

RMT
Internet-Draft
Expires: April 20, 2006

C. Neumann
V. Roca
INRIA Rhone-Alpes
R. Walsh
Nokia
October 17, 2005

A File Aggregation Scheme for FLUTE
draft-neumann-rmt-flute-file-aggregation-02.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April 20, 2006.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

This document introduces a logical and physical file aggregation scheme for File Delivery over Unidirectional Transport (FLUTE). The logical file aggregation mechanism is a generalized grouping mechanism, allowing to logically group files. The physical file aggregation scheme allows, additionally to a logical grouping, to more efficiently use Forward Error Correction (FEC) in the context of

FLUTE, in particular when dealing with a large number of "small" files. Unlike a solution based on the creation of an archive, the object aggregation scheme (1) avoids the need to perform preliminary transformations on the content and (2) preserves the possibility to extract a subset of the content, which may be critical aspect with some partially reliable broadcasting test cases.

Table of Contents

- 1. Introduction 4
 - 1.1 Motivations 4
 - 1.1.1 Logical aggregation 4
 - 1.1.2 Physical aggregation 4
 - 1.1.3 Aggregation Mode Selection 6
 - 1.2 Modifications compared to the FLUTE version 1 specifications 6
- 2. Conventions used in this document 8
- 3. The Generalized Grouping Mechanism 9
 - 3.1 Syntax of FDT Instance with the Generalized Grouping Mechanism 9
 - 3.2 A simple example 9
- 4. The Physical File Aggregation Scheme 11
 - 4.1 multipart/mixed MIME type object and multipart/related MIME type object 11
 - 4.2 Extending the FDT with Aggregated Object Information . . . 11
 - 4.3 Syntax of FDT Instance with Aggregated Object Description Information 12
 - 4.4 Symbol alignment of aggregated files 17
 - 4.4.1 MIME compatible padding 17
 - 4.5 Recovering files at FLUTE receiver supporting physical file aggregation 17
 - 4.5.1 Recovering files before the entire reception of the aggregated object 17
 - 4.5.2 Recovering files after the entire reception of the aggregated object 18
 - 4.6 Recovering files at FLUTE receiver that does not support physical file aggregation 18
 - 4.7 Limitations 19
- 5. FLUTE version 1 backward compatibility 20
 - 5.1 Redirection mechanism 20
- 6. Security Considerations 21
- 7. IANA Considerations 22
- 8. Acknowledgments 23
- 9. References 24
 - 9.1 Normative References 24
 - 9.2 Informative References 25
- Authors' Addresses 25
- A. Example of FDT Instance (informative) 26

B. Example of FDT Instance (informative) 27
Intellectual Property and Copyright Statements 28

1. Introduction

1.1 Motivations

This document introduces a logical and physical file aggregation scheme for File Delivery over Unidirectional Transport (FLUTE) [10], version 1. FLUTE is a protocol for unidirectional delivery of files and builds on Asynchronous Layered Coding (ALC), version 1 [7], the base protocol designed for massively scalable multicast distribution. The logical object aggregation mechanism allows to logically group correlated files. The physical object aggregation scheme additionally allows to more efficiently use Forward Error Correction (FEC) with FLUTE in some situations.

1.1.1 Logical aggregation

The logical aggregation mechanism offers a means to logically group files without physically binding them to the same transport object. This is achieved by labeling files as being part of one or more groups. This functionality is desirable when transmitting a set of closely related files that will be used by the receiver in the conjunction with each other. The effect is to simplify the FLUTE-to-application messaging and processing overhead and to enable selective caching of files when it is not feasible to either promiscuously receive all files or explicitly indicate all wanted files in advance of joining the FLUTE session.

One example is an html page (file) with several embedded images. By labeling the web page file and all related image files as being part of the same group, the FLUTE receiver knows in advance the files he needs to download based on only the URI of the web page file. Without this grouping mechanism it would have to analyze the web page file and then deduce which other files are related and need to be downloaded.

1.1.2 Physical aggregation

The main idea of the physical file aggregation scheme is to aggregate a (possibly large) set of (possibly small) files into one large aggregated object, that is treated as a single transport object by ALC. The benefits of logical aggregation, described above, also apply to physical aggregation. However, a shared-fate model is introduced as the successful reception of one of the aggregated files is to some extent statistically correlated to the successful reception of one or more others. Thus, there is a strong incentive to only physically aggregate files that are logically related into the same aggregated transport object.

The physical file aggregation scheme is made possible by simple extension to FLUTE FDTs which provides a dedicated signaling mechanism, enabling extended FLUTE receivers to extract the files within the large aggregated object.

With physical aggregation, FEC encoding is performed on a large object rather than on each individual file, which can be highly beneficial for transmission performance. Therefore this technique offers two specific transmission performance improvements:

1. the coupon collector problem [14], that is caused by the separate FEC encoding of each individual file when file aggregation is not used, is now significantly reduced or even totally eliminated. Now FEC encoding is done over a single object, whose size is perhaps inferior to the maximum block size permitted by the FEC instance used (this is especially true with a Large Block FEC code).
2. large block FEC codes perform better on large blocks than on small blocks, and using file aggregation offers more opportunities to use such codes whose performance is significantly higher than Reed Solomon codes [13].

The performance gain of these two aspects depends on several parameters such as the FEC instance used, the aggregated object size and the number of files. Detailed quantitative analysis and explanation of the impact of all these parameters is outside the scope of this document. The physical file aggregation is mainly applicable for small files.

The specified physical object aggregation solution is significantly different from a solution that would create a single archive from the list of files (e.g. a gzip compressed tarball archive). With an archive the result is either the full reconstruction of all individual files (if enough packets have been received for decoding to complete at a receiver) or an hazardous result (since the archive will be corrupted, possibly damaging the archive headers which then prevents file extraction). Indeed, although FEC can counter the effects of packet erasures, if the number of packets received is too low for the FEC decoding process to finish, the received parity packets may turn out to be inconsequential. On the opposite, the specified physical object aggregation solution can offer a partial reliability service, i.e. it enables a receiver to reconstruct parts of the content even if the FEC decoding process has not finished. To that purpose, the physical file aggregation scheme can optionally preserve the possibility to decode and exploit a subset of the content, by informing the receivers of the size and position of individual files within the aggregated object.

Another motivation for having an object aggregation scheme compared to a basic archive based solution (e.g. tarball), is that no extra transformation (i.e. archive creation or extraction) is required at either the FLUTE sender or receiver. Everything is managed automatically by the transport mechanism according to transport-specific optimizations and can be transparent to upper applications (i.e. built on top of FLUTE), or enhanced by application hints on file relationships, without breaking the basic semantics of FLUTE sessions.

1.1.3 Aggregation Mode Selection

The selection of whether a set of files would benefit from either no aggregation, logical aggregation or physical aggregation must be made for (or by) the FLUTE sender. The merits of aggregation (Section 1.1.1 and Section 1.1.2), as well as the introduced receiver and sender complexities may be taken into account. In particular, the choice between logical and physical aggregation would be mostly application-specific and dependent on the file size distribution and the inter-file relationship (in receiver use). It would also be tuned to the anticipated end-to-end losses and any selected FEC instance.

1.2 Modifications compared to the FLUTE version 1 specifications

This document describes a simple and light extension of the FLUTE in-band signaling, the File Delivery Table (FDT), which is used to identify the group to which each file belongs to and to inform each receiver about the properties and structure of the aggregated object.

More precisely:

1. The FDT Instance syntax is extended introducing two new elements, "Group" and "aggregatedFile".
2. Additional information concerning the description of the aggregated object is added to the FDT.
3. An extra redirection to "extended" FDT Instances is introduced to be backward compatible with FLUTE version 1 specifications. This enables the use of FLUTE version 1 FDT Instance specifications in combination with the element-extended enhanced FDT schema of this specification.

These modifications are described in details in Section 3 and Section 4 .

All other features, requirements and specifications of the FLUTE

version 1 specification remain valid.

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 [3].

The terms "object" and "transport object" are consistent with the definitions in ALC [7] and LCT [8]. The terms "file" and "source object" are pseudonyms, but they are NOT pseudonyms for "object" like in FLUTE [10], since a file may be transmitted within an aggregated object, that is the only object that ALC needs to understand.

3. The Generalized Grouping Mechanism

The generalized grouping mechanism allows each file of a FLUTE session to be labeled as being part of none, one or several logical groups.

Logical aggregation is performed by using the generalized grouping mechanism.

Since there is a strong incentive to only physically aggregate files that are logically related (Section 1.1.2), physical aggregation may use this generalized grouping mechanism too, in addition to the scheme introduced in Section 4.

3.1 Syntax of FDT Instance with the Generalized Grouping Mechanism

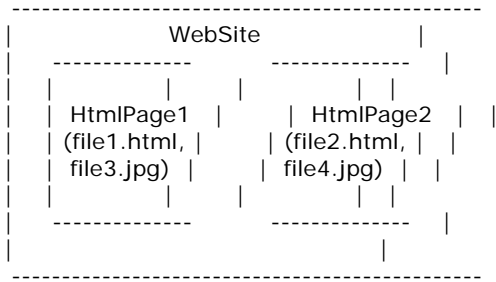
The grouping mechanism is achieved by adding the element "Group" to the FDT.

The element "Group" can be added to a "File" element, an "aggregatedFile" element (introduced in Section 4.3) or to the "FDT-Instance" element. In the first two cases it specifies that the file (or aggregated file) is part of a group that is identified by the value of the element entry "Group". A "Group" entry at "FDT-Instance" level specifies that all files (and aggregated files) listed in the FDT Instance are part of that group.

The extended FDT Instance XML schema is specified in Section 4.3.

3.2 A simple example

With this simple extension any type of relationship between files can be expressed. As an example we want to express the hierarchical relationship depicted in Figure 1. A web site is composed of two html pages (file1.html and file2.html). file1.html contains the image file3.jpg, and file2.html contains the image file4.jpg.



Example of a hierarchical relationship for a web site.

Figure 1

The hierarchical relationship can be expressed as follow: All files are part of the group "WebSite"; file1.html and file3.jpg are part of the group "HtmlPage1"; file2.html and file4.jpg are part of the group "HtmlPage2".

Other file relationships can easily be expressed with the generalized grouping mechanism.

<Editorial note>

The relations between groups are only implicitly expressed (e.g. it is not explicitly stated in the above example that "HtmlPage1" is a sub-group of "WebSite"). Is there a need to specify the relationship between groups explicitly in some way in the FDT?

</Editorial note>

4. The Physical File Aggregation Scheme

4.1 multipart/mixed MIME type object and multipart/related MIME type object

An aggregated object is either a multipart/mixed MIME type object as defined in MIME Part two [2] or a multipart/related MIME type object as defined in [4]. The aggregated object includes several files, each one delimited by the boundary delimiter defined in the MIME header. One body part (in MIME terminology) corresponds to one aggregated file.

Multipart/related MIME type objects should be used in cases where a logical dependence of the files being aggregated needs to be expressed (Multipart/related was initially developed to send entire web-page, i.e. including all images and related files part of that web-page). In all other cases multipart/mixed MIME type object should be used.

There are no restrictions nor recommendations regarding the MIME header fields of each body part compared to what is specified in MIME Part two [2]. Empty header fields are sufficient (i.e. the body parts are only delimited by the boundary delimiter without any header field), but the header field may be filled with any additionally required information.

In some cases adding additional information for each body part may be useful. Especially if we want to enable a receiver that is not aware of the aggregated object FLUTE extension to process the aggregated object and reconstruct aggregated files, it is RECOMMENDED to include the "Content-Location" attribute in each body part MIME header field.

4.2 Extending the FDT with Aggregated Object Information

The Aggregated Object Information (AOI) describes the aggregated object and its structure. In this section we describe a logical view of all information needed to process an aggregated object, and in Section 4.3 we describe its implementation within the FDT Instances as a "File" element for the aggregated object and a set of "aggregatedFile" elements.

The AOI must enable a receiver to:

- o identify that an ALC object is an aggregated object,
- o identify and have a description of the files being transmitted in an aggregated object,

- o know the position (i.e. offset) and length of each file within the aggregated object.

Therefore the AOI MUST contain the following attributes:

- o The Aggregated Object's TOI value
- o The Aggregated Object's content type value, that MUST either be set to "multipart/mixed" or to "multipart/related"
- o The URI of each file being aggregated
- o The offset of each file within the aggregated object (not considering the boundary delimiter and the MIME header)
- o The transfer length of each file within the aggregated object, or the content length if the file is not content encoded
- o The number of files aggregated in one aggregated object.

The attributes of the AOI MUST be included in the FLUTE FDT. The FDT Instances containing AOI are referred as "extended" FDT Instances in this document. Since the AOI is just an extension added to the FLUTE version 1 FDT, it is delivered within the FDT Instances, as specified in FLUTE [10].

The file aggregation scheme does not mandate any mechanism to carry the AOI, but it is RECOMMENDED that the AOI does not straddle several FDT Instances. A receiver can check if he knows the entire list of files of one aggregated object by checking if the number of described files is equal to the number of files specified in the AOI.

<Editorial note>

It has to be discussed if carrying the AOI within one FDT Instance can be mandated as mandatory. The attribute number of files is no longer required in that case.

</Editorial note>

4.3 Syntax of FDT Instance with Aggregated Object Description Information

The syntax of FDT Instances remains the same as the FLUTE version 1 specification, with the addition of the following enhancements:

1. An aggregated object has a "File" element entry in the FDT, like normal files.

- * All rules of the FLUTE version 1 specification for "File" elements apply to this entry.
 - * The entry MUST have the attribute "Content-Type" and its value must either be set to "multipart/mixed" or "multipart/related" according to the MIME object type used for the aggregated object.
 - * The "Content-Location" attribute SHOULD contain a relative URI reference [5], since aggregated objects may have no absolute URI (they are not regular files). An example of such a relative URI for the aggregated object is "/AO1".
 - * To differentiate an FDT file entry of a normal file of type "multipart/mixed" or "multipart/related" from an FDT file entry of an aggregated object, the receiver has to check if there exists elements of type "aggregatedFile" whose "AOTOI" attribute value is equal to the TOI of the aggregated object.
 - * The entry MUST have the attribute "Number-of-Files" and its value is the number of aggregated files within the aggregated object.
2. The element "aggregatedFile" describing a file being aggregated is added to the XML Schema. For this element the attributes must be set according to the following rules:
- * The attribute "AOTOI", that identifies the TOI of the corresponding aggregated object, MUST be set.
 - * The attribute "Content-Location" MUST be set and assigned a valid URI as defined in [10].
 - * The attributes "Content-Length" (or "Transfer-Length" if the file is content encoded FLUTE [10]) and "Content-Offset" MUST be specified. "Content-Offset" specifies the offset of the file within the aggregated object. More precisely, this is the number of bytes (8 bit words) from the start of the aggregated object up to the first byte of the file, not considering the boundary delimiter and the MIME header. (Note, that the use of multipart MIME ensures that the files are byte aligned).
 - * The attributes "Content-Encoding" and "Content-MD5" MAY be used. In that case these attributes MUST be used for the purpose as described in [10].

The following specifies the XML Schema [11][12] for FDT Instance:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fl="http://www.example.com/flute"
  elementFormDefault:xs="qualified"
  targetNamespace:xs="http://www.example.com/flute">
<xs:element name="FDT-Instance">
<xs:complexType>
<xs:sequence>
<xs:element name="File" maxOccurs="unbounded">
<xs:complexType>

<xs:sequence>
<xs:element name="Group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
  <xs:anyAttribute processContents="skip"/>
</xs:complexType>
</xs:element>
<xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>

<xs:attribute name="Content-Location"
  type="xs:anyURI"
  use="required"/>
<xs:attribute name="TOI"
  type="xs:positiveInteger"
  use="required"/>
<xs:attribute name="Content-Length"
  type="xs:unsignedLong"
  use="optional"/>
<xs:attribute name="Transfer-Length"
  type="xs:unsignedLong"
  use="optional"/>
<xs:attribute name="Content-Offset"
  type="xs:unsignedLong"
  use="optional"/>
<xs:attribute name="Number-of-Files"
  type="xs:unsignedLong"
  use="optional"/>
<xs:attribute name="Content-Type"
  type="xs:string"
  use="optional"/>
<xs:attribute name="Content-Encoding"
  type="xs:string"
  use="optional"/>
<xs:attribute name="Content-MD5"
  type="xs:base64Binary"
  use="optional"/>
<xs:attribute name="FEC-OTI-FEC-Encoding-ID"

```

```

        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="FEC-OTI-FEC-Instance-ID"
        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="FEC-OTI-Maximum-Source-Block-Length"
        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="FEC-OTI-Encoding-Symbol-Length"
        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols"
        type="xs:unsignedLong"
        use="optional"/>
<xs:anyAttribute processContents="skip"/>
</xs:complexType>
</xs:element>

<xs:element name="aggregatedFile" maxOccurs="unbounded">
<xs:complexType>

<xs:sequence>
<xs:element name="Group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:anyAttribute processContents="skip"/>
</xs:complexType>
</xs:element>
<xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>

<xs:attribute name="Content-Location"
        type="xs:anyURI"
        use="required"/>
<xs:attribute name="AOTOI"
        type="xs:positiveInteger"
        use="required"/>
<xs:attribute name="Content-Offset"
        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="Content-Length"
        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="Transfer-Length"
        type="xs:unsignedLong"
        use="optional"/>
<xs:attribute name="Content-Offset"
        type="xs:unsignedLong"

```

```

        use="optional"/>
<xs:attribute name="Content-Type"
    type="xs:string"
    use="optional"/>
<xs:attribute name="Content-Encoding"
    type="xs:string"
    use="optional"/>
<xs:attribute name="Content-MD5"
    type="xs:base64Binary"
    use="optional"/>
<xs:anyAttribute processContents="skip"/>
</xs:complexType>
</xs:element>
<xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>

<xs:sequence>
<xs:element name="Group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:anyAttribute processContents="skip"/>
</xs:complexType>
</xs:element>
<xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>

<xs:attribute name="Expires"
    type="xs:string"
    use="required"/>
<xs:attribute name="Complete"
    type="xs:boolean"
    use="optional"/>
<xs:attribute name="Content-Type"
    type="xs:string"
    use="optional"/>
<xs:attribute name="Content-Encoding"
    type="xs:string"
    use="optional"/>
<xs:attribute name="FEC-OTI-FEC-Encoding-ID"
    type="xs:unsignedLong"
    use="optional"/>
<xs:attribute name="FEC-OTI-FEC-Instance-ID"
    type="xs:unsignedLong"
    use="optional"/>
<xs:attribute name="FEC-OTI-Maximum-Source-Block-Length"
    type="xs:unsignedLong"
    use="optional"/>
<xs:attribute name="FEC-OTI-Encoding-Symbol-Length"
    type="xs:unsignedLong"

```

```
        use="optional"/>
<xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols"
  type="xs:unsignedLong"
  use="optional"/>
<xs:anyAttribute processContents="skip"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

4.4 Symbol alignment of aggregated files

In uses cases where the the aggregated files within the aggregated object needs to be symbol aligned, the mechanism described in this section SHOULD be used. The mechanism stays compatible with the MIME object format and allows symbol alignment in the same time.

4.4.1 MIME compatible padding

We define a new MIME header field, that allows to add padding within the MIME part header.

The field is defined as follows (it refers to several syntax rules that are defined by [1]):

```
extension-field = "Padding"      ":" *LWSP-char
```

where *LWSP-char is filled with an appropriate number of whitespaces to achieve symbol alignment of the following file.

Receivers that are not aware of this header field can process the MIME object anyhow, by just skipping process this field (this is the default behavior for unknown header fields).

4.5 Recovering files at FLUTE receiver supporting physical file aggregation

4.5.1 Recovering files before the entire reception of the aggregated object

Aggregated objects SHOULD NOT be content encoded in order to enable file recovery from an aggregated object before the whole aggregated object is received/reconstructed. Content encoding largely restricts the ability to access the individual files within the aggregated object before content decoding was successful. Therefore we do not recommend content encoding of aggregated objects when partial

reliability is required. Conversely individual aggregated files may be content encoded.

The information provided with the FDT Instances allows a receiver to identify all source blocks and source symbols for each aggregated file. File recovery is done, for each aggregated file, by:

1. Identifying the corresponding aggregated object using the "AOTOI" attribute, and thus the ALC object carrying the aggregated object.
2. Identifying the source symbols (and their corresponding source blocks) carrying the file data, using the relevant blocking algorithm and any related FEC-OTI parameters. The "Content-Offset" attribute is used to calculate the first transport object symbol and its corresponding source block of the file. The "Content-Length" (or "Transfer-Length") attribute is used to identify the remaining symbols (and their corresponding source blocks) of the file. Note that the start-of-file and end-of-file boundaries do not necessarily correspond to symbol boundaries, if the symbol alignment mechanism of Section 4.4 is not used. If the mechanism is used, we are ensured that the start-of-file corresponds to a symbol boundary.
3. Waiting until all symbols have been received or decoded to reconstruct the aggregated file.

4.5.2 Recovering files after the entire reception of the aggregated object

After the entire reception of the aggregated objects the receiver is assured that he received all symbols and source blocks to reconstruct all aggregated files. As in the previous section the information provided with the FDT Instances allows a receiver to identify for each aggregated file the symbols carrying the file data, and therefore reconstruct the individual files.

4.6 Recovering files at FLUTE receiver that does not support physical file aggregation

A receiver that does not support physical file aggregation can recover aggregated files after full reception of the aggregated object, if enough information is carried within the aggregated object (see Section 4.1). In that case a MIME reader can interpret the file, and extract the aggregated file out of it.

4.7 Limitations

The following limitations have to be considered when using object aggregation:

1. No individual file can be modified within an aggregated object. If an update is required at least two possibilities exist. Other mechanisms may be used, but are out of scope of this specification: (1) a brand new aggregated object is created and replaces the previous aggregated object whose transmission MUST stop; or (2) the new file is sent individually by FLUTE along with an FDT entry that shows that it out dates the previous version. If a File URI appears on a "higher" TOI than an aggregated object carrying it, the receiver SHALL assume that the individual file is the newer version. Also, an aggregated object on a higher TOI which contains a file previously described on a lower TOI SHALL be assumed to contain a newer (or equal) version.

To that purpose, the TOI assigned by the sender to each object MUST start with at least 1 and be incremented by one for each new object. This ensures that the receiver can unambiguously determine which instance of a certain file URI is not (obsolete (the one with the logically highest TOI)). This has no implication on sending or receiving order, only on allocation.

2. The physical file aggregation scheme offers a limited per-file filtering. The limitation is that a large part of the aggregated object may have to be received and processed before a FLUTE receiver can extract an individual file from it, when only a small subset of the aggregated files are of interest to a FLUTE receiver. It is therefore the responsibility of the FLUTE sender to create an homogeneous aggregation. The criteria to decide what files can be aggregated or not are out of scope of this document.

5. FLUTE version 1 backward compatibility

The XML Schema described in Section 4.2 is not backward compatible with the XML Schema described in the FLUTE version 1 specification. FLUTE version 1 does not allow the addition of new types of elements in the XML Schema.

An extra redirection to "extended" FDT Instances is introduced to be backward compatible with FLUTE version 1 specifications. This enables the combined use of FLUTE version 1 FDT Instance specifications in combination with the extended FDT schema of this specification. Yet if all receivers support this document's extended FDT schema, then the redirection mechanism is not required.

5.1 Redirection mechanism

The backward compatibility mechanism consists carrying the extended FDT Instances on a non-'0' TOIs. The TOI of the extended FDT Instances is signaled to the receivers with a new attribute, "FDTInstanceRedirection". The value of that attribute MUST remain the same during an entire file delivery session.

- o A receiver that only accepts FDT Instances conforming to the FLUTE version 1 specifications and that is not aware of the physical file aggregation scheme skips processing of the "FDTInstanceRedirection" attribute and therefore does not process the extended FDT Instances carried on the non-'0' TOI.
- o A receiver that is aware of the "file aggregation extension" processes the FDT redirection attribute, and therefore receives and processes the extended FDT Instances.

The extended FDT Instances, carried on a non-'0' TOI, have, as the non-extended FDT Instances, an FDT Instance ID. The FDT Instance ID is signaled to the receivers with an FDT Instance header as specified in FLUTE version 1.

FDT Instances on TOI '0' and on non-'0' TOI share the same FDT Instance IDs space. That means that an FDT Instance ID used on one TOI MUST NOT be used anymore for any other FDT Instance on any TOI. The FDT Instance ID MUST be incremented by one instead (wraparound considerations are the same as for FLUTE version 1). With this mechanism the most up-to-date (extended or non-extended) FDT Instance has always the greatest FDT Instance ID value. A consequence is that the FDT Instance ID values for one TOI are not necessarily contiguous.

6. Security Considerations

The security considerations that apply to FLUTE version 1, also apply to this document.

A malicious attacker may send forged FDT Instances. He could use the redirection mechanism (redirecting to false TOIs) or directly send forged FDT Instances, with false descriptions of aggregated objects. The attacker may use this mechanism to send malicious active content like a Trojan horse or some other type of virus within one aggregated object (as a whole) or within the aggregated files. It is thus **STRONGLY RECOMMENDED** that the FLUTE delivery service at the receiver does not have write access to the system files or directories, or any other critical areas, and that authentication schemes be used.

7. IANA Considerations

No information in this specification is directly subject to IANA registration. However, building blocks components used by ALC may introduce additional IANA considerations.

8. Acknowledgments

The authors gratefully acknowledge the contributions of Nabil Layaida. Thanks also for the helpful comments of Michael Luby and Thorsten Lohmar.

9. References

9.1 Normative References

- [1] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.
- [2] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [4] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, August 1998.
- [5] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.
- [6] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [7] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., and J. Crowcroft, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 3450, December 2002.
- [8] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., Handley, M., and J. Crowcroft, "Layered Coding Transport (LCT) Building Block", RFC 3451, December 2002.
- [9] Luby, M. and L. Vicisano, "Compact Forward Error Correction (FEC) Schemes", RFC 3695, February 2004.
- [10] Paila, T., Luby, M., Lehtonen, R., Roca, V., and R. Walsh, "FLUTE - File Delivery over Unidirectional Transport", RFC 3926, October 2004.
- [11] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C Recommendation, May 2001.
- [12] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001.

9.2 Informative References

- [13] Roca, V. and C. Neumann, "Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codec", INRIA Research Report Number 5225, June 2004.
- [14] Byers, J., Luby, M., Mitzenmacher, M., and A. Rege, "A digital fountain approach to reliable distribution of bulk data", ACM SIGCOMM 98, August 1998.

Authors' Addresses

Christoph Neumann
INRIA Rhone-Alpes
655, av. de l'Europe, Montbonnot
St Ismier cedex, 38334
France

Phone: +33 4 76 61 52 69
Email: christoph.neumann_(at)_inrialpes.fr

Vincent Roca
INRIA Rhone-Alpes
655, av. de l'Europe, Montbonnot
St Ismier cedex, 38334
France

Phone: +33 4 76 61 52 16
Email: vincent.roca_(at)_inrialpes.fr

Rod Walsh
Nokia
Visiokatu 1
Tampere, 33720
Finland

Email: rod.walsh_(at)_nokia.com

Appendix A. Example of FDT Instance (informative)

```

<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:fl="http://www.example.com/flute"
xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
Expires="2890842807">
  <File
    Content-Location="http://www.example.com/menu/tracklist.html"
    TOI="1"
    Content-Length="200"
    Content-Type="text/html">
    <Group>MP3_tracks</Group>
  </File>
  <File
    Content-Location="http://www.example.com/tracks/track1.mp3"
    TOI="2"
    Transfer-Length="6100"
    Content-Type="audio/mp3"
    Content-Encoding="gzip"
    Content-MD5="+VP5IrWploFkZWc11iLDdA=="
    Some-Private-Extension-Tag="abc123">
    <Group>MP3_tracks</Group>
  </File>
  <File
    Content-Location="http://www.example.com/index.html"
    TOI="3"
    Content-Length="100"
    Content-Type="text/html">
  </File>
</FDT-Instance>

```

A simple FDT Instance using the generalized grouping mechanism. The files on TOI='1' and TOI='2' are part of the group "MP3_Tracks".

Appendix B. Example of FDT Instance (informative)

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:fl="http://www.example.com/flute"
xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
Expires="2890842807">
  <File
    Content-Location="/AO1"
    TOI="1"
    Content-Length="830"
    Content-Type="multipart/mixed"
    Number-of-Files="2"/>
  <aggregatedFile
    Content-Location="http://www.example.com/menu/description.html"
    AOTOI="1"
    Content-Length="210"
    Content-Offset="100"
    Content-Type="text/html"/>
  <aggregatedFile
    Content-Location="http://www.example.com/menu/details.html"
    AOTOI="1"
    Transfer-Length="500"
    Content-Offset="320"
    Content-Encoding="gzip"
    Content-Type="text/html"/>
</FDT-Instance>
```

A simple FDT Instance using the physical file aggregation scheme. The files "<http://www.example.com/menu/description.html>" and "<http://www.example.com/menu/details.html>" are carried within the aggregated object, which has an TOI='1'.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

