Delay Tolerant Networking Research Group
Internet-Draft
Intended status: Experimental
Expires: October 17, 2011

S. Parikh
S. Symington
K. Scott
R. Durst
R. Edell
The MITRE Corporation

April 15, 2011

Delay-Tolerant Networking Superseding Bundle Extension Block
draft-parikh-bundle-superseding-extension-block-01

Abstract

This document defines an optional Bundle Protocol block called the Superseding Bundle Extension Block (SBEB) for use with the Bundle Protocol in the context of the Delay-Tolerant Networking Architecture. Applications use this block to call for the removal of previously sent bundles that are rendered obsolete by more recent bundles. Upon receiving a bundle with an SBEB, a node will search its bundle store (and outbound queues) for bundles that are obsoleted by other related bundles according to their source and destination EID, and the SBEB cookie (if present), and may delete some or all of them. Discarding obsolete bundles helps conserve storage space and prevents expending resources in further forwarding bundles that are no longer relevant. The bundle protocol already uses expiration times to remove bundles that are no longer useful to applications. The SBEB is a way for applications to mark bundles that a node may delete prior to their expiration times.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 17, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document defines an optional Bundle Protocol block called the Superseding Bundle Extension Block (SBEB) for use with the Bundle Protocol [RFC5050] in the context of the DTN Architecture [RFC4838]. Applications use this block to call for the removal of previously sent bundles that are rendered obsolete by more recent bundles. Upon receiving a bundle with an SBEB, a node will search its bundle store (and outbound queues) for bundles that are obsoleted by other related bundles according to their source and destination EID and the SBEB cookie (if present), and may delete some or all of them. Thus, a bundle with an SBEB can trigger the flushing of bundles from nodes as it traverses the network. A newly arriving bundle might render obsolete an older bundle that is currently in the store, and the node may delete it. If an older bundle arrives after a newer bundle (possible because of out-of- sequence arrivals), then the node may delete the older (but most recently arrived) bundle. The SBEB includes a retention count that instructs the node to retain a number (not just one) of the newest bundles according to their timestamp. Discarding obsolete bundles helps conserve storage space and prevents expending resources in further forwarding bundles that are no longer relevant. The bundle protocol already uses expiration times to remove bundles that are no longer useful to applications. The SBEB is a way for applications to mark bundles that a node may delete prior to their expiration times. However, this extension is not intended as a reliable mechanism for applications to eliminate bundles in the network; it is intended to assist in reducing the load on DTN nodes and the network. The capabilities described in this document are optional for deployment with the Bundle Protocol. The use of this extension block is voluntary by applications, and even bundle nodes that support it can choose to ignore it. To support this extension, a bundle protocol agent must be capable of receiving bundles with this extension block and processing them as described in this document.

2. Motivation

This extension is motivated by applications that send "cyclic telemetry", that is, bundles containing information that supplants information in previously sent bundles.

The primary example comes from applications that distribute status updates. Consider a server that tracks the location of vehicles and distributes that information to clients over a network of DTN nodes. The server sends a bundle containing the location of a particular vehicle when that vehicle reports its position to the server. The clients only want the most recent location of each vehicle; the

history of the vehicle's location is not useful information (in this particular example). Therefore, if multiple location updates are present in a node's bundle store, the node need only forward the most recent one. By using SBEB, the server can send bundles that will replace older bundles along the path to the destination. In this example, the application may use an EID that only identifies the source application, not the vehicle to which the bundle pertains (that is, it uses the same source EID regardless of the particular vehicle). The payload will contain an identifier of the vehicle and its location; thus, a node will need additional information to identify bundles that pertain to a particular vehicle, because nodes do not examine bundle payloads. The cookie field in the extension block can contain an identifier for the vehicle, and the node can search the bundle store for bundles with that cookie (in addition to the source and destination).

A motivating case for retaining more than one matching bundle comes from a camera system that monitors highway traffic, intended for commuters to check roadways that are prone to gridlock or accidents, so they can plan their trip accordingly. The camera sends a snapshot bundle once every minute to a collecting server over a network of DTN nodes. The server then posts that image on a website which commuters check over the Internet (the cameras are not on the Internet). The expiration time of bundles containing the snapshots are conservatively set to 10 minutes to give the image sufficient lifetime to traverse the challenged network of DTN nodes. Suppose that an intermittent link in the challenged network results in snapshot bundles accumulating at a node. Only the five (for the sake of example) most recent photos are useful; previous photos in the bundle store are not. This gives an observer enough pictures to gauge the flow of traffic (something that might be difficult for the camera to compute), but a complete historical record of all images is not needed. The SBEB instructs the node to retain only the freshest images and delete the others, thereby releasing storage resources and preventing the further transmittal of out-of-date bundles. In this scenario (in contrast to the previous example), since each camera would have a unique source EID, the tuple of source EID, destination EID, and timestamp would be sufficient to identify obsolete bundles in a node's bundle store; no additional cookie is needed.

Note that DTN nodes do not guarantee that bundles arrive at a node in order of timestamp. Therefore, a node might discard the newly arrived bundle if a more recent one is already present in the bundle store. The search only applies to bundles that are currently in the bundle store when the node acts in response to the SBEB. The node does not track or consider the timestamps of bundles that have already departed the node.

In some ways, the SBEB serves as a congestion control guidance mechanism for DTN nodes. It provides applications a way to advise which bundles a node can remove to help conserve storage, and in this respect it supplements the bundle expiration time. However, the goal of the SBEB is not to provide applications a way to reliably rid the network of stale bundles. If storage and link resources are plentiful, nodes may choose to ignore the SBEB; applications should not depend on nodes to respond to the SBEB.

3. Superseding Bundle Extension Block Format

The SBEB Block uses the Canonical Bundle Block Format as defined in the Bundle Protocol [2], using the block layout without an EID Reference List. That is, it is comprised of the following elements:

1. Block-type code (1 byte): Defined as in all bundle protocol blocks except the primary bundle block (as described in the Bundle Protocol). The extension block type for SBEBs is [XXX_TO_BE_ASSIGNED_BY_IANA_XXX -- see Section 6]:

2. Block processing control flags (SDNV): Defined as in all bundle protocol blocks except the primary bundle block. SDNV encoding is described in the Bundle Protocol.

   * Bit 0 ("Block must be replicated in every fragment") SHOULD be set.

   * Bit 2 ("Delete bundle if block can't be processed") MUST NOT be set.

   * Bit 4 ("Discard block if it can't be processed") MUST NOT be set.

   * Bit 6 ("Block contains an EID-reference field") MUST NOT be set.

3. Block EID references count and EID references: This field MUST NOT be present.

4. Block data length (SDNV): defined as in all bundle protocol blocks except the primary bundle block. SDNV encoding is described in the bundle protocol.

5. Block-type-specific data fields as follows:

SBEB Flags (SFLAGS, byte) A byte of flags indicating whether or not an SBEB Cookie is preent, and the type of SBEB:

Bit 0: Cookie Present: Indicates the presence and type of SBEB cookie field contained in the current block:

0: No cookie field present

1: Cookie field present

Bit 1: IsSigned Indicates whether or not this SBEB contains a signature of the canonicalized version of the primary bundle block plus the SBEB.

0: No hash present

1: Hash present

Bits 3-2: SBEB Type Indicates the type operation to be done to determine which bundles to keep / discard:

0 Supersede by freshness -- keep the most recent N bundles.

1 Keep fixed time window -- keep all bundles with creation timestamps >= the creation timestamp of the current bundle minus N seconds.

2 Supersede by sequence vector

3 Reserved for future use

Bits 7-3: Reserved Reserved for future use.; these bits SHOULD b set to 0 on send and ignored on receipt.

SBEB Cookie (SDNV) If the Cookie Present bit of the SBEB flags indicates that an SBEB Cookie is present, then the SBEB Cookie MUST be included and MUST follow the SBEB Flags. The SBEB Cookie is an opaque token used by forwarders to match the SBEB against those of other bundles the routers hold. If the Cookie Present value is 0 (No cookie field present) then the SBEB Cookie MUST NOT be present.

SBEB SigLen (SDNV) If the 'IsSigned' bit of the SBEB Flags indicates that a signed hash is present, the length of that signed hash (the following field) appears here, encoded as an SDNV.

SBEBSignature: If the 'IsSigned' bit of the SBEB Flags indicates that a signed hash is present, that signed hash appears here. The signing mechanism used is RSA with SHA256 as specified for the id-sha256 PKCSv2.1 signature scheme in [RFC4055]. The hash is computed over the following fields of the Primary Bundle Block, in the order below with no padding in between:

+ Version

+ Source scheme

+ Source scheme-specific part

+ Creation timestamp time

+ Creation timestamp sequence number

+ Destination scheme

+ Destination scheme specific part

followed by the SBEB block (this block) itself, with the signature-related fields (SBEB SigLength and SBEBSignatures) fields of the SBEB omitted, followed by any required padding bytes as zeros; the hash is then signed and the result of the signing process becomes this field. Nodes that process SBEBs are expected to be able to determine the correct signing key from the other fields of the bundle (such as the source EID).

An alternate way to compute the hash could be to compute it over the concatenation of the mutable canonicalization of the bundle's Primary Bundle Block as defined in [I-D.irtf-dtnrg-bundle-security] section 3.4.2, followed by the SBEB block (this block) itself, with the signature-related fields (SBEB SigLength and SBEBSignatures) fields of the SBEB omitted; the result of the signing process becomes this field. Nodes that process SBEBs are expected to be able to determine the correct signing key from the other fields of the bundle (such as the source EID). Using only the fields listed above doesn't allow for reuse of other code to manage mutable cononicalization of PBBs, but should be simpler.

The rest of the fields of an SBEB differ depending on the value of the SBEB Type field.

If the SBEB Type is 0 or 1, the last field of the SBEB is:

Retention Field (SDNV) For SBEB Type 0 (keep the most recent N bundles) this indicates the number of newest (by creation timestamp) matching bundles that a node MUST retain (if as many bundles are indeed available). The value must be at least one for a node to search for and discard bundles in response to the SBEB; thus, a value greater than zero serves to activate an SBEB. An application uses a value of zero on bundles that might later be subject to removal by a future bundle, but that do not themselves trigger the removal of bundles; this allows the application control over when to activate the discarding of older bundles.

For SBEB Type 1 (keep bundles from the most recent N seconds) the retention field in conjunction with the bundle creation time indicates the threshold beyond which bundles should be retained. Thus a router processing a bundle with an SBEB-2 extension, a creation timestamp of M, and a Retention Field of N MUST retain all bundles with creation timestamps >= M-N. The value must be at least one for a node to search for and discard bundles in response to the SBEB; thus, a value greater than zero serves to activate an SBEB. An application uses a value of zero on bundles that might later be subject to removal by a future bundle, but that do not themselves trigger the removal of bundles; this allows the application control over when to activate the discarding of older bundles.

If the SBEB Type is 2, the remainder of the SBEB contains a sequence number for this bundle together with a list of sequence numbers that this bundle obsoletes:

SBEB Sequence Number (SDNV) The SBEB sequence number of this bundle as assigned by the application. Bundles with a particular (source, destination) pair SHOULD use monotonic non-decreasing values for their SBEB sequence numbers.

Obsoletes Up To A cumulative watermark sequence number of obsoleted bundles. This bundle obsoletes all matching bundles with SBEB sequence numbers up to and including this value. This value MUST be less than the SBEB sequence number of this bundle.

Obsoletes Vector Count (OVC, an SDNV) Count of the number of sequence numbers in the obsoletes vector.

Obsoletes List (sequence of SDNVs) Vector of obsoleted sequence numbers. Each element of the vector is a single sequence number that is obsoleted by this bundle. Each value in the obsoletes vector MUST be less than the SBEB sequence number of this bundle.

The Structure of a Superseding Bundle Extension Block is shown in Figure 1:

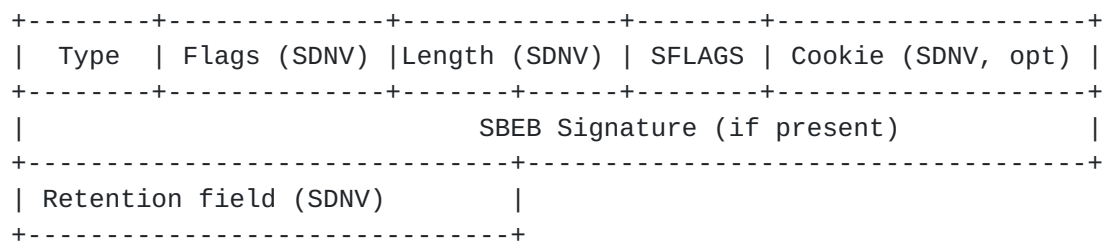

Figure 1: SBEB Format for SBEBS of type 0, 1

The structure of a Superseding Bundle Extension Block of type 2 is shown in Figure 2:

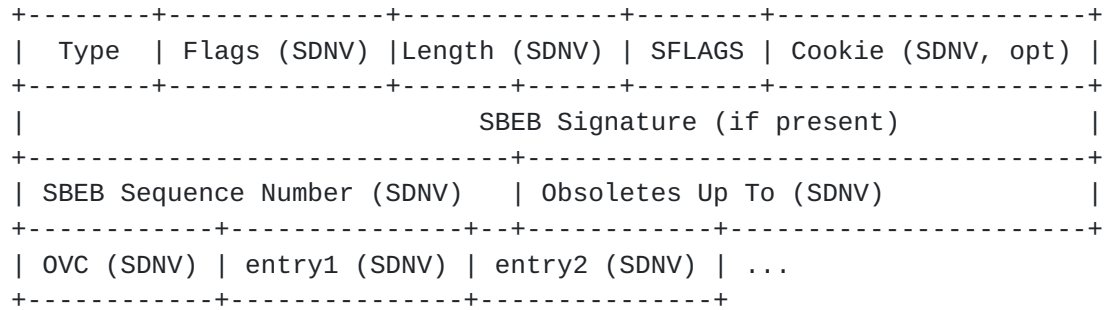

Figure 2: SBEB Format for SBEBS of type 2

4. Superseding Bundle Block Processing

The following are the processing steps that a bundle node must take relative to generation, reception, and processing of SBEB blocks.

4.1. Bundle Transmission

When an outbound bundle is created per the parameters of the bundle transmission request, this bundle MAY (as influenced by local policy) include one SBEB (as defined in this specification).

4.2. Bundle Forwarding

The node MUST NOT insert an SBEB into the bundle before forwarding it. The node MUST NOT modify any part of the SBEB. The node MUST NOT remove the SBEB before forwarding it.

4.3. Bundle Reception

SBEB processing is optional and may be controlled by policy at a node.

If a received bundle is complete (i.e. is not a fragment) and includes an SBEB or upon reassembly of fragments into a complete bundle that includes an SBEB, and if the receiving node chooses to process the SBEB, the node MUST process it as follows. Upon receipt of a bundle with an SBEB, the node searches for other bundles awaiting forwarding that match the current bundle SBEB criteria. Matching bundles are those that:

- do not have the current node as their current custodian,
- themselves contain SBEBs with identical SBEB Flags fields as the one being processed, and
- are members of the same source endpoint as the current bundle, and
- are members of the same destination endpoint as the current bundle, and
- either contain no cookie if the current SBEB does not contain a cookie or else contain the same SBEB cookie as the present bundle, and
- either contain no signature if the current SBEB does not have a signature or else contains a valid SBEB Signature.

Determining which of the matching bundles are retained and which are discarded depends on the SBEB type:

4.3.1. SBEB Type 0

The value of the SBEB Retention field identifies the number of matching bundles, including the current one, that the router should retain. The value N used is that of the most recent (by creation timestamp and timestamp sequence number) matching bundle's SBEB retention field value. The most recent N (by creation timestamp and creation timestamp sequence number) complete bundles MUST be retained. Other matching bundles that are complete SHOULD be discarded. Matching fragments SHOULD be discarded if their creation timestamps and creation timestamp sequence numbers are such that they would have been discarded had they been complete at the time of SBEB processing. Matching bundle fragments with (creation timestamp, creation timestamp sequence number) values more recent that the oldest retained complete bundle MUST be retained.

4.3.2. SBEB Type 1

The value of the SBEB Retention field identifies a time threshold equal to the value of the current bundle's creation timestamp minus the SBEB Retention field value. The value N used is that of the most recent (by creating timestamp and timestamp sequence number) matching bundle's SBEB retention field value. Matching bundles (complete or fragments) with creation timestamps equal to or after this threshould MUST be retained; bundles with creation timestamps before the threshold SHOULD be discarded.

4.3.3. SBEB Type 2

Bundles are retained based on their SBEB sequence numbers as follows. Bundles (complete or fragments) with sequence numbers less than or equal to the value of the 'Obsoletes Up To' field SHOULD be discarded. Bundles with SBEB sequence numbers that are in the 'Obsoletes List' SHOULD be discarded. All other bundles MUST be retained.

5. Interactions with Bundle Protocol Features

When the SBEB results in a node deleting a bundle, the node MAY generate a bundle deletion status report, as described in the Bundle Protocol specification.

If the node implements sequenced outbound queing and if SBEB processing obsoletes a bundle that is queued for transmission, the obsoleting bundle MUST take the obsoleted bundle's place in the queue. This prevents the situation where replacing bundles are always sent to the tail of the queue, causing a condition where bundles never depart from a node.

6. IANA Considerations

This document requests that IANA assign a Bundle Block Type Code for SBEB blocks from the unassigned pool of the Bundle Bock Type Codes DTN Registry [I-D.irtf-dtnrg-bundle-security] section 3.1.

Note to RFC Editor: once the BP registry is established and a block type code has been assigned and its value inserted into Section 3 of this document, this section may be removed on publication as an RFC.

7. Security Considerations

Because the SBEB is not included in the Mutable Canonicalization of the bundle, its end-to-end integrity is not ensured by the use of the mandatory Payload Integrity Block ciphersuite defined in the BSP. Because the SBEB would be included in the Strict Canonicalization of the bundle, its integrity would be ensured between one bundle protocol agent and its adjacent bundle protocol agent, providing the Bundle Authentication Block is used to protect the bundle during that single DTN hop.

SBEBs could be encrypted using the Extension Security Block mechanism of the BSP, but 1) the mandatory ciphersuite for ESBs is the symmetric RSA-AES-128-EXT ciphersuite which uses symmetric keys that would need to be known to large portions of the network if SBEBs were to be processed and 2) even without the ESB key, a bad actor with access to a single bundle containing an encrypted SBEB could (if it guessed the SBEB-containing ESB) copy the encrypted SBEB onto 'bogus' bundles and flood the network, crowding out real bundles. This last attack could be prevented by having the source sign SBEBs.

8. Acknowledgements

9. References

9.1. Normative References (Note: the IDs listed here are in the RFC editor's queue; this document depends on them; replace with appropriate RFC references.)

[I-D.irtf-dtnrg-bundle-security]
Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", draft-irtf-dtnrg-bundle-security-19 (work in progress), March 2011.

[I-D.irtf-dtnrg-iana-bp-registries]
Blanchet, M., "Delay-Tolerant Networks (DTN) Bundle Protocol IANA Registries", draft-irtf-dtnrg-iana-bp-registries-02 (work in progress), February 2011.

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

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in

the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

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

9.2. Informative References

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

Authors' Addresses

Salil Parikh
The MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102
USA

Phone: +1 (703) 983-3560
Email: sparikh@mitre.org

Susan Symington
The MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102
USA

Phone: +1 (703) 983-7209
Email: susan@mitre.org

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

Phone: +1 (703) 983-6547
Email: kscott@mitre.org

Robert Durst
The MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102
USA

Phone: +1 (703) 983-7535
Email: durst@mitre.org

Richard Edell
The MITRE Corporation
7515 Colshire Drive
McLean, Virginia 22102
USA

Phone: +1 (703) 983-7535
Email: redell@mitre.org