Internet Engineering Task Force (IETF)
Request for Comments: 6726
Obsoletes: 3926
Category: Standards Track
ISSN: 2070-1721

T. Paila
Nokia
R. Walsh
Nokia/TUT
M. Luby
Qualcomm Technologies, Inc.
V. Roca
INRIA
R. Lehtonen
TeliaSonera

November 2012

# FLUTE - File Delivery over Unidirectional Transport

## Abstract

This document defines File Delivery over Unidirectional Transport (FLUTE), a protocol for the unidirectional delivery of files over the Internet, which is particularly suited to multicast networks. The specification builds on Asynchronous Layered Coding, the base protocol designed for massively scalable multicast distribution. This document obsoletes RFC 3926.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6726.

## Copyright Notice



## Table of Contents

## 1. Introduction

This document defines FLUTE version 2, a protocol for unidirectional delivery of files over the Internet. This specification is not backwards compatible with the previous experimental version defined in [RFC3926] (see Section 11 for details). The specification builds on Asynchronous Layered Coding (ALC), version 1 [RFC5775], the base protocol designed for massively scalable multicast distribution. ALC defines transport of arbitrary binary objects. For file delivery applications, mere transport of objects is not enough, however. The end systems need to know what the objects actually represent. This document specifies a technique called FLUTE -- a mechanism for signaling and mapping the properties of files to concepts of ALC in a way that allows receivers to assign those parameters for received objects. Consequently, throughout this document the term 'file' relates to an 'object' as discussed in ALC. Although this

specification frequently makes use of multicast addressing as an example, the techniques are similarly applicable for use with unicast addressing.

This document defines a specific transport application of ALC, adding the following specifications:

- Definition of a file delivery session built on top of ALC, including transport details and timing constraints.

- In-band signaling of the transport parameters of the ALC session.

- In-band signaling of the properties of delivered files.

- Details associated with the multiplexing of multiple files within a session.

This specification is structured as follows. Section 3 begins by defining the concept of the file delivery session. Following that, it introduces the File Delivery Table, which forms the core part of this specification. Further, it discusses multiplexing issues of transmission objects within a file delivery session. Section 4 describes the use of congestion control and channels with FLUTE. Section 5 defines how the Forward Error Correction (FEC) Object Transmission Information is to be delivered within a file delivery session. Section 6 defines the required parameters for describing file delivery sessions in a general case. Section 7 outlines security considerations regarding file delivery with FLUTE. Last, there are two informative appendices. Appendix A describes an envisioned receiver operation for the receiver of the file delivery session. Readers who want to see a simple example of FLUTE in operation should refer to Appendix A right away. Appendix B gives an example of a File Delivery Table.

This specification contains part of the definitions necessary to fully specify a Reliable Multicast Transport (RMT) protocol in accordance with [RFC2357].

This document obsoletes [RFC3926], which contained a previous version of this specification and was published in the "Experimental" category. This Proposed Standard specification is thus based on [RFC3926] and has been updated according to accumulated experience and growing protocol maturity since the publication of [RFC3926]. Said experience applies both to this specification itself and to congestion control strategies related to the use of this specification.

The differences between [RFC3926] and this document are listed in Section 11.

This document updates ALC [RFC5775] and Layered Coding Transport (LCT) [RFC5651] in the sense that it defines two new header extensions, EXT_FDT and EXT_CENC.

## 1.1. Applicability Statement

### 1.1.1. The Target Application Space

FLUTE is applicable to the delivery of large and small files to many hosts, using delivery sessions of several seconds or more. For instance, FLUTE could be used for the delivery of large software updates to many hosts simultaneously. It could also be used for continuous, but segmented, data such as time-lined text for subtitling -- potentially leveraging its layering inheritance from ALC and LCT to scale the richness of the session to the congestion status of the network. It is also suitable for the basic transport of metadata, for example, Session Description Protocol (SDP) [RFC4566] files that enable user applications to access multimedia sessions.

### 1.1.2. The Target Scale

Massive scalability is a primary design goal for FLUTE. IP multicast is inherently massively scalable, but the best-effort service that it provides does not provide session management functionality, congestion control, or reliability. FLUTE provides all of this by using ALC and IP multicast without sacrificing any of the inherent scalability of IP multicast.

### 1.1.3. Intended Environments

All of the environmental requirements and considerations that apply to the RMT building blocks used by FLUTE shall also apply to FLUTE. These are the ALC protocol instantiation [RFC5775], the LCT building block [RFC5651], and the FEC building block [RFC5052].

FLUTE can be used with both multicast and unicast delivery, but its primary application is for unidirectional multicast file delivery. FLUTE requires connectivity between a sender and receivers but does not require connectivity from receivers to a sender. Because of its low expectations, FLUTE works with most types of networks, including LANs, WANs, Intranets, the Internet, asymmetric networks, wireless networks, and satellite networks.

FLUTE is compatible with both IPv4 and IPv6, as no part of the packet is IP version specific. FLUTE works with both multicast models: Any-Source Multicast (ASM) [RFC1112] and Source-Specific Multicast (SSM) [PAPER.SSM].

FLUTE is applicable for both shared networks, such as the Internet, with a suitable congestion control building block; and provisioned/controlled networks, such as wireless broadcast radio systems, with a traffic-shaping building block.

### 1.1.4. Weaknesses

FLUTE congestion control protocols depend on the ability of a receiver to change multicast subscriptions between multicast groups supporting different rates and/or layered codings. If the network does not support this, then the FLUTE congestion control protocols may not be amenable to such a network.

FLUTE can also be used for point-to-point (unicast) communications. At a minimum, implementations of ALC MUST support the Wave and Equation Based Rate Control (WEBRC) [RFC3738] multiple-rate congestion control scheme [RFC5775]. However, since WEBRC has been designed for massively scalable multicast flows, it is not clear how appropriate it is to the particular case of unicast flows. Using a separate point-to-point congestion control scheme is another alternative. How to do that is outside the scope of the present document.

FLUTE provides reliability using the FEC building block. This will reduce the error rate as seen by applications. However, FLUTE does not provide a method for senders to verify the reception success of receivers, and the specification of such a method is outside the scope of this document.

## 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 [RFC2119].

The terms "object" and "transmission object" are consistent with the definitions in ALC [RFC5775] and LCT [RFC5651]. The terms "file" and "source object" are pseudonyms for "object".

## 3. File Delivery

Asynchronous Layered Coding [RFC5775] is a protocol designed for delivery of arbitrary binary objects. It is especially suitable for massively scalable, unidirectional multicast distribution. ALC provides the basic transport for FLUTE, and thus FLUTE inherits the requirements of ALC.

This specification is designed for the delivery of files. The core of this specification is to define how the properties of the files are carried in-band together with the delivered files.

As an example, let us consider a 5200-byte file referred to by "http://www.example.com/docs/file.txt". Using the example, the following properties describe the properties that need to be conveyed by the file delivery protocol.

* Identifier of the file, expressed as a URI [RFC3986]. The identifier MAY provide a location for the file. In the above example: "http://www.example.com/docs/file.txt".

* File name (usually, this can be concluded from the URI). In the above example: "file.txt".

* File type, expressed as Internet Media Types (often referred to as "Media Types"). In the above example: "text/plain".

* File size, expressed in octets. In the above example: "5200". If the file is content encoded, then this is the file size before content encoding.

* Content encoding of the file, within transport. In the above example, the file could be encoded using ZLIB [RFC1950]. In this case, the size of the transmission object carrying the file would probably differ from the file size. The transmission object size is delivered to receivers as part of the FLUTE protocol.

* Security properties of the file, such as digital signatures, message digests, etc. For example, one could use S/MIME [RFC5751] as the content encoding type for files with this authentication wrapper, and one could use XML Digital Signatures (XML-DSIG) [RFC3275] to digitally sign the file. XML-DSIG can also be used to provide tamper prevention, e.g., in the Content-Location field. Content encoding is applied to file data before FEC protection.

For each unique file, FLUTE encodes the attributes listed above and other attributes as children of an XML file element. A table of XML file elements is transmitted as a special file called a 'File Delivery Table' (FDT), which is further described in the next subsection and in Section 3.2.

## 3.1. File Delivery Session

ALC is a protocol instantiation of the Layered Coding Transport (LCT) building block [RFC5651]. Thus, ALC inherits the session concept of LCT. In this document, we will use the concept of the ALC/LCT session to collectively denote the interchangeable terms "ALC session" and "LCT session".

An ALC/LCT session consists of a set of logically grouped ALC/LCT channels associated with a single sender sending ALC/LCT packets for one or more objects. An ALC/LCT channel is defined by the combination of a sender and an address associated with the channel by the sender. A receiver joins a channel to start receiving the data packets sent to the channel by the sender, and a receiver leaves a channel to stop receiving data packets from the channel.

One of the fields carried in the ALC/LCT header is the Transport Session Identifier (TSI), an integer carried in a field of size 16, 32, or 48 bits (note that the TSI may be carried by other means, in which case it is absent from the LCT header [RFC5651]). The (source IP address, TSI) pair uniquely identifies a session. Note that the TSI is scoped by the IP address, so the same TSI may be used by several source IP addresses at once. Thus, the receiver uses the (source IP address, TSI) pair from each packet to uniquely identify the session sending each packet. When a session carries multiple objects, the Transmission Object Identifier (TOI) field within the ALC/LCT header names the object used to generate each packet. Note that each object is associated with a unique TOI within the scope of a session.

A FLUTE session consistent with this specification MUST use FLUTE version 2 as specified in this document. Thus, all sessions consistent with this specification MUST set the FLUTE version to 2. The FLUTE version is carried within the EXT_FDT Header Extension (defined in Section 3.4.1) in the ALC/LCT layer. A FLUTE session consistent with this specification MUST use ALC version 1 as specified in [RFC5775], and LCT version 1 as specified in [RFC5651].

If multiple FLUTE sessions are sent to a channel, then receivers MUST determine the FLUTE protocol version, based on version fields and the (source IP address, TSI) pair carried in the ALC/LCT header of the packet. Note that when a receiver first begins receiving packets, it

might not know the FLUTE protocol version, as not every LCT packet carries the EXT_FDT header (containing the FLUTE protocol version). A new receiver MAY keep an open binding in the LCT protocol layer between the TSI and the FLUTE protocol version, until the EXT_FDT header arrives. Alternatively, a new receiver MAY discover a binding between TSI and FLUTE protocol version via a session discovery protocol that is out of scope of this document.

If the sender's IP address is not accessible to receivers, then packets that can be received by receivers contain an intermediate IP address. In this case, the TSI is scoped by this intermediate IP address of the sender for the duration of the session. As an example, the sender may be behind a Network Address Translation (NAT) device that temporarily assigns an IP address for the sender. In this case, the TSI is scoped by the intermediate IP address assigned by the NAT. As another example, the sender may send its original packets using IPv6, but some portions of the network may not be IPv6 capable. Thus, there may be an IPv6-to-IPv4 translator that changes the IP address of the packets to a different IPv4 address. In this case, receivers in the IPv4 portion of the network will receive packets containing the IPv4 address, and thus the TSI for them is scoped by the IPv4 address. How the IP address of the sender to be used to scope the session by receivers is delivered to receivers, whether it is the sender's IP address or an intermediate IP address, is outside the scope of this document.

When FLUTE is used for file delivery over ALC, the ALC/LCT session is called a file delivery session, and the ALC/LCT concept of 'object' denotes either a 'file' or a 'File Delivery Table Instance' (Section 3.2).

Additionally, the following rules apply:

* The TOI field MUST be included in ALC packets sent within a FLUTE session, with the exception that ALC packets sent in a FLUTE session with the Close Session (A) flag set to 1 (signaling the end of the session) and that contain no payload (carrying no information for any file or FDT) SHALL NOT carry the TOI. See Section 5.1 of [RFC5651] for the LCT definition of the Close Session flag, and see Section 4.2 of [RFC5775] for an example of the use of a TOI within an ALC packet.

* The TOI value '0' is reserved for the delivery of File Delivery Table Instances. Each non-expired File Delivery Table Instance is uniquely identified by an FDT Instance ID within the EXT_FDT header defined in Section 3.4.1.

* Each file in a file delivery session MUST be associated with a TOI (>0) in the scope of that session.

* Information carried in the headers and the payload of a packet is scoped by the source IP address and the TSI. Information particular to the object carried in the headers and the payload of a packet is further scoped by the TOI for file objects, and is further scoped by both the TOI and the FDT Instance ID for FDT Instance objects.

### 3.2. File Delivery Table

The File Delivery Table (FDT) provides a means to describe various attributes associated with files that are to be delivered within the file delivery session. The following lists are examples of such attributes and are not intended to be mutually exclusive or exhaustive.

Attributes related to the delivery of a file:

- TOI value that represents the file

- FEC Object Transmission Information (including the FEC Encoding ID and, if relevant, the FEC Instance ID)

- Size of the transmission object carrying the file

- Aggregate rate of sending packets to all channels

Attributes related to the file itself:

- Name, Identification, and Location of file (specified by the URI)

- Media type of file

- Size of file

- Encoding of file

- Message digest of file

Some of these attributes MUST be included in the file description entry for a file; others are optional, as defined in Section 3.4.2.

Logically, the FDT is a set of file description entries for files to be delivered in the session. Each file description entry MUST include the TOI for the file that it describes and the URI identifying the file. The TOI carried in each file description entry

is how FLUTE names the ALC/LCT data packets used for delivery of the file. Each file description entry may also contain one or more descriptors that map the above-mentioned attributes to the file.

Each file delivery session MUST have an FDT that is local to the given session. The FDT MUST provide a file description entry mapped to a TOI for each file appearing within the session. An object that is delivered within the ALC session, but not described in the FDT, other than the FDT itself, is not considered a 'file' belonging to the file delivery session. This object received with an unmapped TOI (non-zero TOI that is not resolved by the FDT) SHOULD in general be ignored by a FLUTE receiver. The details of how to do that are out of scope of this specification.

Note that a client that joins an active file delivery session MAY receive data packets for a TOI > 0 before receiving any FDT Instance (see Section 3.3 for recommendations on how to limit the probability that this situation will occur). Even if the TOI is not mapped to any file description entry, this is hopefully a transient situation. When this happens, system performance might be improved by caching such packets within a reasonable time window and storage size. Such optimizations are use-case and implementation specific, and further details are beyond the scope of this document.

Within the file delivery session, the FDT is delivered as FDT Instances. An FDT Instance contains one or more file description entries of the FDT. Any FDT Instance can be equal to, be a subset of, be a superset of, overlap with, or complement any other FDT Instance. A certain FDT Instance may be repeated multiple times during a session, even after subsequent FDT Instances (with higher FDT Instance ID numbers) have been transmitted. Each FDT Instance contains at least a single file description entry and at most the exhaustive set of file description entries of the files being delivered in the file delivery session.

A receiver of the file delivery session keeps an FDT database for received file description entries. The receiver maintains the database, for example, upon reception of FDT Instances. Thus, at any given time the contents of the FDT database represent the receiver's current view of the FDT of the file delivery session. Since each receiver behaves independently of other receivers, it SHOULD NOT be assumed that the contents of the FDT database are the same for all the receivers of a given file delivery session.

Since the FDT database is an abstract concept, the structure and the maintenance of the FDT database are left to individual implementations and are thus out of scope of this specification.

### 3.3. Dynamics of FDT Instances within a File Delivery Session

The following rules define the dynamics of the FDT Instances within a file delivery session:

* For every file delivered within a file delivery session, there MUST be a file description entry included in at least one FDT Instance sent within the session. A file description entry contains at a minimum the mapping between the TOI and the URI.

* An FDT Instance MAY appear in any part of the file delivery session, and packets for an FDT Instance MAY be interleaved with packets for other files or other FDT Instances within a session.

* The TOI value of '0' MUST be reserved for delivery of FDT Instances. The use of other TOI values (i.e., an integer > 0) for FDT Instances is outside the scope of this specification.

* The FDT Instance is identified by the use of a new fixed-length LCT Header Extension, EXT_FDT (defined later in this section). Each non-expired FDT Instance is uniquely identified within the file delivery session by its FDT Instance ID, carried by the EXT_FDT Header Extension. Any ALC/LCT packet carrying an FDT Instance MUST include EXT_FDT.

* It is RECOMMENDED that an FDT Instance that contains the file description entry for a file be sent at least once before sending the described file within a file delivery session. This recommendation is intended to minimize the amount of file data that may be received by receivers in advance of the FDT Instance containing the entry for a file (such data must either be speculatively buffered or discarded). Note that this possibility cannot be completely eliminated, since the first transmission of FDT data might be lost.

* Within a file delivery session, any TOI > 0 MAY be described more than once. For example, a previous FDT Instance 0 describes a TOI of value '3'. Now, subsequent FDT Instances can either keep TOI '3' unmodified in the table, not include it, or augment the description. However, subsequent FDT Instances MUST NOT change the parameters already described for a specific TOI.

* An FDT Instance is valid until its expiration time. The expiration time is expressed within the FDT Instance payload as a UTF-8 decimal representation of a 32-bit unsigned integer. The value of this integer represents the 32 most significant bits of a 64-bit Network Time Protocol (NTP) [RFC5905] time value. These 32 bits provide an unsigned integer representing the time in

seconds relative to 0 hours 1 January 1900 in the case of the prime epoch (era 0) [RFC5905]. The handling of time wraparound (to happen in 2036) requires that the associated epoch be considered. In any case, both a sender and a receiver easily determine to which (136-year) epoch the FDT Instance expiration time value pertains by choosing the epoch for which the expiration time is closest in time to the current time.

Here is an example. Let us imagine that a new FLUTE session is started on February 7th, 2036, 0h, i.e., at NTP time 4,294,944,000, a few hours before the end of epoch 0. In order to define an FDT Instance valid for the next 48 hours, The FLUTE sender sets an expiry time of 149,504. This FDT Instance will expire exactly on February 9th, 2036, 0h. A client that receives this FDT Instance on the 7th, 0h, just after it has been sent, immediately understands that this value corresponds to epoch 1. A client that joins the session on February 8th, 0h, i.e., at NTP time 63,104, epoch 1, immediately understands that the 149,504 NTP timestamp corresponds to epoch 1.

* The space of FDT Instance IDs is limited by the associated field size (i.e., 20 bits) in the EXT_FDT Header Extension (Section 3.4.1). Therefore, senders should take care to always have a large enough supply of available FDT Instance IDs when specifying FDT expiration times.

* The receiver MUST NOT use a received FDT Instance to interpret packets received beyond the expiration time of the FDT Instance.

* A sender MUST use an expiration time in the future upon creation of an FDT Instance relative to its Sender Current Time (SCT).

* Any FEC Encoding ID MAY be used for the sending of FDT Instances. The default is to use the Compact No-Code FEC Encoding ID 0 [RFC5445] for the sending of FDT Instances. (Note that since FEC Encoding ID 0 is the default for FLUTE, this implies that Source Block Number and Encoding Symbol ID lengths both default to 16 bits each.)

* If the receiver does not support the FEC Scheme indicated by the FEC Encoding ID, the receiver MUST NOT decode the associated FDT.

* It is RECOMMENDED that the mechanisms used for file attribute delivery SHOULD achieve a delivery probability that is higher than the file recovery probability and the file attributes SHOULD be delivered at this higher priority before the delivery of the associated files begins.

Generally, a receiver needs to receive an FDT Instance describing a file before it is able to recover the file itself. In this sense, FDT Instances are of higher priority than files. Additionally, a FLUTE sender SHOULD assume that receivers will not receive all packets pertaining to FDT Instances. The way FDT Instances are transmitted has a large impact on satisfying the recommendation above. When there is a single file transmitted in the session, one way to satisfy the recommendation above is to repeatedly transmit on a regular enough basis FDT Instances describing the file while the file is being transmitted. If an FDT Instance is longer than one packet payload in length, it is RECOMMENDED that an FEC code that provides protection against loss be used for delivering this FDT Instance. When there are multiple files in a session concurrently being transmitted to receivers, the way the FDT Instances are structured and transmitted also has a large impact. As an example, a way to satisfy the recommendation above is to transmit an FDT Instance that describes all files currently being transmitted, and to transmit this FDT Instance reliably, using the same techniques as explained for the case when there is a single file transmitted in a session. If instead the concurrently transmitted files are described in separate FDT Instances, another way to satisfy this recommendation is to transmit all the relevant FDT Instances reliably, using the same techniques as explained for the case when there is a single file transmitted in a session.

In any case, how often the description of a file is sent in an FDT Instance, how often an FDT Instance is sent, and how much FEC protection is provided for an FDT Instance (if longer than one packet payload) are dependent on the particular application and are outside the scope of this document.

Sometimes the various attributes associated with files that are to be delivered within the file delivery session are sent out-of-band. The details of how this is done are out of the scope of this document. However, it is still RECOMMENDED that any out-of-band transmission be managed in such a way that a receiver will be able to recover the attributes associated with a file at least as reliably as the receiver is able to receive enough packets containing encoding symbols to recover the file. For example, the probability of a randomly chosen receiver being able to recover a given file can often be estimated based on a statistical model of reception conditions, the amount of data transmitted, and the properties of any Forward Error Correction in use. The recommendation above suggests that mechanisms used for file attribute delivery should achieve a higher delivery probability than the file recovery probability. The sender MAY also continue sending the various file attributes in-band, in addition to the out-of-band transmission.

### 3.4. Structure of FDT Instance Packets

FDT Instances are carried in ALC packets with TOI = 0 and with an additional REQUIRED LCT Header extension called the FDT Instance Header. The FDT Instance Header (EXT_FDT) contains the FDT Instance ID that uniquely identifies FDT Instances within a file delivery session. Placement of the FDT Instance Header is the same as that of any other LCT Header Extension. There MAY be other LCT Header Extensions in use.

The FDT Instance is encoded for transmission, like any other object, using an FEC Scheme (which MAY be the Compact No-Code FEC Scheme). The LCT Header Extensions are followed by the FEC Payload ID, and finally the Encoding Symbols for the FDT Instance, which contains one or more file description entries. An FDT Instance MAY span several ALC packets -- the number of ALC packets is a function of the file attributes associated with the FDT Instance. The FDT Instance Header is carried in each ALC packet carrying the FDT Instance. The FDT Instance Header is identical for all ALC/LCT packets for a particular FDT Instance.

The overall format of ALC/LCT packets carrying an FDT Instance is depicted in Figure 1 below. All integer fields are carried in "big-endian" or "network order" format (i.e., most significant byte (octet) first). As defined in [RFC5775], all ALC/LCT packets are sent using UDP.

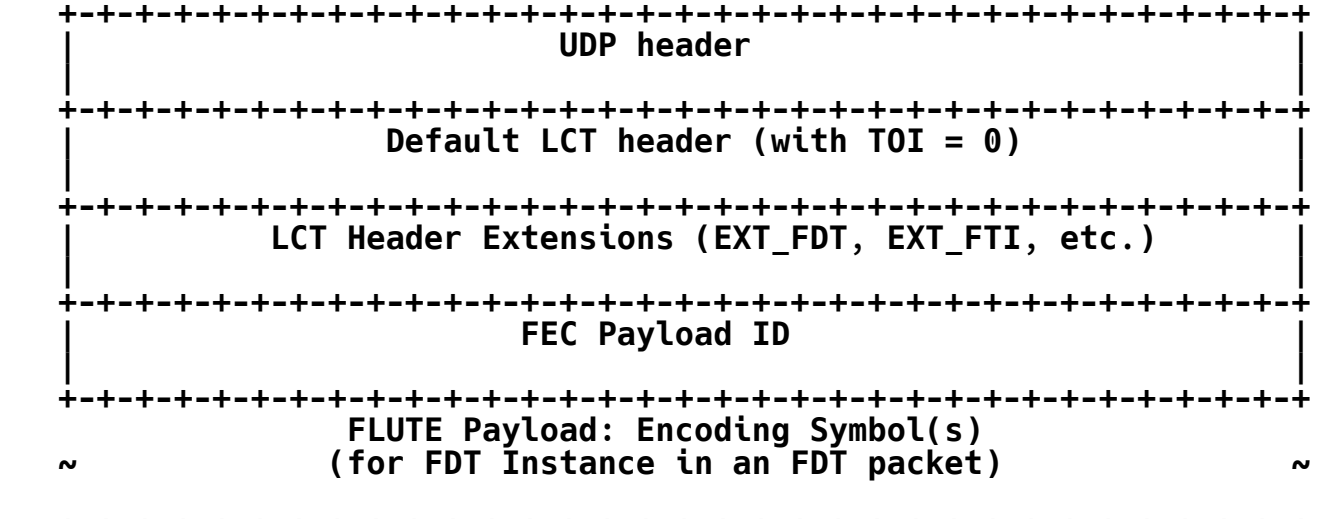


Figure 1: Overall FDT Packet

### 3.4.1. Format of FDT Instance Header

The FDT Instance Header (EXT_FDT) is a new fixed-length, ALC Protocol-Instantiation-specific LCT Header Extension [RFC5651]. The Header Extension Type (HET) for the extension is 192. Its format is defined below:

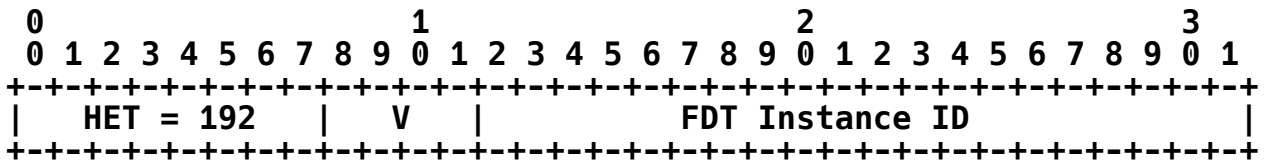


Figure 2: EXT_FDT Format

Version of FLUTE (V), 4 bits:

This document specifies FLUTE version 2. Hence, in any ALC packet that carries an FDT Instance and that belongs to the file delivery session as specified in this specification MUST set this field to '2'.

FDT Instance ID, 20 bits:

For each file delivery session, the numbering of FDT Instances starts from '0' and is incremented by one for each subsequent FDT Instance. After reaching the maximum value (2^20-1), the numbering starts from the smallest FDT Instance ID value assigned to an expired FDT Instance. When wraparound from a greater FDT Instance ID value to a smaller FDT Instance ID value occurs, the smaller FDT Instance ID value is considered logically higher than the greater FDT Instance ID value. Then, the subsequent FDT Instances are assigned the next available smallest FDT Instance ID value, in order to always keep the FDT Instance ID values logically increasing.

Senders MUST NOT reuse an FDT Instance ID value that is already in use for a non-expired FDT Instance. Sender behavior when all the FDT Instance IDs are used by non-expired FEC Instances is outside the scope of this specification and left to individual implementations of FLUTE. Receipt of an FDT Instance that reuses an FDT Instance ID value that is currently used by a non-expired FDT Instance MUST be considered an error case. Receiver behavior in this case (e.g., leave the session or ignore the new FDT Instance) is outside the scope of this specification and left to individual implementations of FLUTE. Receivers MUST be ready to handle FDT Instance ID wraparound and situations where missing FDT Instance IDs result in increments larger than one.

### 3.4.2. Syntax of FDT Instance

The FDT Instance contains file description entries that provide the mapping functionality described in Section 3.2 above.

The FDT Instance is an Extensible Markup Language (XML) structure that has a single root element "FDT-Instance". The "FDT-Instance" element MUST contain the "Expires" attribute, which provides the expiration time of the FDT Instance. In addition, the "FDT-Instance" element MAY contain the "Complete" attribute, a boolean that can be either set to '1' or 'true' for TRUE, or '0' or 'false' for FALSE. When TRUE, the "Complete" attribute signals that this "FDT Instance" includes the set of "File" entries that exhausts both the set of files delivered so far and the set of files to be delivered in the session. This implies that no new data will be provided in future FDT Instances within this session (i.e., that either FDT Instances with higher ID numbers will not be used or, if they are used, will only provide file parameters identical to those already given in this and previous FDT Instances). The "Complete" attribute is therefore used to provide a complete list of files in an entire FLUTE session (a "complete FDT"). Note that when all the FDT Instances received so far have no "Complete" attribute, the receiver MUST consider that the session is not complete and that new data MAY be provided in future FDT Instances. This is equivalent to receiving FDT Instances having the "Complete" attribute set to FALSE.

The "FDT-Instance" element MAY contain attributes that give common parameters for all files of an FDT Instance. These attributes MAY also be provided for individual files in the "File" element. Where the same attribute appears in both the "FDT-Instance" and the "File" elements, the value of the attribute provided in the "File" element takes precedence.

For each file to be declared in the given FDT Instance, there is a single file description entry in the FDT Instance. Each entry is represented by element "File", which is a child element of the FDT Instance structure.

The attributes of the "File" element in the XML structure represent the attributes given to the file that is delivered in the file delivery session. The value of the XML attribute name corresponds to the MIME field name, and the XML attribute value corresponds to the value of the MIME field body [RFC2045]. Each "File" element MUST contain at least two attributes: "TOI" and "Content-Location". "TOI" MUST be assigned a valid TOI value as described in Section 3.3. "Content-Location" [RFC2616] MUST be assigned a syntactically valid URI, as defined in [RFC3986], which identifies the file to be delivered. For example, it can be a URI with the "http" or "file"

URI scheme. Only one "Content-Location" attribute is allowed for each file. The "Content-Location" field MUST be considered a string that identifies a file (i.e., two different strings are two different identifiers). Any use of the "Content-Location" field for anything else other than to identify the object is out of scope of this specification. The semantics for any two "File" elements declaring the same "Content-Location" but differing "TOI" is that the element appearing in the FDT Instance with the greater FDT Instance ID is considered to declare a newer instance (e.g., version) of the same "File".

In addition to mandatory attributes, the "FDT-Instance" element and the "File" element MAY contain other attributes, of which the following are specifically pointed out:

* The attribute "Content-Type" SHOULD be included and, when present, MUST be used for the purpose defined in [RFC2616].

* Where the length is described, the attribute "Content-Length" MUST be used for the purpose defined in [RFC2616]. The transfer length is defined to be the length of the object transported in octets. It is often important to convey the transfer length to receivers, because the source block structure needs to be known for the FEC decoder to be applied to recover source blocks of the file, and the transfer length is often needed to properly determine the source block structure of the file. There generally will be a difference between the length of the original file and the transfer length if content encoding is applied to the file before transport, and thus the "Content-Encoding" attribute is used. If the file is not content encoded before transport (and thus the "Content-Encoding" attribute is not used), then the transfer length is the length of the original file, and in this case the "Content-Length" is also the transfer length. However, if the file is content encoded before transport (and thus the "Content-Encoding" attribute is used), e.g., if compression is applied before transport to reduce the number of octets that need to be transferred, then the transfer length is generally different than the length of the original file, and in this case the attribute "Transfer-Length" MAY be used to carry the transfer length.

* Whenever content encoding is applied, the attribute "Content-Encoding" MUST be included. Whenever the attribute "Content-Encoding" is included, it MUST be used as described in [RFC2616].

* Where the MD5 message digest is described, the attribute "Content-MD5" MUST be used for the purpose defined in [RFC2616]. Note that the goal is to provide a decoded object integrity service in cases where transmission and/or FLUTE/ALC processing errors may occur (the probability of collision is in that case negligible). It MUST NOT be regarded as a security mechanism (see Section 7 for information regarding security measures).

* The FEC Object Transmission Information attributes are described in Section 5.

The following specifies the XML Schema [XML-Schema-Part-1] [XML-Schema-Part-2] for the FDT Instance:

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:ietf:params:xml:ns:fdt"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:ietf:params:xml:ns:fdt"
           elementFormDefault="qualified">
  <xs:element name="FDT-Instance" type="FDT-InstanceType"/>
  <xs:complexType name="FDT-InstanceType">
    <xs:sequence>
      <xs:element name="File" type="FileType" maxOccurs="unbounded"/>
      <xs:any namespace="##other" 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:unsignedByte"
                  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:attribute name="FEC-OTI-Scheme-Specific-Info"
                    type="xs:base64Binary"
                    use="optional"/>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
    <xs:complexType name="FileType">
      <xs:sequence>
        <xs:any namespace="##other" 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-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:unsignedByte"
                    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:attribute name="FEC-OTI-Scheme-Specific-Info"
                    type="xs:base64Binary"
                    use="optional"/>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
  </xs:schema>
  END
```

Figure 3: XML Schema for the FDT Instance

Any valid FDT Instance MUST use the above XML Schema. This way, FDT provides extensibility to support private elements and private attributes within the file description entries. Those could be, for example, the attributes related to the delivery of the file (timing, packet transmission rate, etc.). Unsupported private elements and attributes SHOULD be silently ignored by a FLUTE receiver.

In case the basic FDT XML Schema is extended in terms of new descriptors (attributes or elements), for descriptors applying to a single file, those MUST be placed within the element "File". For descriptors applying to all files described by the current FDT Instance, those MUST be placed within the element "FDT-Instance". It is RECOMMENDED that the new attributes applied in the FDT be in the format of message header fields and be either defined in the HTTP/1.1 specification [RFC2616] or another well-known specification, or in an IANA registry [IANAheaderfields]. However, this specification doesn't prohibit the use of other formats to allow private attributes to be used when interoperability is not a concern.

### 3.4.3. Content Encoding of FDT Instance

The FDT Instance itself MAY be content encoded (e.g., compressed). This specification defines the FDT Instance Content Encoding Header (EXT_CENC). EXT_CENC is a new fixed-length LCT Header Extension [RFC5651]. The Header Extension Type (HET) for the extension is 193. If the FDT Instance is content encoded, EXT_CENC MUST be used to signal the content encoding type. In that case, the EXT_CENC Header Extension MUST be used in all ALC packets carrying the same FDT Instance ID. Consequently, when the EXT_CENC header is used, it MUST be used together with a proper FDT Instance Header (EXT_FDT). Within a file delivery session, FDT Instances that are not content encoded and FDT Instances that are content encoded MAY both appear. If

content encoding is not used for a given FDT Instance, EXT_CENC MUST NOT be used in any packet carrying the FDT Instance. The format of EXT_CENC is defined below:

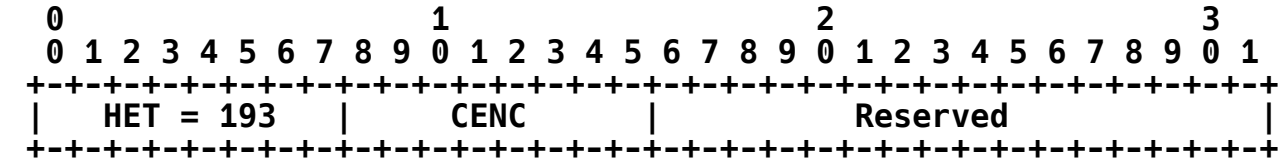


Figure 4: EXT_CENC Format

Content Encoding Algorithm (CENC), 8 bits:

This field signals the content encoding algorithm used in the FDT Instance payload. This subsection reserves the Content Encoding Algorithm values 0, 1, 2, and 3 for null, ZLIB [RFC1950], DEFLATE [RFC1951], and GZIP [RFC1952], respectively.

Reserved, 16 bits:

This field MUST be set to all '0's. This field MUST be ignored on reception.

## 3.5. Multiplexing of Files within a File Delivery Session

The delivered files are carried as transmission objects (identified with TOIs) in the file delivery session. All these objects, including the FDT Instances, MAY be multiplexed in any order and in parallel with each other within a session; i.e., packets for one file may be interleaved with packets for other files or other FDT Instances within a session.

Multiple FDT Instances MAY be delivered in a single session using TOI = 0. In this case, it is RECOMMENDED that the sending of a previous FDT Instance SHOULD end before the sending of the next FDT Instance starts. However, due to unexpected network conditions, packets for the FDT Instances might be interleaved. A receiver can determine which FDT Instance a packet contains information about, since the FDT Instances are uniquely identified by their FDT Instance ID carried in the EXT_FDT headers.

## 4. Channels, Congestion Control, and Timing

ALC/LCT has a concept of channels and congestion control. There are four scenarios in which FLUTE is envisioned to be applied.

(a) Use of a single channel and a single-rate congestion control protocol.

(b) Use of multiple channels and a multiple-rate congestion control protocol. In this case, the FDT Instances MAY be delivered on more than one channel.

(c) Use of a single channel without congestion control supplied by ALC, but only when in a controlled network environment where flow/congestion control is being provided by other means.

(d) Use of multiple channels without congestion control supplied by ALC, but only when in a controlled network environment where flow/congestion control is being provided by other means. In this case, the FDT Instances MAY be delivered on more than one channel.

When using just one channel for a file delivery session, as in (a) and (c), the notion of 'prior' and 'after' are intuitively defined for the delivery of objects with respect to their delivery times.

However, if multiple channels are used, as in (b) and (d), it is not straightforward to state that an object was delivered 'prior' to the other. An object may begin to be delivered on one or more of those channels before the delivery of a second object begins. However, the use of multiple channels/layers may mean that the delivery of the second object is completed before the first. This is not a problem when objects are delivered sequentially using a single channel. Thus, if the application of FLUTE has a mandatory or critical requirement that the first transmission object must complete 'prior' to the second one, it is RECOMMENDED that only a single channel be used for the file delivery session.

Furthermore, if multiple channels are used, then a receiver joined to the session at a low reception rate will only be joined to the lower layers of the session. Thus, since the reception of FDT Instances is of higher priority than the reception of files (because the reception of files depends on the reception of an FDT Instance describing it), the following are RECOMMENDED:

1. The layers to which packets for FDT Instances are sent SHOULD NOT be biased towards those layers to which lower-rate receivers are not joined. For example, it is okay to put all the packets for an FDT Instance into the lowest layer (if this layer carries enough packets to deliver the FDT to higher-rate receivers in a reasonable amount of time), but it is not okay to put all the packets for an FDT Instance into the higher layers that only higher-rate receivers will receive.

2. If FDT Instances are generally longer than one Encoding Symbol in length and some packets for FDT Instances are sent to layers that lower-rate receivers do not receive, an FEC encoding other than Compact No-Code FEC Encoding ID 0 [RFC5445] SHOULD be used to deliver FDT Instances. This is because in this case, even when there is no packet loss in the network, a lower-rate receiver will not receive all packets sent for an FDT Instance.

## 5. Delivering FEC Object Transmission Information

FLUTE inherits the use of the FEC building block [RFC5052] from ALC. When using FLUTE for file delivery over ALC, the FEC Object Transmission Information MUST be delivered in-band within the file delivery session. There are two methods to achieve this: the use of the ALC-specific LCT Header Extension EXT_FTI [RFC5775] and the use of the FDT. The latter method is specified in this section. The use of EXT_FTI requires repetition of the FEC Object Transmission Information to ensure reception (though not necessarily in every packet) and thus may entail higher overhead than the use of the FDT, but may also provide more timely delivery of the FEC Object Transmission Information.

The receiver of a file delivery session MUST support delivery of FEC Object Transmission Information using EXT_FTI for the FDT Instances carried using TOI value 0. For the TOI values other than 0, the receiver MUST support both methods: the use of EXT_FTI and the use of the FDT.

The FEC Object Transmission Information that needs to be delivered to receivers MUST be exactly the same whether it is delivered using EXT_FTI or using the FDT (or both). The FEC Object Transmission Information that MUST be delivered to receivers is defined by the FEC Scheme. This section describes the delivery using the FDT.

The FEC Object Transmission Information regarding a given TOI may be available from several sources. In this case, it is RECOMMENDED that the receiver of the file delivery session prioritize the sources in the following way (in order of decreasing priority).

1. FEC Object Transmission Information that is available in EXT_FTI.

2. FEC Object Transmission Information that is available in the FDT.

The FDT delivers FEC Object Transmission Information for each file using an appropriate attribute within the "FDT-Instance" or the "File" element of the FDT structure.

* "Transfer-Length" carries the "Transfer-Length" Object Transmission Information element defined in [RFC5052].

* "FEC-OTI-FEC-Encoding-ID" carries the "FEC Encoding ID" Object Transmission Information element defined in [RFC5052], as carried in the Codepoint field of the ALC/LCT header.

* "FEC-OTI-FEC-Instance-ID" carries the "FEC Instance ID" Object Transmission Information element defined in [RFC5052] for Under-Specified FEC Schemes.

* "FEC-OTI-Maximum-Source-Block-Length" carries the "Maximum-Source-Block-Length" Object Transmission Information element defined in [RFC5052], if required by the FEC Scheme.

* "FEC-OTI-Encoding-Symbol-Length" carries the "Encoding-Symbol-Length" Object Transmission Information element defined in [RFC5052], if required by the FEC Scheme.

* "FEC-OTI-Max-Number-of-Encoding-Symbols" carries the "Max-Number-of-Encoding-Symbols" Object Transmission Information element defined in [RFC5052], if required by the FEC Scheme.

* "FEC-OTI-Scheme-Specific-Info" carries the "encoded Scheme-specific FEC Object Transmission Information" as defined in [RFC5052], if required by the FEC Scheme.

In FLUTE, the FEC Encoding ID (8 bits) for a given TOI MUST be carried in the Codepoint field of the ALC/LCT header. When the FEC Object Transmission Information for this TOI is delivered through the FDT, then the associated "FEC-OTI-FEC-Encoding-ID" attribute and the Codepoint field of all packets for this TOI MUST be the same.

## 6. Describing File Delivery Sessions

To start receiving a file delivery session, the receiver needs to know transport parameters associated with the session. Interpreting these parameters and starting the reception therefore represent the entry point from which thereafter the receiver operation falls into the scope of this specification. According to [RFC5775], the transport parameters of an ALC/LCT session that the receiver needs to know are:

* The source IP address;

* The number of channels in the session;

* The destination IP address and port number for each channel in the session;

* The Transport Session Identifier (TSI) of the session;

* An indication that the session is a FLUTE session. The need to demultiplex objects upon reception is implicit in any use of FLUTE, and this fulfills the ALC requirement of an indication of whether or not a session carries packets for more than one object (all FLUTE sessions carry packets for more than one object).

Optionally, the following parameters MAY be associated with the session (note that the list is not exhaustive):

* The start time and end time of the session;

* FEC Encoding ID and FEC Instance ID when the default FEC Encoding ID 0 is not used for the delivery of the FDT;

* Content encoding format if optional content encoding of the FDT Instance is used, e.g., compression;

* Some information that tells receiver, in the first place, that the session contains files that are of interest;

* Definition and configuration of a congestion control mechanism for the session;

* Security parameters relevant for the session;

* FLUTE version number.

It is envisioned that these parameters would be described according to some session description syntax (such as SDP [RFC4566] or XML based) and held in a file that would be acquired by the receiver before the FLUTE session begins by means of some transport protocol (such as the Session Announcement Protocol (SAP) [RFC2974], email, HTTP [RFC2616], SIP [RFC3261], manual preconfiguration, etc.). However, the way in which the receiver discovers the above-mentioned parameters is out of scope of this document, as it is for LCT and ALC. In particular, this specification does not mandate or exclude any mechanism.

## 7. Security Considerations

### 7.1. Problem Statement

A content delivery system is potentially subject to attacks. Attacks may target:

* the network (to compromise the routing infrastructure, e.g., by creating congestion),

* the Content Delivery Protocol (CDP) (e.g., to compromise the normal behavior of FLUTE), or

* the content itself (e.g., to corrupt the files being transmitted).

These attacks can be launched either:

* against the data flow itself (e.g., by sending forged packets),

* against the session control parameters (e.g., by corrupting the session description, the FDT Instances, or the ALC/LCT control parameters) that are sent either in-band or out-of-band, or

* against some associated building blocks (e.g., the congestion control component).

In the following sections, we provide more details on these possible attacks and sketch some possible countermeasures. We provide recommendations in Section 7.5.

### 7.2. Attacks against the Data Flow

Let us consider attacks against the data flow first. At the least, the following types of attacks exist:

* attacks that are meant to give access to a confidential file (e.g., in the case of non-free content) and

* attacks that try to corrupt the file being transmitted (e.g., to inject malicious code within a file, or to prevent a receiver from using a file, which is a kind of denial of service (DoS)).

#### 7.2.1. Access to Confidential Files

Access control to the file being transmitted is typically provided by means of encryption. This encryption can be done over the whole file, i.e., before applying FEC protection (e.g., by the content provider, before submitting the file to FLUTE), or can be done on a packet-by-packet basis (e.g., when IPsec/ESP [RFC4303] is used; see Section 7.5). If confidentiality is a concern, it is RECOMMENDED that one of these solutions be used.

#### 7.2.2. File Corruption

Protection against corruptions (e.g., if an attacker sends forged packets) is achieved by means of a content integrity verification/sender authentication scheme. This service can be provided at the file level, i.e., before applying content encoding and FEC encoding. In that case, a receiver has no way to identify which symbol(s) is(are) corrupted if the file is detected as corrupted. This service can also be provided at the packet level, i.e., after applying content encoding and FEC encoding, on a packet-by-packet basis. In this case, after removing all corrupted packets, the file may be in some cases recovered from the remaining correct packets.

Integrity protection applied at the file level has the advantage of lower overhead, since only relatively few bits are added to provide the integrity protection compared to the file size. However, it has the disadvantage that it cannot distinguish between correct packets and corrupt packets, and therefore correct packets, which may form the majority of packets received, may be unusable. Integrity protection applied at the packet level has the advantage that it can distinguish between correct and corrupt packets, at the cost of additional per-packet overhead.

Several techniques can provide this source authentication/content integrity service:

* At the file level, the file MAY be digitally signed (e.g., by using RSA Probabilistic Signature Scheme Public-Key Cryptography Standards version 1.5 (RSASSA-PKCS1-v1_5) [RFC3447]). This signature enables a receiver to check the file's integrity once the file has been fully decoded. Even if digital signatures are computationally expensive, this calculation occurs only once per file, which is usually acceptable.

* At the packet level, each packet can be digitally signed [RFC6584]. A major limitation is the high computational and transmission overheads that this solution requires. To avoid this problem, the signature may span a set of symbols (instead of a single one) in order to amortize the signature calculation, but if a single symbol is missing, the integrity of the whole set cannot be checked.

* At the packet level, a Group-Keyed Message Authentication Code (MAC) [RFC2104] [RFC6584] scheme can be used; an example is using HMAC-SHA-256 with a secret key shared by all the group members, senders, and receivers. This technique creates a cryptographically secured digest of a packet that is sent along with the packet. The Group-Keyed MAC scheme does not create prohibitive processing load or transmission overhead, but it has a major limitation: it only provides a group authentication/integrity service, since all group members share the same secret group key, which means that each member can send a forged packet. It is therefore restricted to situations where group members are fully trusted (or in association with another technique as a pre-check).

* At the packet level, Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [RFC4082] [RFC5776] is an attractive solution that is robust to losses, provides a true authentication/integrity service, and does not create any prohibitive processing load or transmission overhead. However, checking a packet requires a small delay (a second or more) after its reception.

* At the packet level, IPsec/ESP [RFC4303] can be used to check the integrity and authenticate the sender of all the packets being exchanged in a session (see Section 7.5).

Techniques relying on public key cryptography (digital signatures and TESLA during the bootstrap process, when used) require that public keys be securely associated to the entities. This can be achieved by

a Public Key Infrastructure (PKI), or by a Pretty Good Privacy (PGP) Web of Trust, or by pre-distributing the public keys of each group member.

Techniques relying on symmetric key cryptography (Group-Keyed MAC) require that a secret key be shared by all group members. This can be achieved by means of a group key management protocol, or simply by pre-distributing the secret key (but this manual solution has many limitations).

It is up to the developer and deployer, who know the security requirements and features of the target application area, to define which solution is the most appropriate. Nonetheless, in case there is any concern of the threat of file corruption, it is RECOMMENDED that at least one of these techniques be used.

### 7.3. Attacks against the Session Control Parameters and Associated Building Blocks

Let us now consider attacks against the session control parameters and the associated building blocks. The attacker has at least the following opportunities to launch an attack:

* the attack can target the session description,

* the attack can target the FDT Instances,

* the attack can target the ALC/LCT parameters, carried within the LCT header, or

* the attack can target the FLUTE associated building blocks (e.g., the multiple-rate congestion control protocol).

The consequences of these attacks are potentially serious, since they might compromise the behavior of the content delivery system itself.

#### 7.3.1. Attacks against the Session Description

A FLUTE receiver may potentially obtain an incorrect session description for the session. The consequence of this is that legitimate receivers with the wrong session description are unable to correctly receive the session content, or that receivers inadvertently try to receive at a much higher rate than they are capable of, thereby possibly disrupting other traffic in the network.

To avoid these problems, it is RECOMMENDED that measures be taken to prevent receivers from accepting incorrect session descriptions. One such measure is source authentication to ensure that receivers only

accept legitimate session descriptions from authorized senders. How these measures are achieved is outside the scope of this document, since this session description is usually carried out-of-band.

### 7.3.2. Attacks against the FDT Instances

Corrupting the FDT Instances is one way to create a DoS attack. For example, the attacker changes the MD5 sum associated to a file. This possibly leads a receiver to reject the files received, no matter whether the files have been correctly received or not.

Corrupting the FDT Instances is also a way to make the reception process more costly than it should be. This can be achieved by changing the FEC Object Transmission Information when the FEC Object Transmission Information is included in the FDT Instance. For example, an attacker may corrupt the FDT Instance in such a way that Reed-Solomon over GF(2^^16) would be used instead of GF(2^^8) with FEC Encoding ID 2. This may significantly increase the processing load while compromising FEC decoding.

More generally, because FDT Instance data is structured using the XML language by means of an XML media type, many of the security considerations described in [RFC3023] and [RFC3470] also apply to such data.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender's identity of the FDT Instances. To that purpose, one of the countermeasures mentioned above (Section 7.2.2) SHOULD be used. These measures will either be applied on a packet level or globally over the whole FDT Instance object. Additionally, XML digital signatures [RFC3275] are a way to protect the FDT Instance by digitally signing it. When there is no packet-level integrity verification scheme, it is RECOMMENDED to rely on XML digital signatures of the FDT Instances.

### 7.3.3. Attacks against the ALC/LCT Parameters

By corrupting the ALC/LCT header (or header extensions), one can execute attacks on the underlying ALC/LCT implementation. For example, sending forged ALC packets with the Close Session flag (A) set to one can lead the receiver to prematurely close the session. Similarly, sending forged ALC packets with the Close Object flag (B) set to one can lead the receiver to prematurely give up the reception of an object.

It is therefore RECOMMENDED that measures be taken to guarantee the integrity and to check the sender's identity of the ALC packets received. To that purpose, one of the countermeasures mentioned above (Section 7.2.2) SHOULD be used.

### 7.3.4. Attacks against the Associated Building Blocks

Let us first focus on the congestion control building block, which may be used in the ALC session. A receiver with an incorrect or corrupted implementation of the multiple-rate congestion control building block may affect the health of the network in the path between the sender and the receiver. That may also affect the reception rates of other receivers who joined the session.

When the congestion control building block is applied with FLUTE, it is RECOMMENDED that receivers be required to identify themselves as legitimate before they receive the session description needed to join the session. How receivers identify themselves as legitimate is outside the scope of this document. If authenticating a receiver does not prevent this receiver from launching an attack, this authentication will enable the network operator to identify him and to take countermeasures.

When the congestion control building block is applied with FLUTE, it is also RECOMMENDED that a packet-level authentication scheme be used, as explained in Section 7.2.2. Some of them, like TESLA, only provide a delayed authentication service, whereas congestion control requires a rapid reaction. It is therefore RECOMMENDED [RFC5775] that a receiver using TESLA quickly reduce its subscription level when the receiver believes that congestion did occur, even if the packet has not yet been authenticated. Therefore, TESLA will not prevent DoS attacks where an attacker makes the receiver believe that congestion occurred. This is an issue for the receiver, but this will not compromise the network. Other authentication methods that do not feature this delayed authentication could be preferred, or a Group-Keyed MAC scheme could be used in parallel with TESLA to prevent attacks launched from outside of the group.

## 7.4. Other Security Considerations

The security considerations that apply to, and are described in, ALC [RFC5775], LCT [RFC5651], and FEC [RFC5052] also apply to FLUTE, as FLUTE builds on those specifications. In addition, any security considerations that apply to any congestion control building block used in conjunction with FLUTE also apply to FLUTE.

Even if FLUTE defines a purely unidirectional delivery service, without any feedback information that would be sent to the sender, security considerations MAY require bidirectional communications. For instance, if an automated key management scheme is used, a bidirectional point-to-point channel is often needed to establish a shared secret between each receiver and the sender. Each shared secret can then be used to distribute additional keys to the associated receiver (e.g., traffic encryption keys).

As an example, [MBMSsecurity] details a complete security framework for the Third Generation Partnership Project (3GPP) Multimedia Broadcast/Multicast Service (MBMS) that relies on FLUTE/ALC for Download Sessions. It relies on bidirectional point-to-point communications for User Equipment authentication and for key distribution, using the Multimedia Internet KEYing (MIKEY) protocol [RFC3830]. Because this security framework is specific to this use case, it cannot be reused as such for generic security recommendations in this specification. Instead, the following section introduces minimum security recommendations.

### 7.5. Minimum Security Recommendations

We now introduce a mandatory-to-implement, but not necessarily to use, security configuration, in the sense of [RFC3365]. Since FLUTE relies on ALC/LCT, it inherits the "baseline secure ALC operation" of [RFC5775]. More precisely, security is achieved by means of IPsec/ESP in transport mode. [RFC4303] explains that ESP can be used to potentially provide confidentiality, data origin authentication, content integrity, anti-replay, and (limited) traffic flow confidentiality. [RFC5775] specifies that the data origin authentication, content integrity, and anti-replay services SHALL be supported, and that the confidentiality service is RECOMMENDED. If a short-lived session MAY rely on manual keying, it is also RECOMMENDED that an automated key management scheme be used, especially in the case of long-lived sessions.

Therefore, the RECOMMENDED solution for FLUTE provides per-packet security, with data origin authentication, integrity verification, and anti-replay. This is sufficient to prevent most of the in-band attacks listed above. If confidentiality is required, a per-packet encryption SHOULD also be used.

## 8. IANA Considerations

This specification contains five separate items upon which IANA has taken action:

1. Registration of the FDT Instance XML Namespace.

2. Registration of the FDT Instance XML Schema.

3. Registration of the application/fdt+xml Media Type.

4. Registration of the Content Encoding Algorithms.

5. Registration of two LCT Header Extension Types (EXT_FDT and EXT_CENC).

### 8.1. Registration of the FDT Instance XML Namespace

IANA has registered the following new XML Namespace in the IETF XML "ns" registry [RFC3688] at http://www.iana.org/assignments/xml-registry/ns.html.

URI: urn:ietf:params:xml:ns:fdt

Registrant Contact: Toni Paila (toni.paila@gmail.com)

XML: N/A

### 8.2. Registration of the FDT Instance XML Schema

IANA has registered the following in the IETF XML "schema" registry [RFC3688] at http://www.iana.org/assignments/xml-registry/schema.html.

URI: urn:ietf:params:xml:schema:fdt

Registrant Contact: Toni Paila (toni.paila@gmail.com)

XML: The XML Schema specified in Section 3.4.2

### 8.3. Registration of the application/fdt+xml Media Type

IANA has registered the following in the "Application Media Types" registry at http://www.iana.org/assignments/media-types/application/.

Type name: application

Subtype name: fdt+xml

Required parameters: none

Optional parameters: charset="utf-8"

Encoding considerations: binary (the FLUTE file delivery protocol does not impose any restriction on the objects it carries and in particular on the FDT Instance itself)

Restrictions on usage: none

Security considerations: fdt+xml data is passive and does not generally represent a unique or new security threat. However, there is some risk in sharing any kind of data, in that unintentional information may be exposed, and that risk applies to fdt+xml data as well.

Interoperability considerations: None

Published specification: [RFC6726], especially noting Section 3.4.2. The specified FDT Instance functions as an actual media format of use to the general Internet community, and thus media type registration under the Standards Tree is appropriate to maximize interoperability.

Applications that use this media type: file and object delivery applications and protocols (e.g., FLUTE).

Additional information:

```
    Magic number(s): none
    File extension(s): ".fdt" (e.g., if there is a need to store an
                        FDT Instance as a file)
    Macintosh File Type Code(s): none
```

Person and email address to contact for further information:
Toni Paila (toni.paila@gmail.com)

Intended usage: Common

Author/Change controller: IETF

### 8.4. Creation of the FLUTE Content Encoding Algorithms Registry

IANA has created a new registry, "FLUTE Content Encoding Algorithms", with a reference to [RFC6726]; see Section 3.4.3. The registry entries consist of a numeric value from 0 to 255, inclusive, and may be registered using the Specification Required policy [RFC5226].

The initial contents of the registry are as follows, with unspecified values available for new registrations:

| Value | Algorithm Name | Reference |
|---|---|---|
| 0 | null | [RFC6726] |
| 1 | ZLIB | [RFC1950] |
| 2 | DEFLATE | [RFC1951] |
| 3 | GZIP | [RFC1952] |

### 8.5. Registration of LCT Header Extension Types

IANA has registered two new entries in the "Layered Coding Transport (LCT) Header Extension Types" registry [RFC5651], as follows:

| Number | Name | Reference |
|---|---|---|
| 192 | EXT_FDT | [RFC6726] Section 3.4.1 |
| 193 | EXT_CENC | [RFC6726] Section 3.4.3 |

## 9. Acknowledgments

The following persons have contributed to this specification: Brian Adamson, Mark Handley, Esa Jalonen, Roger Kermode, Juha-Pekka Luoma, Topi Pohjolainen, Lorenzo Vicisano, Mark Watson, David Harrington, Ben Campbell, Stephen Farrell, Robert Sparks, Ronald Bonica, Francis Dupont, Peter Saint-Andre, Don Gillies, and Barry Leiba. The authors would like to thank all the contributors for their valuable work in reviewing and providing feedback regarding this specification.

## 10. Contributors

Jani Peltotalo
Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FIN-33101
Finland
EMail: jani.peltotalo@tut.fi

Sami Peltotalo
Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FIN-33101
Finland
EMail: sami.peltotalo@tut.fi

Magnus Westerlund
Ericsson Research
Ericsson AB
SE-164 80 Stockholm
Sweden
EMail: magnus.westerlund@ericsson.com

Thorsten Lohmar
Ericsson Research (EDD)
Ericsson Allee 1
52134 Herzogenrath
Germany
EMail: thorsten.lohmar@ericsson.com

## 11. Change Log

### 11.1. RFC 3926 to This Document

Incremented the FLUTE protocol version from 1 to 2, due to concerns about backwards compatibility. For instance, the LCT header changed between RFC 3451 and [RFC5651]. In RFC 3451, the T and R fields of the LCT header indicate the presence of Sender Current Time and Expected Residual Time, respectively. In [RFC5651], these fields MUST be set to zero and MUST be ignored by receivers (instead, the EXT_TIME Header Extensions can convey this information if needed). Thus, [RFC5651] is not backwards compatible with RFC 3451, even though both use LCT version 1. FLUTE version 1 as specified in [RFC3926] MUST use RFC 3451. FLUTE version 2 as specified in this document MUST use [RFC5651]. Therefore, an implementation that relies on [RFC3926] and RFC 3451 will not be backwards compatible with FLUTE as specified in this document.

Updated dependencies to other RFCs to revised versions; e.g., changed ALC reference from RFC 3450 to [RFC5775], changed LCT reference from RFC 3451 to [RFC5651], etc.

Added clarification for the use of FLUTE for unicast communications in Section 1.1.4.

Clarified how to reliably deliver the FDT in Section 3.3 and the possibility of using out-of-band delivery of FDT information.

Clarified how to address FDT Instance expiration time wraparound with the notion of the NTPv4 "epoch" in Section 3.3.

Clarified what should be considered erroneous situations in Section 3.4.1 (definition of FDT Instance ID). In particular, a receiver MUST be ready to handle FDT Instance ID wraparounds and missing FDT Instances.

Updated Section 7.5 to define IPsec/ESP as a mandatory-to-implement security solution.

Removed the 'Statement of Intent' from Section 1. The statement of intent was meant to clarify the "Experimental" status of [RFC3926]. It does not apply to this document.

Added clarification of "XML-DSIG" near the end of Section 3.

In Section 3.2, replaced "complete FDT" with text that is more descriptive.

Clarified Figure 1 with regard to "Encoding Symbol(s) for FDT Instance".

Clarified the text regarding FDT Instance ID wraparound at the end of Section 3.4.1.

Clarified "complete FDT" in Section 3.4.2.

Added semantics for the case where two TOIs refer to the same Content-Location. It is now in line with the way that 3GPP and Digital Video Broadcasting (DVB) standards interpret this case.

In Section 3.4.2, the XML Schema of the FDT Instance was modified per advice from various sources. For example, extension by element was missing but is now supported. Also, the namespace definition was changed to URN format.

Clarified FDT-schema extensibility at the end of Section 3.4.2.

The CENC value allocation has been added at the end of Section 3.4.3.

Section 5 has been modified so that EXT_FTI and the FEC issues were replaced by a reference to the ALC specification [RFC5775].

Added a clarifying paragraph on the use of the Codepoint field at the end of Section 5.

Reworked Section 8 -- IANA Considerations; it now contains six IANA registration requests:

* Registration of the FDT Instance XML Namespace.

* Registration of the FDT Instance XML Schema.

* Registration of the application/fdt+xml Media Type.

* Registration of the Content Encoding Algorithms.

* Registration of two LCT Header Extension Types and corresponding values in the LCT Header Extension Types Registry (192 for EXT_FDT and 193 for EXT_CENC).

Added Section 10 -- Contributors.

Revised lists of both Normative and Informative references.

Added a clarification that the receiver should ignore reserved bits of Header Extension type 193 upon reception.

Elaborated on what kinds of networks cannot support FLUTE congestion control (Section 1.1.4).

In Section 3.2, changed "several" (meaning 3-n vs. "couple" = 2) to "multiple" (meaning 2-n).

Moved the requirement in Section 3.3 (to send FDT more reliably than files) to a bulleted RECOMMENDED requirement, making check-off easier for testers.

In Section 3.3, sharpened the definition that future FDT file instances can "augment" (meaning enhance) rather than "complement" (sometimes meaning negate, which is not allowed) the file parameters.

Elaborated in Sections 3.3 and 4 that FEC Encoding ID = 0 is Compact No-Code FEC, so that the reader doesn't have to search other RFCs to understand these protocol constants used by FLUTE.

Required in Section 3.3 that FLUTE receivers SHALL NOT attempt to decode FDTs if they do not understand the FEC Encoding ID.

Removed the restriction of Section 3.3, in bullet #4, that TOI = 0 for the FDT, to be consistent with Appendix A step 6 and elsewhere. An FDT is signaled by an FDT Instance ID, NOT only by TOI = 0.

Standardized on the term "expiration time", and avoided using the redundant and possibly confusing term "expiry time".

To interwork with experimental FLUTE, stipulated in Section 3.1 that only 1 instantiation of all 3 protocols -- FLUTE, ALC, and LCT -- can be associated with a session (source IP Address, TSI), and mentioned in Section 6 that one may (optionally) derive the FLUTE version from the file delivery session description.

Used a software writing tool to lower the reading grade level and simplify Section 3.1.

## 12. References

### 12.1. Normative References

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

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, August 2007.

[RFC5445] Watson, M., "Basic Forward Error Correction (FEC) Schemes", RFC 5445, March 2009.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC2616] 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.

[XML-Schema-Part-1]
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", W3C Recommendation, October 2004, <http://www.w3.org/TR/xmlschema-1/>.

[XML-Schema-Part-2]
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, <http://www.w3.org/TR/xmlschema-2/>.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, May 2008.

[RFC3738] Luby, M. and V. Goyal, "Wave and Equation Based Rate Control (WEBRC) Building Block", RFC 3738, April 2004.

Note: The RFC 3738 reference is to a target document of a lower maturity level. Some caution should be used, since it may be less stable than the present document.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

## 12.2. Informative References

[RFC3926] Paila, T., Luby, M., Lehtonen, R., Roca, V., and R. Walsh, "FLUTE - File Delivery over Unidirectional Transport", RFC 3926, October 2004.

[RFC2357] Mankin, A., Romanow, A., Bradner, S., and V. Paxson, "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols", RFC 2357, June 1998.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, January 2003.

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

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

[IANAheaderfields] IANA, "Message Header Fields", <http://www.iana.org/protocols>.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, August 1989.

[PAPER.SSM] Holbrook, H., "A Channel Model for Multicast", Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California, August 2001.

[RFC3365] Schiller, J., "Strong Security Requirements for Internet Engineering Task Force Standard Protocols", BCP 61, RFC 3365, August 2002.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC3275] Eastlake 3rd, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

[RFC5776] Roca, V., Francillon, A., and S. Faurite, "Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 5776, April 2010.

[RFC6584] Roca, V., "Simple Authentication Schemes for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 6584, April 2012.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[MBMSsecurity] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security of Multimedia Broadcast/Multicast Service (MBMS) (Release 10)", December 2010, <http://www.3gpp.org/ftp/Specs/archive/33_series/33.246/>.

## Appendix A. Receiver Operation (Informative)

This section gives an example of how the receiver of the file delivery session may operate. Instead of a detailed state-by-state specification, the following should be interpreted as a rough sequence of an envisioned file delivery receiver.

1. The receiver obtains the description of the file delivery session identified by the (source IP address, Transport Session Identifier) pair. The receiver also obtains the destination IP addresses and respective ports associated with the file delivery session.

2. The receiver joins the channels in order to receive packets associated with the file delivery session. The receiver may schedule this join operation utilizing the timing information contained in a possible description of the file delivery session.

3. The receiver receives ALC/LCT packets associated with the file delivery session. The receiver checks that the packets match the declared Transport Session Identifier. If not, the packets are silently discarded.

4. While receiving, the receiver demultiplexes packets based on their TOI and stores the relevant packet information in an appropriate area for recovery of the corresponding file. Multiple files can be reconstructed concurrently.

5. The receiver recovers an object. An object can be recovered when an appropriate set of packets containing Encoding Symbols for the transmission object has been received. An appropriate set of packets is dependent on the properties of the FEC Encoding ID and FEC Instance ID, and on other information contained in the FEC Object Transmission Information.

6. Objects with TOI = 0 are reserved for FDT Instances. All FDT Instances are signaled by including an EXT_FDT Header Extension in the LCT header. The EXT_FDT header contains an FDT Instance ID (i.e., an FDT version number). If the object has an FDT Instance ID 'N', the receiver parses the payload of the instance 'N' of the FDT and updates its FDT database accordingly.

7. If the object recovered is not an FDT Instance but a file, the receiver looks up its FDT database to get the properties described in the database, and assigns the file the given properties. The receiver also checks that the received content

length matches with the description in the database. Optionally, if an MD5 checksum has been used, the receiver checks that the calculated MD5 matches the description in the FDT database.

8. The actions the receiver takes with imperfectly received files (missing data, mismatching content integrity digest, etc.) are outside the scope of this specification. When a file is recovered before the associated file description entry is available, a possible behavior is to wait until an FDT Instance is received that includes the missing properties.

9. If the file delivery session end time has not been reached, go back to step 3. Otherwise, end.

## 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"
  xsi:schemaLocation="urn:ietf:params:xml:ns:fdt
                      ietf-flute-fdt.xsd"
  Expires="2890842807">
  <File
    Content-Location="http://www.example.com/menu/tracklist.html"
    TOI="1"
    Content-Type="text/html"/>
  <File
    Content-Location="http://www.example.com/tracks/track1.mp3"
    TOI="2"
    Content-Length="6100"
    Content-Type="audio/mp3"
    Content-Encoding="gzip"
    Content-MD5="+VP5IrWploFkZWc11iLDdA=="
    Some-Private-Extension-Tag="abc123"/>
</FDT-Instance>
```

## Authors' Addresses

Toni Paila
Nokia
Itamerenkatu 11-13
Helsinki 00180
Finland

EMail: toni.paila@gmail.com

Rod Walsh
Nokia/Tampere University of Technology
P.O. Box 553 (Korkeakoulunkatu 1)
Tampere FI-33101
Finland

EMail: roderick.walsh@tut.fi

Michael Luby
Qualcomm Technologies, Inc.
2030 Addison Street, Suite 420
Berkeley, CA 94704
USA

EMail: luby@qti.qualcomm.com

Vincent Roca
INRIA
655, av. de l'Europe
Inovallee; Montbonnot
ST ISMIER cedex 38334
France

EMail: vincent.roca@inria.fr

Rami Lehtonen
TeliaSonera
Hatanpaankatu 1
Tampere FIN-33100
Finland

EMail: rami.lehtonen@teliasonera.com