DTN Erasure Coding Extension Block Specification

IETF 81: DTNRG Presentation (Extended)

Dr. John Zinky
Dr. Armando Caro
Dr. Greg Stein
jzinky@bbn.com
acaro@bbn.com
gstein@ece.umd.edu

July 25, 2011

Raytheon
BBN Technologies

Outline

Raytheon
BBN Technologies

- **Erasure Coding Context**
- Erasure Coding Architecture
- Encoding Process for Random Binary Code
- Erasure Coding Headers
- End-to-End File Transfer
- End-to-End Bundle Transfer
- Proof of Concept Implementation Results
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

Problem Statement:

Raytheon
BBN Technologies

- **Transfer a large file** over a disrupted network.
- **Short contact times** between BPA rendezvous may not be long enough to send the whole file.
  - The sender must partition the file, which may be sent over multiple DTN paths.
  - The partitions can be *shuffled, duplicated, and dropped* along the DTN path.
  - The application transfer specification may imposed limitations on the amount of the file that can be required for each contact, link, BPA, or path, to meet quality of service (QoS) and content exposure requirements.
- The file sender needs **NO feedback** about the file transfer.
  - No E2E acknowledgement is expected.
  - Reliable transfer depends on existing DTN reliability and timeout mechanisms.
  - Open-Loop flow control depends on existing DTN buffering.
  - Feedback-based flow and congestion control needs to define an additional control protocol not included in this specification.
    - e.g. End-to-End *Stop*, Store-and-Forward *Purge*

Eraser Coding Extension Protocol:

Raytheon
BBN Technologies

1. Erasure Coding Extension Protocol partitions the file into multiple chunks and encodes the chunks using one of many **Erasure Correction** schemes. The encoded bundles can be further partitioned using the existing bundle fragmentation protocol.
2. Eraser Coding Extension Protocol can send a **linear combination** of file chunks and the receiver recovers the original chunks by solving the resulting set of equations.
3. Erasure Coding effectively makes **every bundle equivalent**, removing dependency on bundle delivery order, drop, or duplication.
4. File content is **fixed size, known before the transfer begins, and every bit must be delivered within the timeout** or the transfer has no mission utility. The file is not a stream, nor are some parts of the file more important than other parts, i.e. every bit of the file has the same reliability and latency requirements.

NOTE: The term, ***File***, refers to a fixed sized application data block that a DTN application needs to send. It may or may not actually be a file on the sending host's file system.

Goals:

Raytheon
BBN Technologies

1. The Erasure Coding Extension Protocol should be **transparent to legacy** BPAs and DTN Applications.
   - The existing fields in the bundle header will work as defined, with no extended functionality.
   - Bundle-layer routing, timeouts, bundle uniqueness, priority, and fragmentation, will work as specified in RFC 5050.
2. **Only Expose information** to protocol layers that can act on the information
   - Set BPA header fields for effective End-to-End transfer characteristics.
   - Expose a bundle extension block with meta data that will increase effectiveness of Store-and Forward transfer across a DTN network.
   - File meta data for a destination is only encoded in the file payload.
3. BPA-layer extensions **increase store-and-forward efficiency** by supporting different transfer options:
   - Routing: multiple destination types: unicast, multicast, geographic.
   - Session: detect duplicate or redundant encodings.
   - QoS : rate limit, redundancy limit, end-to-end *stop*, store and forward *purge.*

Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- **Erasure Coding Architecture**
- Encoding Process for Random Binary Code
- Erasure Coding Headers
- End-to-End File Transfer
- End-to-End Bundle Transfer
- Proof of Concept Implementation Results
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

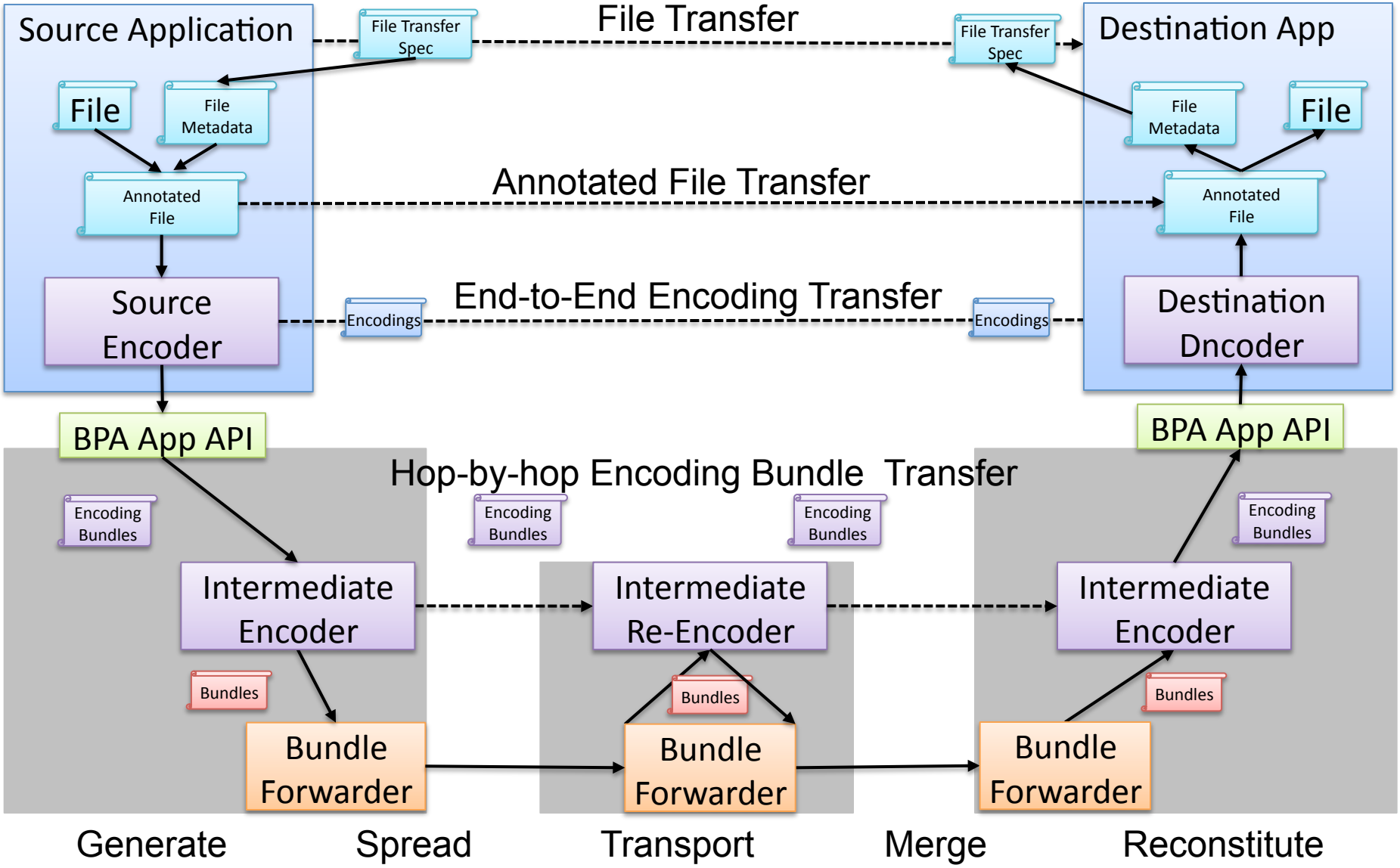
Architectural Components

Raytheon
BBN Technologies

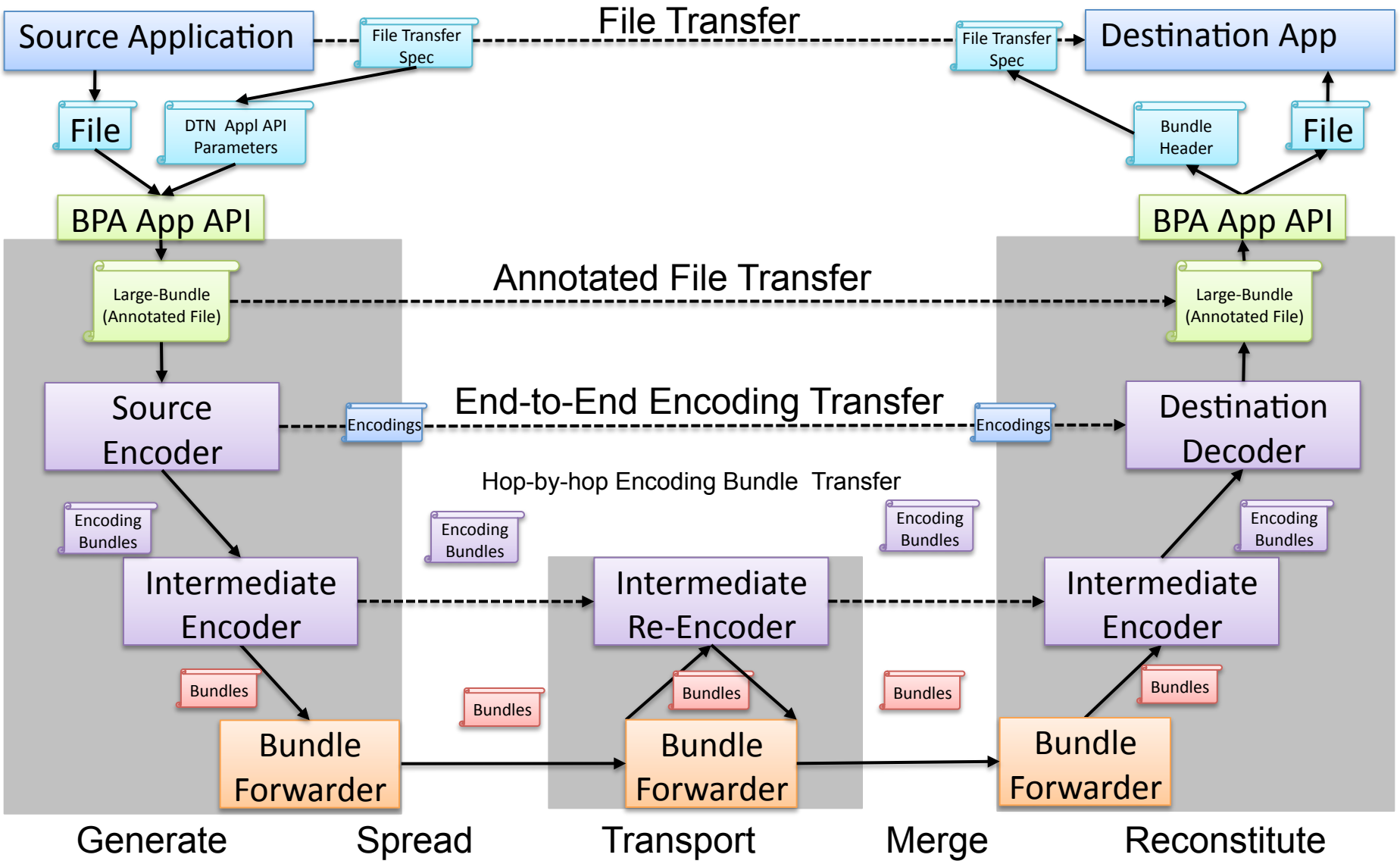
- Source Application
  - Originates a ***File*** for transmission
- Destination Application
  - Receives the *File*
- Source Encoder
  - Encodes an ***Annotated File*** into a set of Encodings for transmission
- Destination Decoder
  - Decodes a set of ***Encoding*** to retrieve the *Annotated File*
- Intermediate Encoder
  - Change order, count, duplicate, drop, route **Encoding Bundles**
- Intermediate Re-Encoder
  - Create new encodings from existing encodings without decoding
- Bundle Forwarder
  - Knows nothing about Encodings, just forwards bundles

NOTE: The term, ***File****,* refers to a fixed sized application data block that a DTN application needs to send. It may or may not actually be a file on the sending host's file system.

Encoding Application Architecture



Large-Bundle BPA Encoder Architecture



Transfer Specification

File Transfer Spec gets translated onto Bundle Fields and Erasure Coding Extension Block

File Transfer Spec could include:

- Destination
- End-to-End File Transfer Timeout
- Progress Reports
- Privacy
- Efficiency
- Resource consumption limits

Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- Erasure Coding Architecture
- **Encoding Process for Random Binary Code**
- Erasure Coding Headers
- End-to-End File Transfer
- End-to-End Bundle Transfer
- Proof of Concept Implementation Results
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

Decomposition for File into Bundles

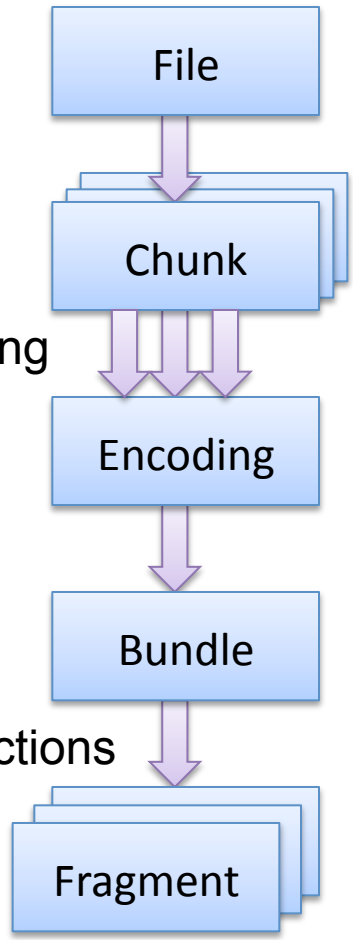
Raytheon BBN Technologies

File partitioned into N Chunks

Chunks Linearly Combined into an Encoding

One Encoding sent per Bundle

Bundle Fragment size based on CLA restrictions



Example:

128MB, Compressed

N=256
Length = 500KB

500KB Data
256bit Vector

500KB Payload
32byte Extension block

64KB (8 fragments)

File Transfer Characteristics:

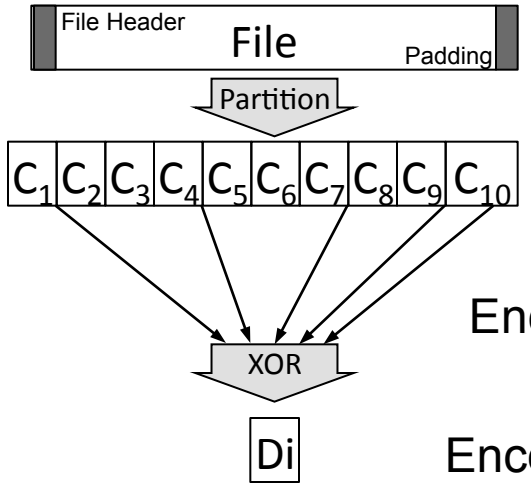
All bits must be received before timeout, or file transfer has no mission utility.
All bits are known before start of transfer.

Constructing an Encoding from Chunks

Raytheon BBN Technologies

Random Binary Code Example:

1. Partition the File and File Meta data header into N Chunks.
2. Pad the last Chunk with random bits, to make all Chunks the same length.
3. Generate an Encoding Vector.
   - 1 in i^{th} slot means include $Chunk_i$
   - 0 in i^{th} slot means exclude $Chunk_i$
4. XOR together included Chunks to form the Encoding Data.
   - This operation is called a Binary Dot Product of Encoding Vector with Chunks Vector
5. The resulting Encoding includes both the Vector and Data.



Exposed File Meta Data:
File UUID

Chunks (N=10)

1 0 0 1 0 0 1 0 1 1

Encoding Vector (V_i)

Encoding Data (D_i)

Fun Facts:

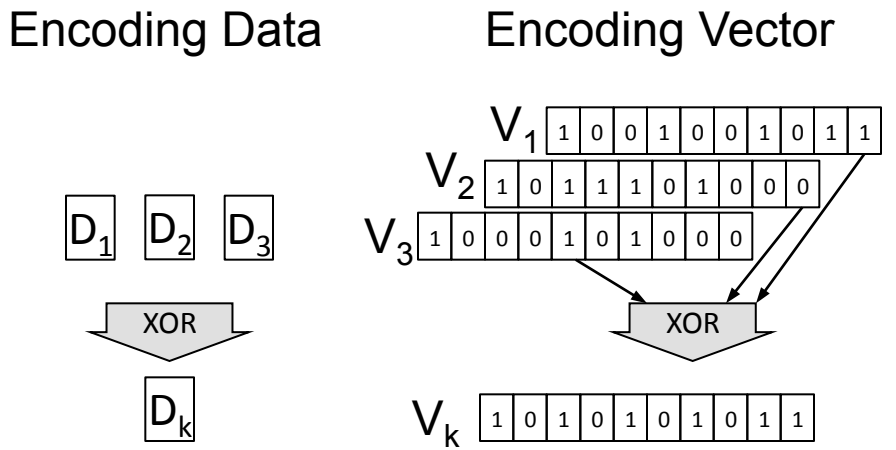
- Two Encodings are *equal*, if their vectors are equal.
- *Hamming Weight* is the number of 1's in the Encoding Vector.
- At least N Encodings are needed to reconstitute all the original Chunks

Constructing a new Encoding from Existing Encodings

Raytheon BBN Technologies

Random Binary Code Example:

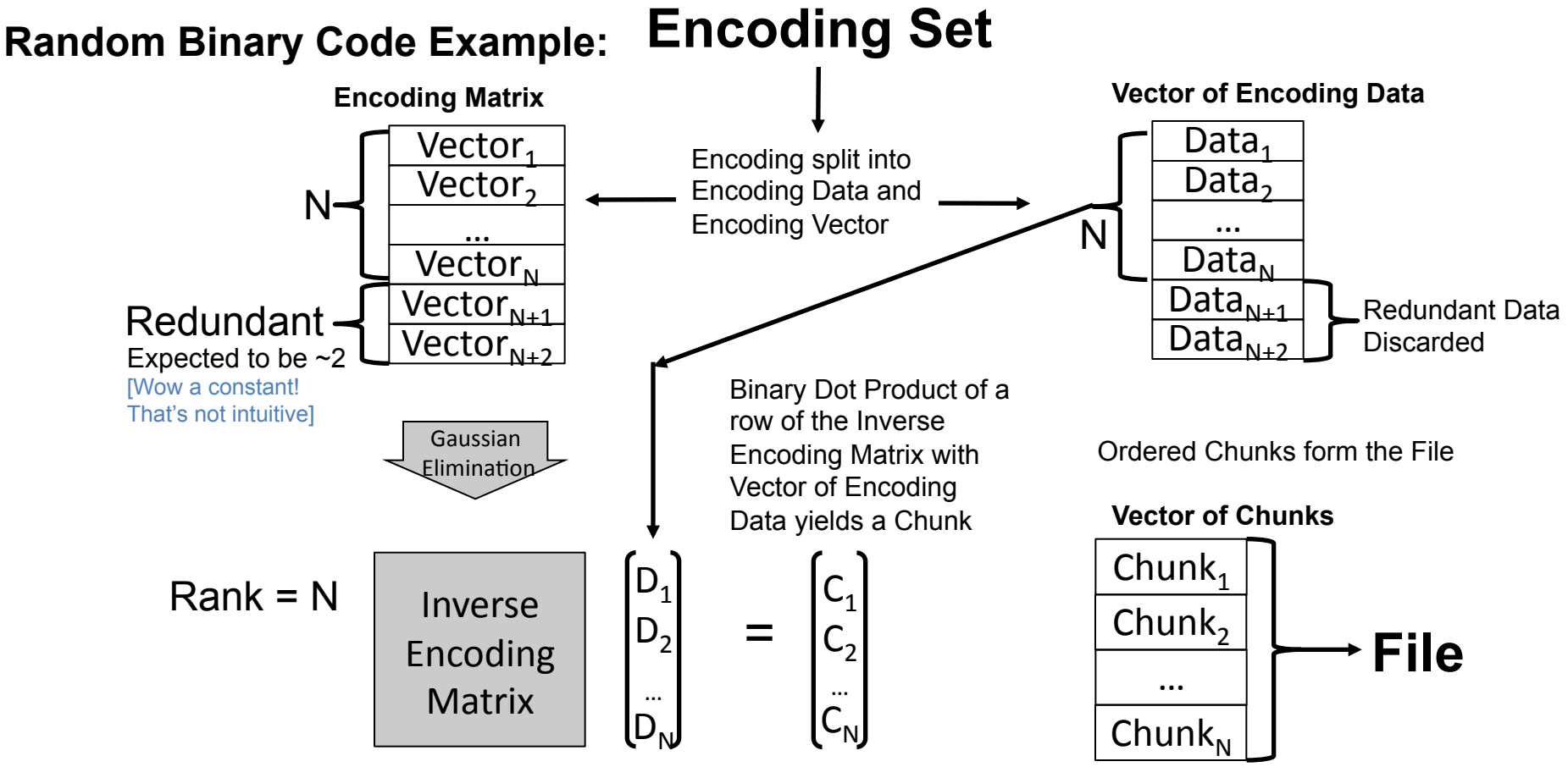
1. Multiple Encodings can be combined into a new Encoding.
2. XOR the Encoding Data together.
3. XOR the Encoding Vectors together.
4. The resulting Data and Vector forms a new Encoding.



Fun Facts:

- If both Hamming weights < N/2, then resulting weight is expected to be greater
- If both Hamming weights > N/2 then resulting weight is expected to be less
- If both Hamming weights = N/2 then resulting weight is expected to be N/2

Reconstituting an Encoding Set back into a File



Fun Facts:

If Rank = N, then Encoding Set can be used to decode all the chunks in the file.

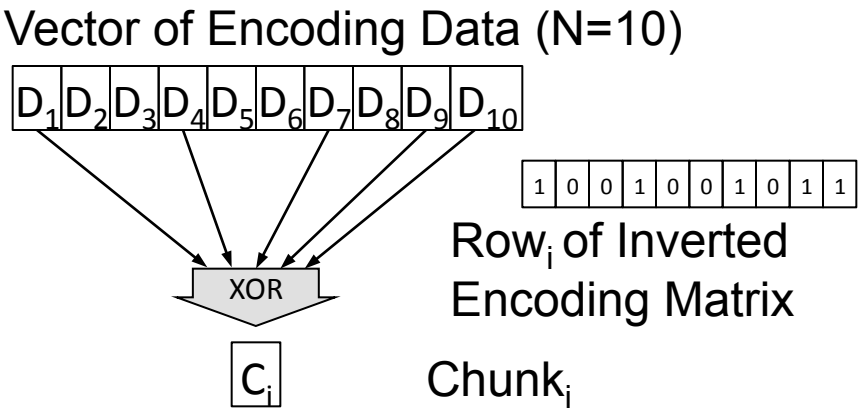
If Rank < N, then ***all*** chunks can not be decoded, but **some** may be decoded

Reconstituting a Chunk from Encodings

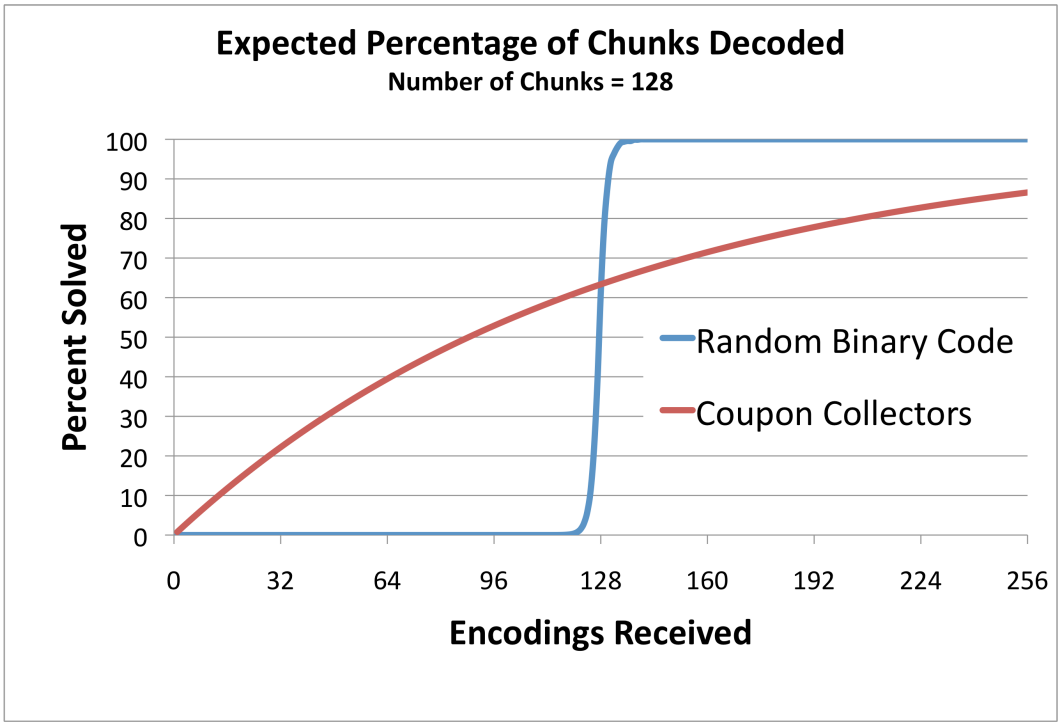
Raytheon BBN Technologies

Random Binary Code:

- Each Encoding Vector V_m is added as a Row_m into an Encoding Matrix
  - The matrix is M rows by N columns, where M>=N.
- The Encoding Matrix is inverted into the Inverted Encoding Matrix (NxN).
  - If the Encodings are not full rank, (Rank < N), the inverse will not be found, so wait for more Encodings and try again.
  - A Linear algebra package can be used to invert the Encoding Matrix
- Perform the Binary Dot Product operation on Row_i of the inverted Encoding Matrix with the Vector of Encoding Data, to yield $Chunk_i$



Random Binary Code



Bundle Fragmentation in a shuffled environment* is equivalent to the **coupon collector's problem**.

Many random fragments are need to be received to get all fragments.

For example, the probability of getting the last fragment is 1/N

Expect number 0(n log(n)) or ≈ 600

- The expected number** of random binary encodings needed to solve for all chunks is N + ε, where ε ≈ 1.6

* Randomly receive a new fragment with replacement.

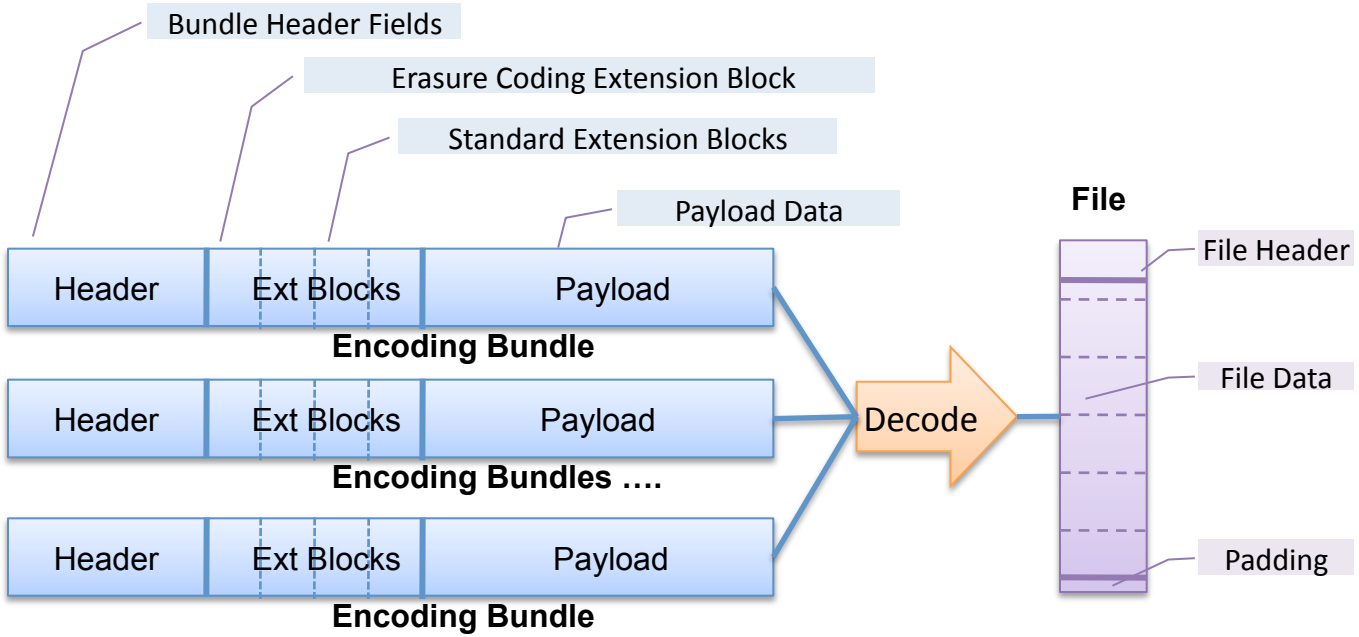
** V.F. Kolchin, *Random graphs, Cambridge University Press (1999).*

Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- Erasure Coding Architecture
- Encoding Process for Random Binary Code
- **Erasure Coding Headers**
- End-to-End File Transfer
- End-to-End Bundle Transfer
- Proof of Concept Implementation Results
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

Multiple Headers used for Full EC Protocol



Place	Accessible	Fragments	Example
Bundle Header Fields	Bundle forwarder and all Encoders	Every fragment	Transfer Spec
Erasure Extension Block	Encoder, Decoder, Intermediate, Re-coder	Once per EC bundle	Encoding Vector
Standard Ext Block	Bundle forwarder and all Encoders	Once per EC bundle	Sender Signature
Payload Data	Encoder, Decoder, Re-coder	Once per EC bundle	Encoding Data
File Header	Encoder, Decoder	Once per File	File Name, Length
File Data	Encoder, Decoder	Once per File	File Content
Padding	Ignored by Decoder,	Once Per File	Ignored

Erasure Coding (EC) Extension Block

Raytheon BBN Technologies

Block Type: 0xEC

Not a metadata extension block.

Proc. Flags: 0x00

Block Length: SDNV

Length of extension block data.

Version: SDNV

EC Extension Block version which increments with newer versions

Next Encapsulated Protocol: SDNV

Specifies protocol (i.e., data type) of original data (e.g., bundle, app datagram, app stream).

Defines format for encoding bundle payload and decoded chunks.

Note: similar to IP's "Protocol" field and used by Destination Decoder.

Encoding Set ID: UUID (128bits) [RFC 4122]

Universally Unique ID for encodings that represent the same data chunks

Next Encapsulated Protocol can use a hash function to map their GUID to ECID

Handling Spec: SDNV

Hints on how intermediate routers should order, prioritize, or drop this encoding relative to other encoding bundles with the same Encoding Set ID. (Undefined for Version 1)

Number of Chunks: SDNV

Number of chunks in Encoding Set, if 0 then no maximum number of chunks (e.g. stream)

Encoding Vector Format : SDNV

Number of format type used to to interpret Encoding Vector

Encoding Vector: Bytes

Determined by EV Format

Encoding Vector Formats

Raytheon
BBN Technologies

Full Vector (1): Good for dense vectors
- Vector (bytes) length is Ceiling(Number of Chunks / 8)
- Binary Vector over the GF(2) Field

List of Chunk Indexes (2): good for sparse vector with large number of chunks
- List length <SDNV>
- [chunk index<SDNV>, chunk index<SDNV>, ….]
- Binary Vector over the GF(2) Field

Start/Extent (3): Good when coefficients are clustered, e.g. blocks or windows
- Start chunk number <SDNV>
- Coefficients byte array <length <SDNV> bytes>
- Binary Vector over the GF(2) Field

Other Fields or Rings (TBD): Different Tradeoffs than Random Binary Code
- Encoding and Decoding time and storage
- Expected number of encodings needed to solve.

Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- Erasure Coding Architecture
- Encoding Process for Random Binary Code
- Erasure Coding Headers
- **End-to-End File Transfer**
- End-to-End Bundle Transfer
- Proof of Concept Implementation Results
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

File Format:

Raytheon
BBN Technologies

Type: 4 Bytes: 0xECECECEC

Indicates File Header type

Version: 4 Bytes: 0x01

version number of header which increments with newer versions

Format: 4 Bytes: 0x01

Length of extension block data.

File UUID: 128 bits

Must match Encoding Set UUID in Erasure Coding Extension Block

File Length: 128 bits

Length of file in bytes

File Name: String

String Length: 4 Bytes

String Bytes: byte array

Sting Terminator: byte 0x00

Path Name: String

String Length: 4 Bytes

String Bytes: byte array

Sting Terminator: byte 0x00

File Data: byte array

Data bytes for the file data, length is File Length

Padding:

Extra Bytes needed to pad last Chunk to be full chunk length

File Transfer Spec Mapping to Bundle Headers

Raytheon BBN Technologies

Name	Type	Header	Description
Destination	EID	Bundle	Place to send file
Class of Service	Int	Bundle	Prioity
Expire Time	Int	Bundle	For the File (LifeTime + Creation Time)
Hop Limit	Int	Bundle	Possible RFC 5050 extension.
Send Stop	flag	Bundle? File?	Request Acknowledgement Field File header
Report Progress	flags	Bundle	Report progress flags for each bundle
Number of Chunks	Int	Extension	Keep small to reduce decode time
Chunk Length	Int	Extension	Keep small to avoid fragmentation
Percentage Redundancy	Int	Extension	Keep redundancy small for efficiency
Contact Limit	Int	Extension	Rate or amount or percent?
Link Limit	Int	Extension	Rate or amount or percent?
BPA Limit	Int	Extension	Rate or amount or percent?
Path Limit		Extension	Endemic routing?
File Meta Data	Properties	Payload	File directory info: name, permission, date,
File Content	File	Payload	File data

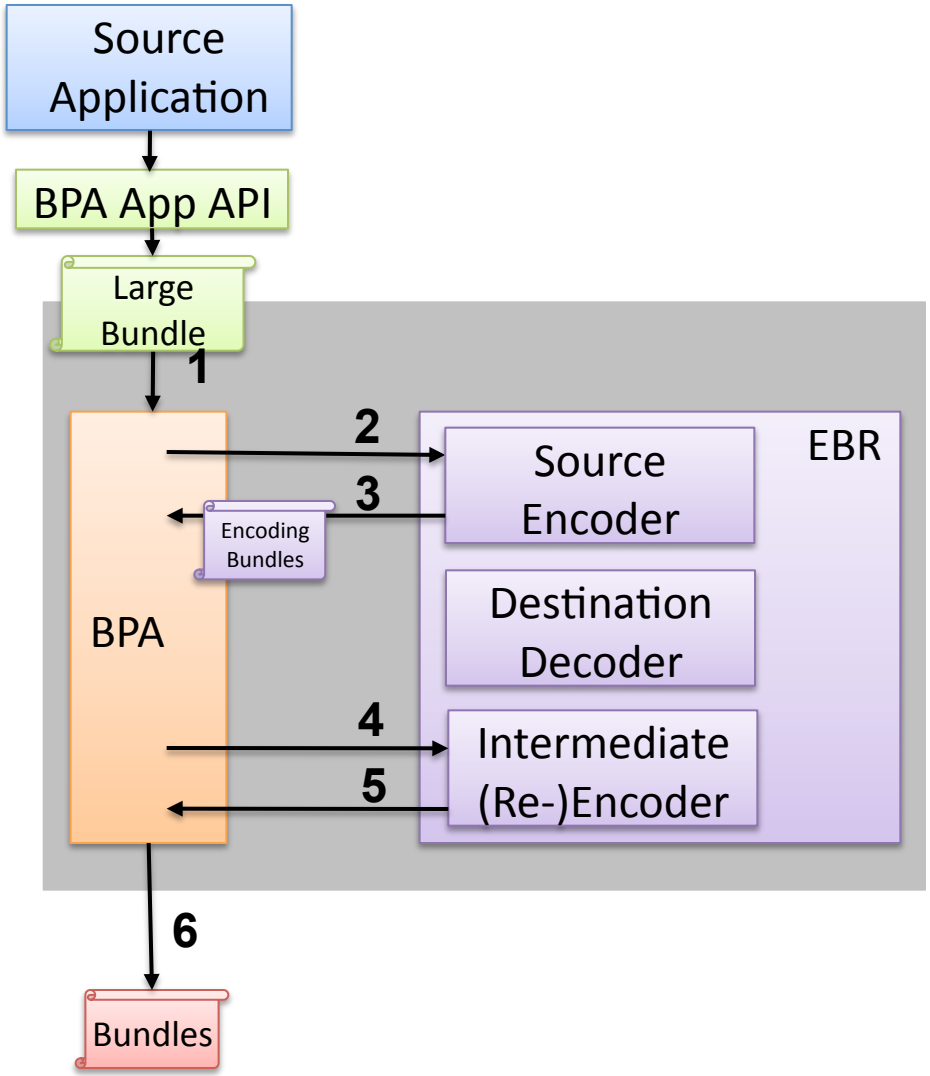
Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- Erasure Coding Architecture
- Encoding Process for Random Binary Code
- Erasure Coding Headers
- End-to-End File Transfer
- **End-to-End Bundle Transfer**
- Proof of Concept Implementation Results
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

Processing Steps: Source Node

Raytheon
BBN Technologies



1. Incoming app large-bundle addressed as:
   From: dtn://SourceHost/SourceApp
   To: dtn://DestHost/DestApp

2. Source Encoder processes bundles addressed to EIDs with “dtn” scheme.

3. Source Encoder deletes original large-bundle and injects Encoding Bundles addressed as:

   From: ebr://SourceHost/ebr
   To: ebr://DestHost/ebr

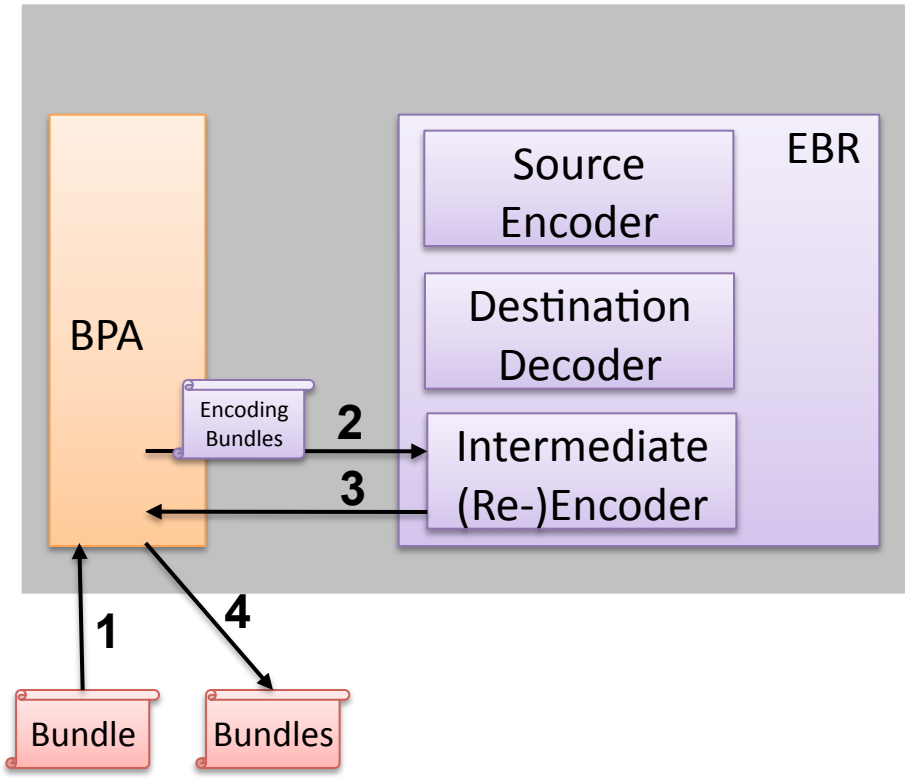
4. Intermediate Encoder handles bundles addressed to EIDs with “ebr” scheme.

5. At source node, Intermediate Encoder simply forwards Encoding Bundles as if it were an Intermediate Node (see next slide).

6. BPA forwards Encoding Bundles as instructed in Step 5. From the BPA’s point of view, they are simply ordinary bundles.

Processing Steps: Intermediate Node

Raytheon
BBN Technologies



1. Incoming bundle addressed as:
   From: ebr://SourceHost/ebr
   To: ebr://DestHost/ebr

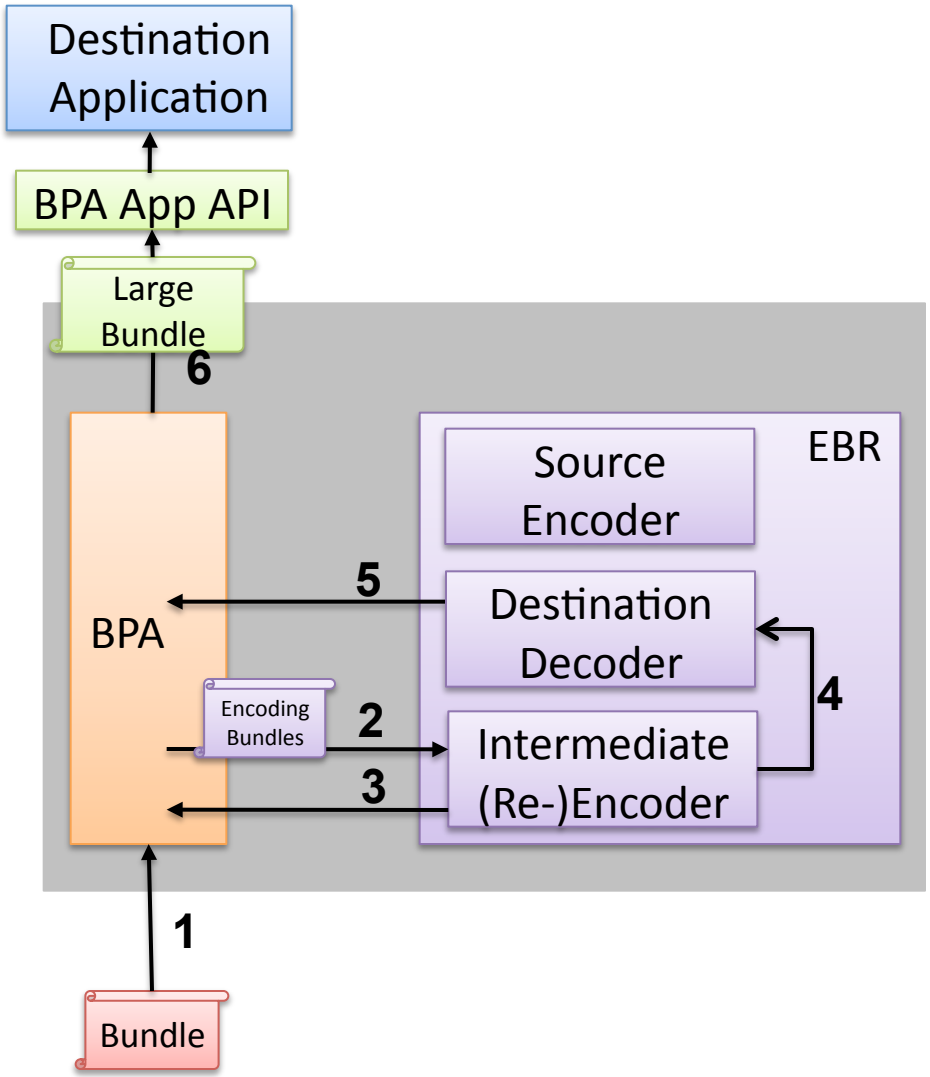
2. Intermediate Encoder processes Encoding Bundles addressed to EIDs with "ebr" scheme.

3. Intermediate Encoder prepares Encodings to forwarding to neighbors, one or more of the following:
   a) Forward the Encoding Bundle
   b) Store it for re-encoding and/or filtering redundant subsequent encodings
   c) Delete it if determined to be redundant
   d) Generate & send new Encoding Bundles
   e) Change Order of outgoing bundles
   f) Limit bundles per neighbor
   g) Collect statistics

4. BPA forwards Encoding Bundles as instructed in Step 3. From the BPA's point of view, they are simply ordinary bundles.

Processing Steps: Destination Node

Raytheon
BBN Technologies



1. Incoming bundle addressed as:
   From: ebr://SourceHost/ebr
   To: ebr://DestHost/ebr

2. Intermediate Encoder processes Encoding Bundles addressed to EIDs with “ebr” scheme.

3. Intermediate Encoder may do one or more of the following:
   a) Store the Encoding Bundle
   b) Delete it if redundant

4. If received a full rank of encodings, Intermediate Encoder signals the Destination Decoder to decode the original source application large-bundle.

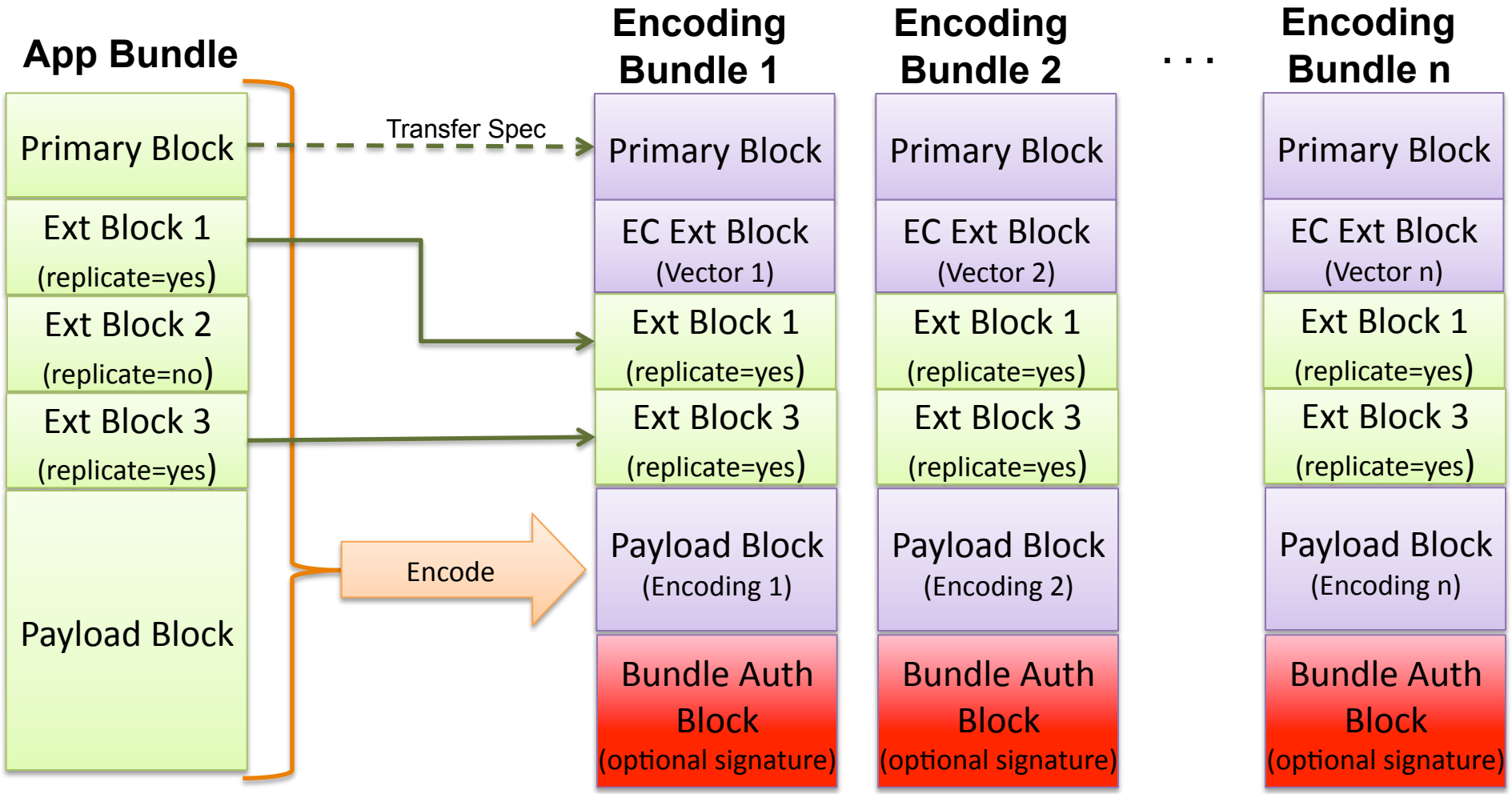
5. Destination Decoder deletes the Encoding Bundles and injects the decoded application bundle.

6. Destination Decoder could send a “Stop” and/or “Purge” control messages, (to be defined)

7. The original source application large-bundle is delivered to the destination application.

Application Bundle to Encoding Bundle

Raytheon
BBN Technologies



Note: Bundle Auth Block details are TBD

Application Bundle Transfer Spec

Raytheon
BBN Technologies

- Refers to an Application Bundle's metadata for properly transporting the bundle
  - Destination Host
  - Lifetime
  - *Some* Processing Control Flags
    - Class of Service
    - Singleton Destination
- Is conveyed in the Encoding Bundles' primary block
  - (see next slide)

Encoding Bundles' Primary Block

Raytheon BBN Technologies

- Source EID: ebr://<SourceHost>/ebr
  - <SourceHost> is the host generating the Encoding Bundle, which can be either the Source Node or an Intermediate Node
- Destination EID: ebr://<DestinationHost>/ebr
  - Source Node copies <DestinationHost> from the Destination SSP field in the Application Bundle's primary block
  - Intermediate Nodes generating new Encoding Bundles from existing Encoding Bundles simply copy the Destination EID
- Creation Timestamp: changed to the time the Encoding Bundle was created, not to the Application Bundle creation time,.
- Lifetime: changed to expire at same time as the original Application Bundle
- Processing Control Flags: CoS & Singleton Destination bits
  - Source Node copies them from the Application Bundle's primary block
  - Intermediate Nodes generating new Encoding Bundles copies them from corresponding existing Encoding Bundles
- Other fields are set by EBR as needed

Handling Application Bundle's Extension Blocks

Raytheon BBN Technologies

- Extension blocks have a flag that dictate whether or not they should be replicated in every bundle fragment
- Encoding Bundles are not technically "bundle fragments", but are semantically similar
  - If "replicate" flag is set, then the extension block should be included in every Encoding Bundle
  - All extension blocks, regardless of replicate flag, are encoded with the rest of the original Application Bundle (see next slide for rationale)
- When Destination Decoder decodes Application Bundle, one copy of each replicated extension block is included
  - Destination Decoder may choose any of the copies, including the original copy in the original Application Bundle
  - RFC 5050 provides no guidance about which fragment's copy of the extension block to use during reassembly

Rationale for Encoding Entire Original Bundle

Raytheon BBN Technologies

- To reconstitute the original Application Bundle, Encoding Bundles need to convey the following:
  - Entire primary block
  - Entire extension blocks
  - Entire payload
- Specifying a new format to convey this info would essentially result in a new bundle format
- Instead we use the existing RFC 5050 bundle format
  - i.e., we encode the entire “over-the-wire” representation of the original Application Bundle

Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- Erasure Coding Architecture
- Encoding Process for Random Binary Code
- Erasure Coding Headers
- End-to-End File Transfer
- End-to-End Bundle Transfer
- **Proof of Concept Implementation Results**
- Future Work
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

Erasure Coding Implementation:

Raytheon
BBN Technologies

- **File-transfer encoder and decoder** as standalone DTN applications using Erasure Coding Extension Blocks (both C and Java implementation).
- **Large Bundle encoder and decoder** integrated into DTN 2.7 BPA (C implementation)
  - Extension Block replication and Signature Block not supported
- **Prototype Intermediate Re-encoder** integrated into DTN v2.7 BPA (C implementation)
- All all prototypes used Random Binary Encoding over the GF(2) field.

Recommendations for Random Binary Coding

Raytheon BBN Technologies

- Use sparse hamming weight over dense hamming weights encodings.
  - This reduces the time to encode
  - But has the same time solve, the same time to check if redundant, and the roughly the same expected number of extra encodings needed to get a full rank set.
- Decoding time is dominated by XOR of encoding data
  - Use 64 bit XOR operation and not byte wise XOR operations.
  - Reduce the number operations, e.g. compare equation solvers to matrix inversion.

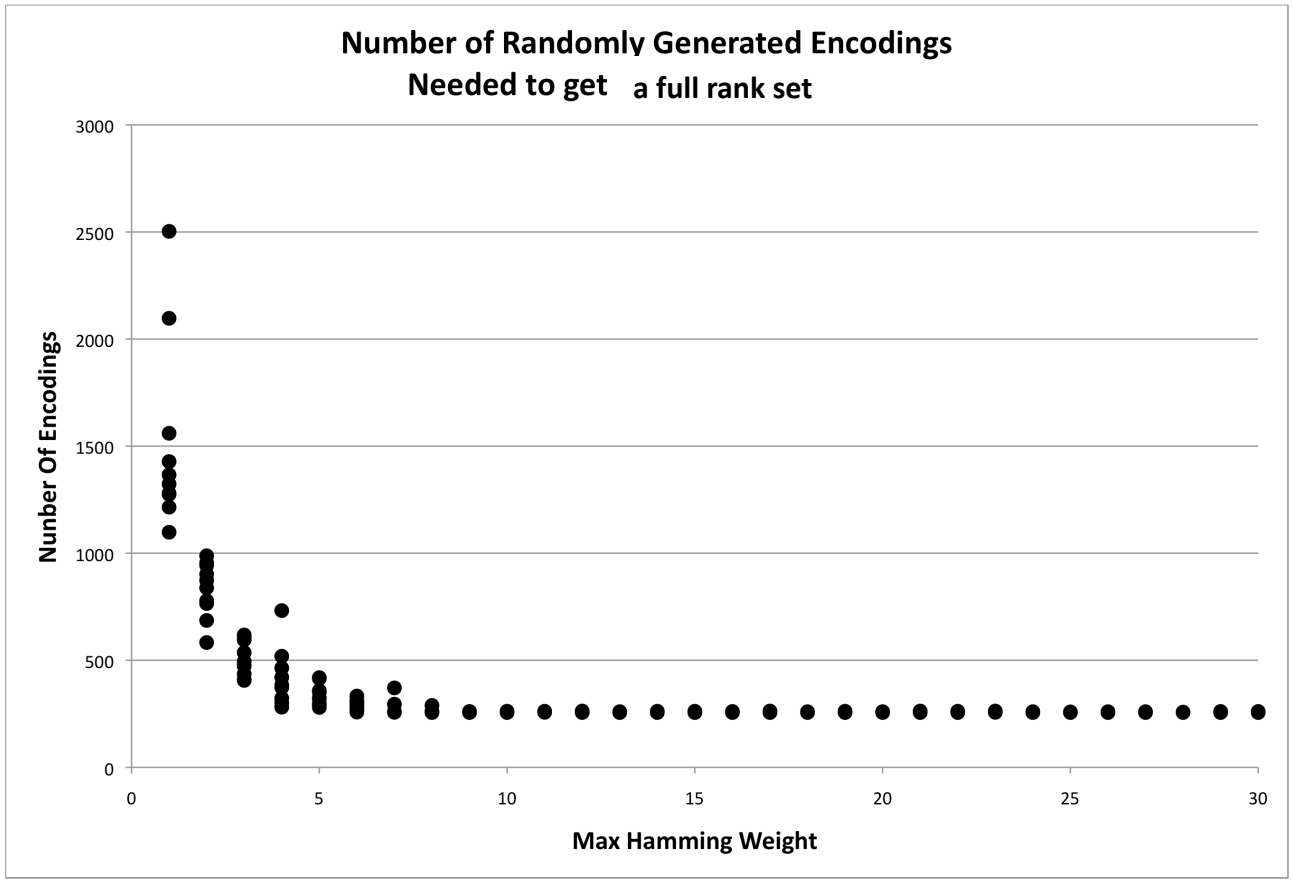
Test Setup

Raytheon BBN Technologies

- Encodings were generated at random with a max hamming weight.
  - Max Hamming weight is the independent variable for these experiments
  - chunk indexes were chosen at random with replacements, so actual Hamming weight might be less than max hamming weight.
- The encoding were inserted into an encoding set until a full rank set was collected.
  - Number of encoding generated
  - Time to generate the encoding (dominated by cost to XOR chunks)
  - Time to check if a encoding raises the rank of current set (order N^2 of the num chunks)
- The encoding set was solved and the file chunks was reconstructed.
  - Time to solve the matrix inversion (order N^3 of the num chunks)
  - Time to reconstruct chunk (dominated by cost to XOR encoding data)
- All Test held these parameters constant:
  - File size 829KB
  - Number of Chunks 256
  - Chunk Length 3240 bytes
  - 12 measurements for each hamming weight
  - The processor was a 2.5Ghz Windows laptop running Java 1.6

Using Low Hamming Weight

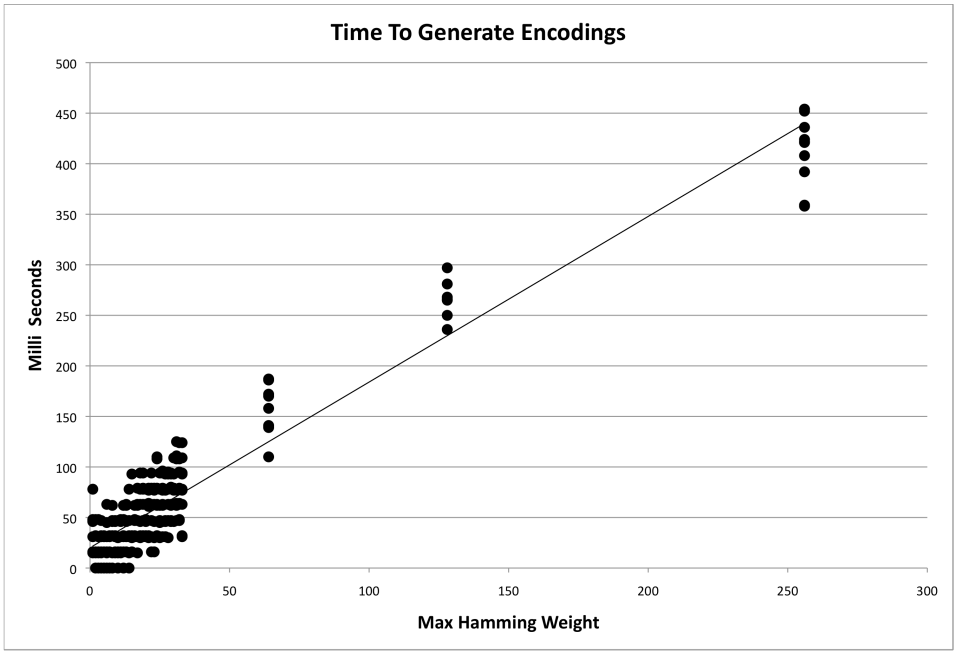
Raytheon
BBN Technologies



- Encodings are generated randomly until a full rank set was collected
- For extremely sparse hamming weight substantially more encodings are needed to get a full rank set
- But even for fairly sparse weight (10 out of 256), the number converges to the theoretical expected value (256 + 2) for dense vectors.

Encoding Generation time grows Linearly

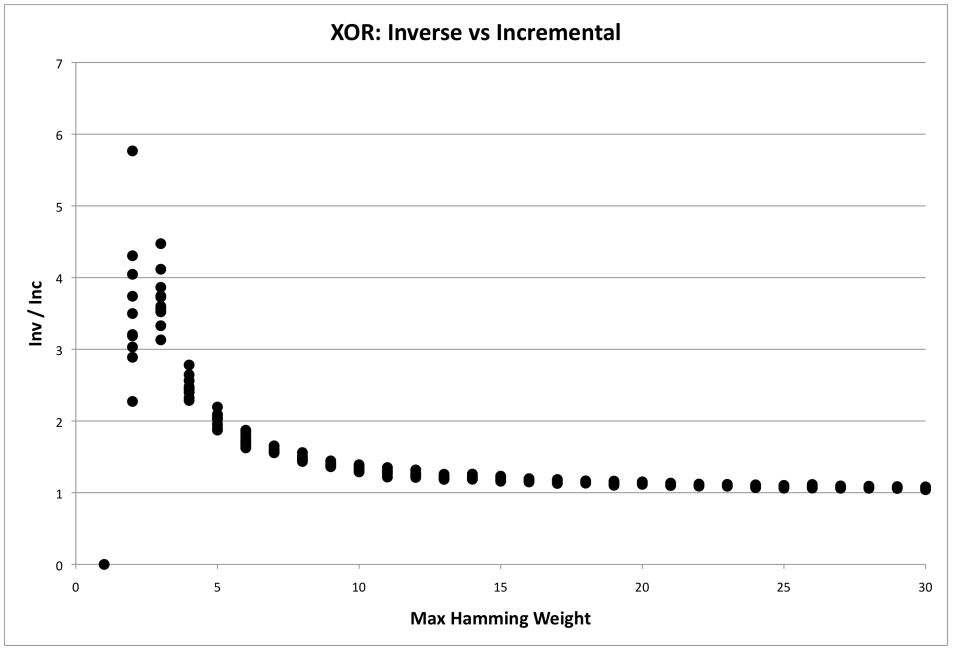
Raytheon
BBN Technologies



- To calculate the Encoding Data, a chunk is XORed for each bit in the Encoding Vector, i.e. the hamming weight.
- The time to generate an Encoding grows linearly with its hamming weight.

Matrix Inverse vs Incremental Solver

Raytheon
BBN Technologies



- An alternative mechanism for deriving the chunks, is to use an incremental solver.
- For extremely sparse hamming weights, incremental was up to 5 times faster than performing the inverse (in terms of the number of encoding data XORS).
- But even for fairly sparse weights (15 out of 256), the solve time converges to be the same.

Summary

Raytheon
BBN Technologies

- Encode Side:
  - Using sparse hamming weights reduces the cost to generate an encoding. (linear relationship)
  - Unfortunately to get a full rank set of the number of random encodings with extremely sparse weights (< 5 of 256) needs many encodings (low error recovery)
- Decode side:
  - For extremely sparse hamming weights (<5 out of 256), the the use of an incremental solver shows dramatic improvements
  - but for sparse to heavy hamming weights there is no advantage is no advantage over a matrix inversion.

Outline

Raytheon
BBN Technologies

- Erasure Coding Context
- Erasure Coding Architecture
- Encoding Process for Random Binary Code
- Erasure Coding Headers
- End-to-End File Transfer
- End-to-End Bundle Transfer
- Proof of Concept Implementation Results
- **Future Work**
  - Additional End-to-End Transfer Types
  - Intermediate Encoders

Inter-BPA Algorithms

Raytheon
BBN Technologies

- End-to-End flow control and congestion control
  - Stop and Purge control bundles.
- Intermediate Re-encoder
  - Epidemic and Endemic Flooding Protocols for Encodings
  - Contact-time re-encoding based on neighbor.
- Intermediate Encoder
  - Handling Specification for encoding order, buffering, or custody.
- Push vs Pull Content
  - Push: Disseminate to file to multiple destinations
  - Pull: Receive Encoding from multiple caches
    - E.g. Distributed Hash table, Bit torrent, or Software Updates

Other Use Cases for EC Architecture

Raytheon BBN Technologies

- *File (All chunks are known a priori, all or no chunks needed within expire time)
  - Vector has complete range in which chunk bits can be set
  - Data is XOR of just the included chunks
- Traditional Fragmentation (Fragment # is Chunk #)
  - Vector has only **one** chuck bit is set
  - Data in clear
- FEC Block Parity Packet (packet # is Chunk #)
  - For Packet (like Fragmentation)
    - Vector has only **one** chunk bit is set
    - Data in clear
  - For block parity,
    - Vector has **All** chucks in block
    - Data is XOR of **all** chunks in block
- Multicast NACK (Destination needs a specific chunk)
  - Vector has only **one** chuck bit set
  - Data in clear
- Stream (Sliding window of chunks)
  - Vector has a range in which chunk bits can be set
  - Data is XOR of just the included chunks
- Variable Importance data stream (Video)
  - (different chunks have different reliable or latency requirements)

Support for other Coding Schemes

Raytheon
BBN Technologies

- The Encoding Vector Format can accommodate other encoding schemes than Random Binary Encoding.

- Other Coding Schemes have different tradeoffs:
  - Encoding vs Decoding Resources
  - Percentage of decoded of chunks, before all chucks can be decoded
  - Expected number encodings before chunks can be decoded.
  - Exploitation of the type of data being transferred (e.g. video, voice, pictures)
  - Exploitation of multicast links

Security Issues

Raytheon
BBN Technologies

- Inserting an Encoding where the Encoding Vector does not match Encoding Data (i.e. random data) will scramble part of the file.
  - Encodings bundles should be signed
- What operations can the Intermediate Nodes be trusted?
- Does the source have to be authenticated?