD be configurable since it will be application and network type dependent.

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

Once a TCP connection is established, each node MUST immediately transmit a contact header over the TCP connection. The format of the contact header is described in Section 4.1.

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

After receiving the contact header from the other node, either node MAY also refuse the session by sending a SHUTDOWN message. If session 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:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|                          magic='dtn!'                         |
+---------------+---------------+---------------+---------------+
|     Version   |   Flags       |      Keepalive Interval       |
+---------------+---------------+---------------+---------------+
|                          Segment MRU...                       |
+---------------+---------------+---------------+---------------+
|                          contd.                               |
+---------------+---------------+---------------+---------------+
|                         Transfer MRU...                       |
+---------------+---------------+---------------+---------------+
|                          contd.                               |
+---------------+---------------+---------------+---------------+
|          EID Length           |             EID Data...       |
+---------------+---------------+---------------+---------------+
|                        EID Data contd.                        |
+---------------+---------------+---------------+---------------+
|            TCPCLv4 Header Extension Items...                  |
+---------------+---------------+---------------+---------------+
```

Figure 3: Contact Header Format

See Section 4.2 for details on the use of each of these contact header fields. The fields of the contact header are:

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

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

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

Keepalive Interval: A 16-bit unsigned integer indicating the longest allowable interval in seconds between KEEPALIVE messages received in this session.

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

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

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

Header Extension Values: The remaining items of the contact header represent protocol extension data not defined by this specification. The encoding of each Header Extension Item is identical form as described in Section 4.1.1.

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

Table 1: Contact Header Flags

4.1.1. Header Extension Items

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

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

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

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

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extensions use of available Item Value data. Extension specification SHOULD avoid the use of large data exchanges within the TCPCLv4 contact header as no bundle transfers can begin until the a full contact exchange and negotiation has been completed.

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

Figure 4: Header Extention Item Format

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

Table 2: Header Extension Item Flags

4.2. Validation and Parameter Negotiation

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

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

If a node receives a contact header containing a version that is greater than the current version of the protocol that the node implements, then the node SHALL shutdown the session with a reason code of "Version mismatch". If a node receives a contact header with a version that is lower than the version of the protocol that the node implements, the node MAY either terminate the session (with a reason code of "Version mismatch"). Otherwise, the node MAY adapt its operation to conform to the older version of the protocol. This decision is an implementation matter. When establishing the TCPCL session, a node SHOULD send the contact header for the latest version of TCPCL that it can use.

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

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of this two contact headers' Keepalive Interval. If the negotiated Session Keepalive is zero (i.e. one or both contact headers contains a zero Keepalive Interval), then the keepalive feature (described in Section 5.2.1) is disabled. There is no logical minimum value for the keepalive interval, but when used for many sessions on an open, shared network a short interval could lead to excessive traffic. For shared network use, endpoints SHOULD choose a keepalive interval no shorter than 30 seconds. There is no logical maximum value for the keepalive interval, but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens-of-minutes. Endpoints SHOULD choose a keepalive interval no longer than 10 minutes (600 seconds).

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two contact headers' CAN_TLS flags. If the negotiated Enable TLS value is true then TLS negotiation feature (described in Section 5.3) begins immediately following the contact header exchange.

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

5. Established Session Operation

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

5.1. Message Type Codes

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

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

Figure 5: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 3 below.

The message types defined in this specificaiton are listed in Table 3. Encoded values are listed in Section 8.4.

Type	Description
XFER_INIT	Contains the length (in octets) of the next transfer, as described in Section 5.4.2.
XFER_SEGMENT	Indicates the transmission of a segment of bundle data, as described in Section 5.4.3.
XFER_ACK	Acknowledges reception of a data segment, as described in Section 5.4.4.
XFER_REFUSE	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.4.5.
KEEPALIVE	Used to keep TCPCL session active, as described in Section 5.2.1.
SHUTDOWN	Indicates that one of the nodes participating in the session wishes to cleanly terminate the session, as described in Section 6.
MSG_REJECT	Contains a TCPCL message rejection, as described in Section 5.2.2.

Table 3: TCPCL Message Types

5.2. Upkeep and Status Messages

5.2.1. Session Upkeep (KEEPALIVE)

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

As described in Section 4.1, 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-octet message type code of KEEPALIVE (as described in Table 3) with no additional data. Both sides SHOULD send a KEEPALIVE message whenever the negotiated interval has elapsed with no transmission of any message (KEEPALIVE or other).

If no message (KEEPALIVE or other) has been received for at least twice the Keepalive Interval, then either party MAY terminate the session by transmitting a one-octet SHUTDOWN message (as described in Section 6.1, with reason code "Idle Timeout") and by closing the session.

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

5.2.2. Message Rejection (MSG_REJECT)

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

The format of a MSG_REJECT message follows:

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

Figure 6: Format of MSG_REJECT Messages

The fields of the MSG_REJECT message are:

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

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

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

Table 4: MSG_REJECT Reason Codes

5.3. Session Security

This version of the TCPCL supports establishing a session-level Transport Layer Security (TLS) session within an existing TCPCL session. Negotation of whether or not to initiate TLS within TCPCL session is part of the contact header as described in Section 4.2.

When TLS is used within the TCPCL it affects the entire session. By convention, this protocol uses the endpoint which initiated the underlying TCP connection as the "client" role of the TLS handshake request. Once a TLS session is established within TCPCL, there is no mechanism provided to end the TLS session and downgrade the session. If a non-TLS session is desired after a TLS session is started then the entire TCPCL session MUST be shutdown first.

After negotiating an Enable TLS parameter of true, and before any other TCPCL messages are sent within the session, the session endpoints SHALL begin a TLS handshake in accordance with [RFC5246]. The parameters within each TLS negotation are implementation dependent but any TCPCL endpoint SHOULD follow all recommended best practices of [RFC7525].

5.3.1. TLS Handshake Result

If a TLS handshake cannot negotiate a TLS session, both endpoints of the TCPCL session SHALL cause a TCPCL shutdown with reason "TLS negotiation failed".

After a TLS session is successfuly established, both TCPCL endpoints SHALL re-exchange TCPCL Contact Header messages. Any information cached from the prior Contact Header exchange SHALL be discarded. This re-exchange avoids man-in-the-middle attack in identical fashion to [RFC2595].

5.3.2. Example TLS Initiation

A summary of a typical CAN_TLS usage is shown in the sequence in Figure 7 below.

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

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

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

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

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

                       ... secured TCPCL messaging ...

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

Figure 7: A simple visual example of TCPCL TLS Establishment between two nodes

5.4. Bundle Transfer

All of the message in this section are directly associated with tranfering a bundle between TCPCL endpoints.

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one endpoint to the other. In TCPCL a transfer is accomplished by dividing a single bundle up into "segments" based on the receving-side Segment MRU (see Section 4.1).

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

5.4.1. Bundle Transfer ID

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

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

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

5.4.2. Transfer initialization (XFER_INIT)

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

The purpose of the XFER_INIT message is to allow nodes to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving node for the upcoming bundle data. See Section 5.4.5 for details on when refusal based on XFER_INIT content is acceptable.

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

The format of the XFER_INIT message is as follows:

```
+-----------------------------+
|       Message Header        |
+-----------------------------+
|      Transfer ID (U64)      |
+-----------------------------+
|  Total bundle length (U64)  |
+-----------------------------+
```

Figure 8: Format of XFER_INIT Messages

The fields of the XFER_INIT message are:

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

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

XFER_INIT messages SHALL be sent immediately before transmission of any XFER_SEGMENT messages. XFER_INIT messages MUST NOT be sent unless the next XFER_SEGMENT message has the 'START' bit set to "1" (i.e., just before the start of a new transfer).

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

Upon reception of a XFER_INIT message not immediately before the start of a starting XFER_SEGMENT the reciever SHALL send a MSG_REJECT message with a Reason Code of "Message Unexpected".

5.4.3. Data Transmission (XFER_SEGMENT)

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

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

Figure 9: Format of XFER_SEGMENT Messages

The fields of the XFER_SEGMENT message are:

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

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

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

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

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

Table 5: XFER_SEGMENT Flags

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

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

5.4.4. Data Acknowledgments (XFER_ACK)

Although the TCP transport provides reliable transfer of data between transport peers, the typical BSD sockets interface provides no means to inform a sending application of when the receiving application has processed some amount of transmitted data. Thus, after transmitting some data, 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 provides feedback messaging whereby a receiving node transmits acknowledgments of reception of data segments.

The format of an XFER_ACK message follows in Figure 10.

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

Figure 10: Format of XFER_ACK Messages

The fields of the XFER_ACK message are:

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

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

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

A receving TCPCL endpoing SHALL send an XFER_ACK message in response to each received XFER_SEGMENT message. The flags portion of the XFER_ACK header SHALL be set to match the corresponding DATA_SEGEMNT message being acknowledged. The acknowledged length of each XFER_ACK contains the sum of the data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer.

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.

5.4.5. Transfer Refusal (XFER_REFUSE)

As bundles can be large, the TCPCL supports an optional mechanism 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 XFER_INIT or XFER_SEGMENT message, the node MAY transmit a XFER_REFUSE message. As data segments and acknowledgments MAY cross on the wire, the bundle that is being refused SHALL be identified by the Transfer ID of the refusal.

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

The format of the XFER_REFUSE message is as follows:

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

Figure 11: Format of XFER_REFUSE Messages

The fields of the XFER_REFUSE message are:

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

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

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

Table 6: XFER_REFUSE Reason Codes

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

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

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

6. Session Termination

This section describes the procedures for ending a TCPCL session.

6.1. Shutdown Message (SHUTDOWN)

To cleanly shut down a session, a SHUTDOWN message MUST be transmitted by either node at any point following complete transmission of any other message. A receiving node SHOULD acknowledge all received data segments before sending a SHUTDOWN message to end the session. A transmitting node SHALL treat a SHUTDOWN message received mid-transfer (i.e. before the final acknowledgement) as a failure of the transfer.

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

The format of the SHUTDOWN message is as follows:

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

Figure 12: Format of SHUTDOWN Messages

The fields of the SHUTDOWN message are:

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

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

Reconnection Delay: A 16-bit unsigned integer indicating the desired delay until further TCPCL sessions to the sending endpoint. The Reconnection Delay is present or absent as indicated by one of the flags.

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

Table 7: SHUTDOWN Flags

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

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

Table 8: SHUTDOWN Reason Codes

It is also possible to convey a requested reconnection delay to indicate how long the other node MUST wait before attempting session re-establishment. To do so, the node sets the 'D' bit in the message flags and then transmits an 16-bit unsigned integer 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 session. In contrast, if the node does not wish to request a delay, it SHOULD omit the reconnection delay field (and set the 'D' bit to zero).

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

If either node terminates a session 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 the magic string was not present, a node SHOULD close the TCP connection without sending a SHUTDOWN message. If a node does not want its peer to reopen a 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 session setup.

If a session is to be terminated before another protocol message has completed being sent, then the node MUST NOT transmit the SHUTDOWN message but still SHOULD close the TCP connection. This means that a SHUTDOWN cannot be used to preempt any other TCPCL messaging in-progress (particularly important when large segment sizes are being transmitted).

6.2. Idle Session Shutdown

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

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

7. Security Considerations

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

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

Table 9: Recommended Unsecured Contact Header

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

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

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

8. IANA Considerations

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

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

8.1. Port Number

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

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

8.2. Protocol Versions

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

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

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

8.3. Header Extension Types

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

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

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

Table 10: Header Extension Type Codes

8.4. Message Types

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

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

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

Table 11: Message Type Codes

8.5. XFER_REFUSE Reason Codes

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

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

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

Table 12: XFER_REFUSE Reason Codes

8.6. SHUTDOWN Reason Codes

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

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

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

Table 13: SHUTDOWN Reason Codes

8.7. MSG_REJECT Reason Codes

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

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

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

Table 14: REJECT Reason Codes

9. Acknowledgments

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

10. References

10.1. Normative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

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

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

[I-D.ietf-dtn-bpbis] Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol", draft-ietf-dtn-bpbis-06 (work in progress), October 2016.

[refs.IANA-BP] IANA, "Bundle Protocol registry", May 2016.

10.2. Informative References

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

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

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

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

[I-D.ietf-dtn-bpsec] Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-04 (work in progress), March 2017.

Appendix A. Significant changes from RFC7242

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

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

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

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

Authors' Addresses

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

Email: BSipos@rkf-eng.com

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

Email: demmer@cs.berkeley.edu

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

Email: jo@netlab.tkk.fi

Simon Perreault
Quebec, QC
Canada

Email: simon@per.reau.lt

Delay-Tolerant Networking B. Sipos
Internet-Draft RKF Engineering
Intended status: Standards Track M. Demmer
Expires: 18 May 2021 UC Berkeley
J. Ott
Aalto University
S. Perreault
14 November 2020

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-23

Abstract

This document describes a TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). This version of the TCPCL protocol resolves implementation issues in the earlier TCPCL Version 3 of RFC7242 and updates to the Bundle Protocol (BP) contents, encodings, and convergence layer requirements in BP Version 7. Specifically, the TCPCLv4 uses CBOR-encoded BPv7 bundles as its service data unit being transported and provides a reliable transport of such bundles. This version of TCPCL also includes security and extensibility mechanisms.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

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

1.1. Scope

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

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

* Mechanisms for locating or identifying other bundle entities (peers) within a network or across an internet. The mapping of Node ID to potential convergence layer (CL) protocol and network address is left to implementation and configuration of the BP Agent and its various potential routing strategies.

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

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

* Uses of TLS which are not based on PKIX certificate authentication (see Section 8.11.2) or in which authentication of both entities is not possible (see Section 8.11.1).

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

2. Requirements Language

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

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions specific to the TCPCL protocol.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2: The relationships between TCPCL entities

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

Figure 3: The relationship within a TCPCL Session of its two streams

3. General Protocol Description

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

3.1. Convergence Layer Services

This version of the TCPCL provides the following services to support the overlaying Bundle Protocol agent. In all cases, this is not an API definition but a logical description of how the CL can interact with the BP agent. Each of these interactions can be associated with any number of additional metadata items as necessary to support the operation of the CL or BP agent.

Attempt Session: The TCPCL allows a BP agent to preemptively attempt to establish a TCPCL session with a peer entity. Each session attempt can send a different set of session negotiation parameters as directed by the BP agent.

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

Session State Changed: The TCPCL entity indicates to the BP agent when the session state changes. The top-level session states indicated are:

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

Contact Negotiating: A TCP connection has been made (as either active or passive entity) and contact negotiation has begun.

Session Negotiating: Contact negotiation has been completed (including possible TLS use) and session negotiation has begun.

Established: The session has been fully established and is ready for its first transfer.

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

Terminated: The session has finished normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed: The TCPCL entity indicates to the BP agent when the live/idle sub-state of the session changes. This occurs only when the top-level session state is "Established". The session transitions from Idle to Live at the at the start of a transfer in either transfer stream; the session transitions from Live to Idle at the end of a transfer when the other transfer stream does not have an ongoing transfer. Because TCPCL transmits serially over a TCP connection it suffers from "head of queue blocking," so a transfer in either direction can block an immediate start of a new transfer in the session.

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

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

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

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

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

Interrupt Reception: The TCPCL entity allows a BP agent to interrupt an individual transfer before it has fully completed (successfully or not). Interruption can occur any time after the reception is initialized.

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

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

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

3.2. TCPCL Session Overview

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

Once negotiated, the parameters of a TCPCL session cannot change and if there is a desire by either peer to transfer data under different parameters then a new session must be established. This makes CL logic simpler but relies on the assumption that establishing a TCP connection is lightweight enough that TCP connection overhead is negligible compared to TCPCL data sizes.

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

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

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

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

A SESS_TERM message is used to initiate the ending of a TCPCL session (see Section 6.1). During termination sequencing, in-progress transfers can be completed but no new transfers can be initiated. A

SESS_TERM message can also be used to refuse a session setup by a peer (see Section 4.3). Regardless of the reason, session termination is initiated by one of the entities and responded-to by the other as illustrated by Figure 13 and Figure 14. Even when there are no transfers queued or in-progress, the session termination procedure allows each entity to distinguish between a clean end to a session and the TCP connection being closed because of some underlying network issue.

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

3.3. TCPCL States and Transitions

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

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

Figure 4: Top-level states of a TCPCL session

Notes on Established Session states:

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

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

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

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

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

Figure 5: Contact Initiation as Active Entity

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

Figure 6: Contact Initiation as Passive Entity

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

Figure 7: Processing of Contact Header [PCH]

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

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

Figure 8: Session Initiation [SI] as Active Entity

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

Figure 9: Session Initiation [SI] as Passive Entity

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

Figure 10: Processing of Session Initiation [PSI]

Transfers can occur after a session is established and it’s not in the Ending state. Each transfer occurs within a single logical transfer stream between a sender and a receiver, as illustrated in Figure 11 and Figure 12 respectively.

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

Figure 11: Transfer sender states

Notes on transfer sending:

Pipelining of transfers can occur when the sending entity begins a new transfer while in the "Waiting for Ack" state.

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

Figure 12: Transfer receiver states

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

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

Figure 13: Session Termination [ST] from the Initiator

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

Figure 14: Session Termination [ST] from the Responder

3.4. PKIX Environments and CA Policy

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

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

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

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

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

3.5. Session Keeping Policies

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

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

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

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

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

3.6. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. A receiving entity can set the Segment MRU in its SESS_INIT message to determine the largest acceptable segment size, and a transmitting entity can segment a transfer into any sizes smaller than the receiver's Segment MRU. It is a network administration matter to determine an appropriate segmentation policy for entities operating TCPCL, but guidance given here can be used to steer policy toward performance goals. It is also advised to consider the Segment MRU in relation to chunking/packetization performed by TLS, TCP, and any intermediate network-layer nodes.

Minimum Overhead: For a simple network expected to exchange relatively small bundles, the Segment MRU can be set to be identical to the Transfer MRU which indicates that all transfers can be sent with a single data segment (i.e., no actual segmentation). If the network is closed and all transmitters are known to follow a single-segment transfer policy, then receivers can avoid the necessity of segment reassembly. Because this CL operates over a TCP stream, which suffers from a form of head-of-queue blocking between messages, while one entity is transmitting a single XFER_SEGMENT message it is not able to transmit any XFER_ACK or XFER_REFUSE for any associated received transfers.

Predictable Message Sizing: In situations where the maximum message size is desired to be well-controlled, the Segment MRU can be set to the largest acceptable size (the message size less XFER_SEGMENT header size) and transmitters can always segment a transfer into maximum-size chunks no larger than the Segment MRU. This guarantees that any single XFER_SEGMENT will not monopolize the TCP stream for too long, which would prevent outgoing XFER_ACK and XFER_REFUSE associated with received transfers.

Dynamic Segmentation: Even after negotiation of a Segment MRU for each receiving entity, the actual transfer segmentation only needs to guarantee than any individual segment is no larger than that MRU. In a situation where TCP throughput is dynamic, the transfer segmentation size can also be dynamic in order to control message transmission duration.

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

3.7. Example Message Exchange

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

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

No errors or rejections are shown in this example.

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

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

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


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

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

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

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating entities. It is up to the implementation to decide how and when session setup is triggered. For example, some sessions can be opened proactively and maintained for as long as is possible given the network conditions, while other sessions are be opened only when there is a bundle that is queued for transmission and the routing algorithm selects a certain next-hop node.

4.1. TCP Connection

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

If the entity is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. An entity MAY decide to re-attempt to establish the connection. If it does so, it MUST NOT overwhelm its target with repeated connection attempts. Therefore, the entity MUST NOT retry the connection setup earlier than some delay time from the last attempt, and it SHOULD use a (binary) exponential back-off mechanism to increase this delay in case of repeated failures. The upper limit on a re-attempt back-off is implementation defined but SHOULD be no longer than one minute (60 seconds) before signaling to the BP agent that a connection cannot be made.

Once a TCP connection is established, the active entity SHALL immediately transmit its Contact Header. Once a TCP connection is established, the passive entity SHALL wait for the peer's Contact Header. If the passive entity does not receive a Contact Header after some implementation-defined time duration after TCP connection is established, the entity SHALL close the TCP connection. Entities SHOULD choose a Contact Header reception timeout interval no longer than one minute (60 seconds). Upon reception of a Contact Header, the passive entity SHALL transmit its Contact Header. The ordering of the Contact Header exchange allows the passive entity to avoid allocating resources to a potential TCPCL session until after a valid Contact Header has been received from the active entity. This ordering also allows the passive peer to adapt to alternate TCPCL protocol versions.

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

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

4.2. Contact Header

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

If an entity is capable of exchanging messages according to TLS 1.3 [RFC8446] or any successors which are compatible with that TLS ClientHello, the the CAN_TLS flag within its Contact Header SHALL be set to 1. This behavior prefers the use of TLS when possible, even if security policy does not allow or require authentication. This follows the opportunistic security model of [RFC7435], though an active attacker could interfere with the exchange in such cases.

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

The format for the Contact Header is as follows:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|                          magic='dtn!'                         |
+---------------+---------------+---------------+---------------+
|     Version   |   Flags       |
+---------------+---------------+
```

Figure 16: Contact Header Format

See Section 4.3 for details on the use of each of these Contact Header fields.

The fields of the Contact Header are:

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

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

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

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

Table 1: Contact Header Flags

4.3. Contact Validation and Negotiation

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

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

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

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

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

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

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two Contact Headers' CAN_TLS flags. A local security policy is then applied to determine of the negotiated value of Enable TLS is acceptable. It can be a reasonable security policy to require or disallow the use of TLS depending upon the desired network flows. Because this state is negotiated over an unsecured medium, there is a risk of a TLS Stripping as described in Section 8. If the Enable TLS state is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note that this contact failure reason is different than a failure of TLS handshake or TLS authentication after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is true and acceptable then TLS negotiation feature (described in Section 4.4) begins immediately following the Contact Header exchange.

4.4. Session Security

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

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

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

4.4.1. Entity Identification

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

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

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

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

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

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

When only a DNS-ID or IPADDR-ID can be identified by a certificate, it is implied that an entity which authenticates using that certificate is trusted to provide a valid Node ID in its SESS_INIT; the certificate itself does not actually authenticate that Node ID.

The RECOMMENDED security policy of an entity is to validate a NODE-ID when present, and only require DNS-ID/IPADDR-ID authentication in the absence of a NODE-ID.

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

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

4.4.2. TLS Handshake

The use of TLS is negotiated using the Contact Header as described in Section 4.3. After negotiating an Enable TLS parameter of true, and before any other TCPCL messages are sent within the session, the session entities SHALL begin a TLS handshake in accordance with [RFC8446]. By convention, this protocol uses the entity which initiated the underlying TCP connection (the active peer) as the "client" role of the TLS handshake request.

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

The parameters within each TLS negotiation are implementation dependent but any TCPCL entity SHALL follow all recommended practices of BCP 195 [RFC7525], or any updates or successors that become part of BCP 195. Within each TLS handshake, the following requirements apply (using the rough order in which they occur):

Client Hello: When a resolved DNS name was used to establish the TCP connection, the TLS ClientHello SHOULD include a Server Name Indication (SNI) in accordance with [RFC6066]. When present, the "server_name" extension SHALL contain a "HostName" value taken from the DNS name (of the passive entity) which was resolved. Note: The 'HostName in the "server_name" extension is the network name for the passive entity, not the Node ID of that entity.

Server Certificate: The passive entity SHALL supply a certificate within the TLS handshake to allow authentication of its side of the session. Unless prohibited by CA policy, the passive entity certificate SHALL contain a NODE-ID which authenticates the Node ID of the peer. When assigned a stable DNS name, the passive entity certificate SHOULD contain a DNS-ID which authenticates that (fully qualified) name. When assigned a stable network address, the passive entity certificate MAY contain a IPADDR-ID which authenticates that address. The passive entity MAY use the SNI DNS name to choose an appropriate server-side certificate which authenticates that DNS name.

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

Client Certificate: The active entity SHALL supply a certificate chain within the TLS handshake to allow authentication of its side of the session. Unless prohibited by CA policy, the active entity certificate SHALL contain a NODE-ID which authenticates the Node ID of the peer. When assigned a stable DNS name, the active entity certificate SHOULD contain a DNS-ID which authenticates that (fully qualified) name. When assigned a stable network address, the active entity certificate MAY contain a IPADDR-ID which authenticates that address.

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

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

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

4.4.3. TLS Authentication

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

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

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

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

By using the SNI DNS name (see Section 4.4.2) a single passive entity can act as a convergence layer for multiple BP agents with distinct Node IDs. When this "virtual host" behavior is used, the DNS name is used as the indication of which BP Node the active entity is attempting to communicate with. A virtual host CL entity can be authenticated by a certificate containing all of the DNS names and/or Node IDs being hosted or by several certificates each authenticating a single DNS name and/or Node ID, using the SNI value from the peer to select which certificate to use.

For any peer certificate received during TLS handshake, the entity SHALL perform the certification path validation of [RFC5280] up to one of the entity's trusted CA certificates. If certificate validation fails or if security policy disallows a certificate for any reason, the entity SHALL fail the TLS handshake with a bad certificate error. Leaving out part of the certification chain can cause the entity to fail to validate a certificate if the left-out certificates are unknown to the entity (see Section 8.6).

The result of authenticating a peer identity (i.e., Node ID, DNS name, or IP address) against one type of certificate claim is one of:

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

Success: Indicating that one or more claims of that type are present and one matches the peer identity value.

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

Either during or immediately after the TLS handshake if the active entity resolved a DNS name (of the passive entity) in order to initiate the TCP connection, the active entity SHALL authenticate that DNS name using any DNS-ID of the peer certificate. If the DNS name authentication result is Failure or if the result is Absent and security policy requires an authenticated DNS name, the entity SHALL terminate the session (with a reason code of "Contact Failure").

Either during or immediately after the TLS handshake, each side of the session SHALL authenticate the IP address of the other side of the TCP connection using any IPADDR-ID of the peer certificate. If the address authentication result is Failure or if the result is Absent and security policy requires an authenticated network address, the entity SHALL terminate the session (with a reason code of "Contact Failure").

Immediately before Session Parameter Negotiation, each side of the session SHALL authenticate the Node ID of the peer's SESS_INIT message using any NODE-ID of the peer certificate. If the Node ID authentication result is Failure or if the result is Absent and security policy requires an authenticated Node ID, the entity SHALL terminate the session (with a reason code of "Contact Failure").

4.4.4. Example TLS Initiation

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

```
               Entity A                             Entity B
              active peer                          passive peer

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

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

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

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

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

                        NODE-ID authentication occurs.
                 Session is established, transfers can begin.

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

      +-------------------------+
      |    TLS Closure Alert    | ->        +-------------------------+
      +-------------------------+      <- |    TLS Closure Alert    |
                                          +-------------------------+
      +-------------------------+         +-------------------------+
      |        TCP Close        | ->   <- |        TCP Close        |
      +-------------------------+         +-------------------------+
```

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

4.5. Message Header

After the initial exchange of a Contact Header and (if TLS is negotiated to be used) the TLS handshake, all messages transmitted over the session are identified by a one-octet header with the following structure:

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

Figure 18: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 2 below. Encoded values are listed in Section 9.5.

Name	Code	Description
SESS_INIT	0x07	Contains the session parameter inputs from one of the entities, as described in Section 4.6.
SESS_TERM	0x05	Indicates that one of the entities participating in the session wishes to cleanly terminate the session, as described in Section 6.1.
XFER_SEGMENT	0x01	Indicates the transmission of a segment of bundle data, as described in Section 5.2.2.
XFER_ACK	0x02	Acknowledges reception of a data segment, as described in Section 5.2.3.
XFER_REFUSE	0x03	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.2.4.
KEEPALIVE	0x04	Used to keep TCPCL session active, as described in Section 5.1.1.
MSG_REJECT	0x06	Contains a TCPCL message rejection, as described in Section 5.1.2.

Table 2: TCPCL Message Types

4.6. Session Initialization Message (SESS_INIT)

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

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

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

Figure 19: SESS_INIT Format

The fields of the SESS_INIT message are:

Keepalive Interval: A 16-bit unsigned integer indicating the minimum interval, in seconds, to negotiate as the Session Keepalive using the method of Section 4.7.

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

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

Node ID Length and Node ID Data: Together these fields represent a variable-length text string. The Node ID Length is a 16-bit unsigned integer indicating the number of octets of Node ID Data to follow. A zero-length Node ID SHALL be used to indicate the lack of Node ID rather than a truly empty Node ID. This case allows an entity to avoid exposing Node ID information on an untrusted network. A non-zero-length Node ID Data SHALL contain the UTF-8 encoded Node ID of the Entity which sent the SESS_INIT message. Every Node ID SHALL be a URI consistent with the requirements of [RFC3986] and the URI schemes of the IANA "Bundle Protocol URI Scheme Type" registry [IANA-BUNDLE]. The Node ID itself can be authenticated as described in Section 4.4.3.

Session Extension Length and Session Extension Items: Together these fields represent protocol extension data not defined by this specification. The Session Extension Length is the total number of octets to follow which are used to encode the Session Extension Item list. The encoding of each Session Extension Item is within a consistent data container as described in Section 4.8. The full set of Session Extension Items apply for the duration of the TCPCL session to follow. The order and multiplicity of these Session Extension Items is significant, as defined in the associated type specification(s). If the content of the Session Extension Items data disagrees with the Session Extension Length (e.g., the last Item claims to use more octets than are present in the Session Extension Length), the reception of the SESS_INIT is considered to have failed.

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

4.7. Session Parameter Negotiation

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

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

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

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

4.8. Session Extension Items

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

The fields of the Session Extension Item are:

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

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

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

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specifications SHOULD avoid the use of large data lengths, as no bundle transfers can begin until the full extension data is sent.

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

Figure 20: Session Extension Item Format

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

Table 3: Session Extension Item Flags

5. Established Session Operation

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

5.1. Upkeep and Status Messages

5.1.1. Session Upkeep (KEEPALIVE)

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

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

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

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

If no message (KEEPALIVE or other) has been received in a session after some implementation-defined time duration, then the entity SHALL terminate the session by transmitting a SESS_TERM message (as described in Section 6.1) with reason code "Idle Timeout". If configurable, the idle timeout duration SHOULD be no shorter than twice the keepalive interval. If not configurable, the idle timeout duration SHOULD be exactly twice the keepalive interval.

5.1.2. Message Rejection (MSG_REJECT)

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

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

The format of a MSG_REJECT message is as follows in Figure 21.

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

Figure 21: Format of MSG_REJECT Messages

The fields of the MSG_REJECT message are:

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

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

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

Table 4: MSG_REJECT Reason Codes

5.2. Bundle Transfer

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

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one entity to the other. In TCPCL a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU (see Section 4.2). The choice of the length to use for segments is an implementation matter, but each segment MUST NOT be larger than the receiving entity's maximum receive unit (MRU) (see the field Segment MRU of Section 4.2). The first segment for a bundle is indicated by the 'START' flag and the last segment is indicated by the 'END' flag.

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

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

5.2.1. Bundle Transfer ID

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

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

Although there is not a strict requirement for Transfer ID initial values or ordering (see Section 8.12), in the absence of any other mechanism for generating Transfer IDs an entity SHALL use the following algorithm: The initial Transfer ID from each entity is zero and subsequent Transfer ID values are incremented from the prior Transfer ID value by one.

5.2.2. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message follows in Figure 22.

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

Figure 22: Format of XFER_SEGMENT Messages

The fields of the XFER_SEGMENT message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

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

Transfer Extension Length and Transfer Extension Items: Together these fields represent protocol extension data for this specification. The Transfer Extension Length and Transfer Extension Item fields SHALL only be present when the 'START' flag is set to 1 on the message. The Transfer Extension Length is the total number of octets to follow which are used to encode the Transfer Extension Item list. The encoding of each Transfer Extension Item is within a consistent data container as described in Section 5.2.5. The full set of transfer extension items apply only to the associated single transfer. The order and multiplicity of these transfer extension items is significant, as defined in the associated type specification(s). If the content of the Transfer Extension Items data disagrees with the Transfer Extension Length (e.g., the last Item claims to use more octets than are present in the Transfer Extension Length), the reception of the XFER_SEGMENT is considered to have failed.

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

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

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

Table 5: XFER_SEGMENT Flags

The flags portion of the message contains two flag values in the two low-order bits, denoted 'START' and 'END' in Table 5. The 'START' flag SHALL be set to 1 when transmitting the first segment of a transfer. The 'END' flag SHALL be set to 1 when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the 'START' and 'END' flags SHALL be set to 1.

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

5.2.3. Data Acknowledgments (XFER_ACK)

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

The format of an XFER_ACK message follows in Figure 23.

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

Figure 23: Format of XFER_ACK Messages

The fields of the XFER_ACK message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

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

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

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

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

5.2.4. Transfer Refusal (XFER_REFUSE)

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

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

A transfer receiver MAY send an XFER_REFUSE message as soon as it receives any XFER_SEGMENT message. The transfer sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

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

The format of the XFER_REFUSE message is as follows in Figure 24.

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

Figure 24: Format of XFER_REFUSE Messages

The fields of the XFER_REFUSE message are:

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

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

Name	Code	Description
Unknown	0x00	Reason for refusal is unknown or not specified.
Completed	0x01	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
No Resources	0x02	The receiver’s resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	0x03	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.
Not Acceptable	0x04	Some issue with the bundle data or the transfer extension data was encountered. The sender SHOULD NOT retry the same bundle with the same extensions.
Extension Failure	0x05	A failure processing the Transfer Extension Items has occurred.
Session Terminating	0x06	The receiving entity is in the process of terminating the session. The sender MAY retry the same bundle at a later time in a different session.

Table 6: XFER_REFUSE Reason Codes

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

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

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

5.2.5. Transfer Extension Items

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

The fields of the Transfer Extension Item are:

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

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

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

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specifications SHOULD avoid the use of large data lengths, as the associated transfer cannot begin until the full extension data is sent.

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

Figure 25: Transfer Extension Item Format

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

Table 7: Transfer Extension Item Flags

5.2.5.1. Transfer Length Extension

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

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

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

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

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

Figure 26: Format of Transfer Length data

The fields of the Transfer Length extension are:

Total Length: A 64-bit unsigned integer indicating the size of the data-to-be-transferred. The Total Length field SHALL be treated as authoritative by the receiver. If, for whatever reason, the actual total length of bundle data received differs from the value indicated by the Total Length value, the receiver SHALL treat the transmitted data as invalid and send an XFER_REFUSE with a Reason Code of "Not Acceptable".

6. Session Termination

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

6.1. Session Termination Message (SESS_TERM)

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

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

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

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

The format of the SESS_TERM message is as follows in Figure 27.

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

Figure 27: Format of SESS_TERM Messages

The fields of the SESS_TERM message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 8. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

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

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

Table 8: SESS_TERM Flags

Name	Code	Description
Unknown	0x00	A termination reason is not available.
Idle timeout	0x01	The session is being terminated due to idleness.
Version mismatch	0x02	The entity cannot conform to the specified TCPCL protocol version.
Busy	0x03	The entity is too busy to handle the current session.
Contact Failure	0x04	The entity cannot interpret or negotiate a Contact Header or SESS_INIT option.
Resource Exhaustion	0x05	The entity has run into some resource limit and cannot continue the session.

Table 9: SESS_TERM Reason Codes

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

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

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

6.2. Idle Session Shutdown

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

If there is a configured time to terminate idle sessions and if no TCPCL messages (other than KEEPALIVE messages) has been received for at least that amount of time, then either entity MAY terminate the session by transmitting a SESS_TERM message indicating the reason code of "Idle timeout" (as described in Table 9).

7. Implementation Status

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to [RFC7942] and [github-dtn-bpbis-tcpcl].]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations can exist.

An example implementation of the this draft of TCPCLv4 has been created as a GitHub project [github-dtn-bpbis-tcpcl] and is intended to use as a proof-of-concept and as a possible source of interoperability testing. This example implementation uses D-Bus as the CL-BP Agent interface, so it only runs on hosts which provide the Python "dbus" library.

8. Security Considerations

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

8.1. Threat: Passive Leak of Node Data

When used without TLS security, the TCPCL exposes the Node ID and other configuration data to passive eavesdroppers. This occurs even when no transfers occur within a TCPCL session. This can be avoided by always using TLS, even if authentication is not available (see Section 8.11).

8.2. Threat: Passive Leak of Bundle Data

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

When used without TLS security, the TCPCL exposes all bundle data to passive eavesdroppers. This can be avoided by always using TLS, even if authentication is not available (see Section 8.11).

8.3. Threat: TCPCL Version Downgrade

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

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

8.4. Threat: Transport Security Stripping

When security policy allows non-TLS sessions, TCPCL does not protect against active network attackers. It is possible for a man-in-the-middle attacker to set the CAN_TLS flag to 0 on either side of the Contact Header exchange. This leads to the "SSL Stripping" attack described in [RFC7457].

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

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

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

8.5. Threat: Weak TLS Configurations

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

8.6. Threat: Untrusted End-Entity Certificate

The profile in Section 4.4.3 uses end-entity certificates chained up to a trusted root CA. During TLS handshake, either entity can send a certificate set which does not contain the full chain, possibly excluding intermediate or root CAs. In an environment where peers are known to already contain needed root and intermediate CAs there is no need to include those CAs, but this has a risk of an entity not actually having one of the needed CAs.

8.7. Threat: Certificate Validation Vulnerabilities

Even when TLS itself is operating properly an attacker can attempt to exploit vulnerabilities within certificate check algorithms or configuration to establish a secure TCPCL session using an invalid certificate. A BP agent treats the peer Node ID within a TCPCL session as authoritative and an invalid certificate exploit could lead to bundle data leaking and/or denial of service to the Node ID being impersonated. There are many reasons, described in [RFC5280], why a certificate can fail to validate, including using the certificate outside of its valid time interval, using purposes for which it was not authorized, or using it after it has been revoked by its CA. Validating a certificate is a complex task and can require network connectivity outside of the primary TCPCL network path(s) if a mechanism such as the Online Certificate Status Protocol (OCSP) [RFC6960] is used by the CA. The configuration and use of particular certificate validation methods are outside of the scope of this document.

8.8. Threat: Symmetric Key Limits

Even with a secure block cipher and securely-established session keys, there are limits to the amount of plaintext which can be safely encrypted with a given set of keys as described in [AEAD-LIMITS]. When permitted by the negotiated TLS version (see [RFC8446]), it is advisable to take advantage of session key updates to avoid those limits.

8.9. Threat: BP Node Impersonation

The certificates exchanged by TLS enable authentication of peer DNS name and Node ID, but it is possible that a peer either not provide a valid certificate or that the certificate does not validate either the DNS name or Node ID of the peer (see Section 3.4). Having a CA-validated certificate does not alone guarantee the identity of the network host or BP node from which the certificate is provided; additional validation procedures in Section 4.4.2 bind the DNS name or Node ID based on the contents of the certificate.

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

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

8.10. Threat: Denial of Service

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

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

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

There is the possibility of a "data dribble" attack in which an entity presents a very small Segment MRU which causes transfers to be split among an large number of very small segments and causes the segmentation overhead to overwhelm the actual bundle data segments. Similarly, an entity can present a very small Transfer MRU which will cause resources to be wasted on establishment and upkeep of a TCPCL session over which a bundle could never be transferred. The victim entity can terminate the session during the negotiation of Section 4.7 if the MRUs are unacceptable.

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

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

8.11. Alternate Uses of TLS

This specification makes use of PKIX certificate validation and authentication within TLS. There are alternate uses of TLS which are not necessarily incompatible with the security goals of this specification, but are outside of the scope of this document. The following subsections give examples of alternate TLS uses.

8.11.1. TLS Without Authentication

In environments where PKI is available but there are restrictions on the issuance of certificates (including the contents of certificates), it may be possible to make use of TLS in a way which authenticates only the passive entity of a TCPCL session or which does not authenticate either entity. Using TLS in a way which does not successfully authenticate some claim of both peer entities of a TCPCL session is outside of the scope of this document but does have similar properties to the opportunistic security model of [RFC7435].

8.11.2. Non-Certificate TLS Use

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

8.12. Predictability of Transfer IDs

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

9. IANA Considerations

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

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

9.1. Port Number

Within the port registry of [IANA-PORTS], TCP port number 4556 has been previously assigned as the default port for the TCP convergence layer in [RFC7242]. This assignment is unchanged by TCPCL version 4, but the assignment reference is updated to this specification. Each TCPCL entity identifies its TCPCL protocol version in its initial contact (see Section 9.2), so there is no ambiguity about what protocol is being used. The related assignments for UDP and DCCP port 4556 (both registered by [RFC7122]) are unchanged.

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

Table 10

9.2. Protocol Versions

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

Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4	TCPCLv4	This specification.
5-255	Unassigned	

Table 11

9.3. Session Extension Types

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

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Session Extension Types" and initialize it with the contents of Table 12. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

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

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

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

Table 12: Session Extension Type Codes

9.4. Transfer Extension Types

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

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Transfer Extension Types" and initialize it with the contents of Table 13. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

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

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

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

Table 13: Transfer Extension Type Codes

9.5. Message Types

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

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

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

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

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

Table 14: Message Type Codes

9.6. XFER_REFUSE Reason Codes

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

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" and initialize it with the contents of Table 15. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

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

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

Code	Refusal Reason
0x00	Unknown
0x01	Completed
0x02	No Resources
0x03	Retransmit
0x04	Not Acceptable
0x05	Extension Failure
0x06	Session Terminating
0x07--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 15: XFER_REFUSE Reason Codes

9.7. SESS_TERM Reason Codes

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

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 SESS_TERM Reason Codes" and initialize it with the contents of Table 16. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

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

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

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

Table 16: SESS_TERM Reason Codes

9.8. MSG_REJECT Reason Codes

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

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 MSG_REJECT Reason Codes" and initialize it with the contents of Table 17. The registration procedure is Specification Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

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

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

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

Table 17: MSG_REJECT Reason Codes

10. Acknowledgments

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

11. References

11.1. Normative References

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

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

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

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

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

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

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

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

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

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

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

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

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

[I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", Work in Progress, Internet-Draft, draft-ietf-dtn-bpbis-27, 27 October 2020, <https://tools.ietf.org/html/draft-ietf-dtn-bpbis-27>.

11.2. Informative References

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

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

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

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

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

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

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

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

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

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

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

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

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

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

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

[I-D.ietf-dtn-bpsec]
Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", Work in Progress, Internet-Draft, draft-ietf-dtn-bpsec-23, 25 October 2020, <https://tools.ietf.org/html/draft-ietf-dtn-bpsec-23>.

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

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

Appendix A. Significant changes from RFC7242

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

* Split Contact Header into pre-TLS protocol negotiation and SESS_INIT parameter negotiation. The Contact Header is now fixed-length.

* Changed Contact Header content to limit number of negotiated options.

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

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

* Added session extension capability.

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

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

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

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

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

* Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.

* Removed all uses of SDNV fields and replaced with fixed-bit-length (network byte order) fields.

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

* Removed the notion of a re-connection delay parameter.

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

* Added contact negotiation failure SESS_TERM reason code.

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

* Added TLS connection security mechanism.

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

* Added "Resource Exhaustion" SESS_TERM reason code.

Authors' Addresses

Brian Sipos
RKF Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
United States of America

Email: BSipos@rkf-eng.com

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

Email: demmer@cs.berkeley.edu

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

Email: ott@in.tum.de

Simon Perreault
Quebec QC
Canada

Email: simon@per.reau.lt