FCAST: Massively Scalable File Delivery on Top of Unidirectional Transport Channels

Vincent Roca

FCAST: Massively Scalable File Delivery on Top of Unidirectional Transport Channels

INRIA Rhône-Alpes, Planète research team, France
vincent.roca
http://planete.inrialpes.fr/roca/

October 11, 2006

1 Introduction and Motivations

This document introduces the FCAST object (e.g. file) delivery application on top of unidirectional channels. FCAST is a highly scalable application that provides a reliable object delivery service. The current specification is a follow-up of work carried out in [6], implemented in the MCLv3 open-source project [4], and it offers an alternative to the FLUTE [3] standard.

Since FCAST is built on top of ALC/LCT [1, 2], it inherits the concepts defined there, in particular: the concept of "PUSH" and "ON-DEMAND" delivery modes, the concept of Transport Object ID (`TOI`) that uniquely identify the objects being transported in an ALC session, the concept of header extension that provide an efficient way of extending the ALC/LCT packet header, and the concept of object close (LCT flag `B`) to inform the clients that no more data packet will be sent for a given object in this session.

Depending on the target use case, the delivery service that FCAST defines will be more or less reliable. For instance, when used in ON-DEMAND mode over a time period that largely exceeds the typical download time, the service can be considered as fully reliable. When FCAST is used along with a session control application capable of collecting reception information and taking appropriate corrective measures, then the service can be considered as fully reliable. But if FCAST operates in Push mode, for a time period that is close to the typical download time, then the service is usually only partially reliable.

Depending on the target use case, the FCAST scalability will be more or less important. For instance, if used on top of purely unidirectional transport channels, with no feedback information at all, which is the default mode of operation, then the scalability is maximum since neither FCAST, nor ALC/LCT, UDP or IP will generate feedback messages. But if FCAST is used along with a session control application (e.g. an `rdist` like tool), capable of collecting reception information from the receivers, then this session control application (that is kept separate from FCAST) will probably create a limit on the FCAST scalability. This situation can be mitigated by using a hierarchy of feedback message aggregators or servers.

A goal of FCAST is to enable the receivers to collect meta-data information sent in-band, along with the objects. The transmission of meta-data is done in two complementary ways. Depending on the target use case, the sender will use one scheme or the other, or even both of them. When the client must be able to process the meta-data, or a subset of the meta-data, early in the reception process, and in particular before the object has been completely received and decoded, then meta-data (or a subset) is included in a dedicated ALC/LCT header extension. When it is sufficient for the client to extract the meta-data, or a subset of the meta-data, once the object has been fully received and decoded, then meta-data (or a subset) is appended to the object. Several motivations can lead a sender to use one method or the other, or both. Using a dedicated header extension enables a client to select the objects it is interested in, based on various criteria (like the object size or encoding). In that case the client can decide to receive the following packets, or to drop them if he's not interested in, which saves both processing and memory resources. But appending meta-data to the object is also an efficient way to carry the information, in particular with very small objects, for instance when the size of the object and the associated meta-data is less than the maximum payload size. With larger objects, another benefit of appending the meta-data is that this later benefits from the erasure protection provided by the FEC encoding of the object, when FEC is used.

2 FCAST Specifications

This section gives more details on the key design principles behind FCAST, and their motivations.

2.1 Meta-Data Associated to Objects

Meta-data are associated to each object sent during a session. These meta-data can be composed of, but are not limited to:

- the name and location of the object;
- the MIME type of the object;
- the size of the object;
- the (optional) encoding of the object performed by FCAST;
- the size of the object after the (optional) encoding performed by FCAST;
- the timestamp associated to this object;
- the time validity of the object, after which the object is considered as out-of-date;
- the message digest of the object, for instance its MD5 sum, in order to check its integrity;
- a digital signature for this object;
- when a file is split into several objects by FCAST, an indication that this is a split of the original object;
- when a file is split into several objects by FCAST, the slice index in $[0..Slice\_Nb[$, and the total number of slices;
- when a file is split into several objects by FCAST, a counter indicating the offset at which this slice starts within the original object;
- when a file is split into several objects by FCAST, the size of the original file;
- the FEC Object Transmission Information (FEC-OTI) when it is preferable to carry the information as meta-data rather than within the ALC/LCT EXT_FTI header extension;

This list is not limited, and new meta-data information can be added to this list as the need arises. Note also that these items are not all mandatory. The only mandatory meta-data item is the name and location of the object (i.e. "Content-Location" attribute).

2.2 Meta-Data Transmission in ALC Packets

FCAST proposes two complementary but optional ways to carry meta-data:

- *by appending meta-data to the object being transmitted:* this is a very efficient scheme, since meta-data are received along with the object, and benefit from the optional FEC erasure protection of the object. Yet a limit of this scheme is that a client has no information on the object's content before receiving and decoding it completely;
- *by adding a dedicated EXT_OMD (Object Meta-Data) header extension to the ALC/LCT packets for a given object:* this header contains a subset or all of the meta-data of the associated object. Because this header extension can be retrieved and processed before the object is finished, it gives the opportunity to a client to know the object's content and determine whether or not to receive it altogether. There are constraints on the EXT_OMD size (see 3.1): the EXT_OMD size cannot exceed 1020 bytes, and the resulting ALC packet should remain below the PMTU for higher efficiency.

Depending on the target use case, the sender can choose:

- to send the meta-data in the EXT_OMD header extension of all (or most of) the ALC packets sent. In that case, a client will immediately retrieve the meta-data of the object, no matter when he joins the session, and can then decide whether or not to decode the object;
- to send the meta-data in the EXT_OMD header extension of the first few ALC packets sent. In that case, in PUSH mode, a client having joined the session before transmissions actually start, quickly recovers the meta-data of the object, and can decide whether or not to decode the object;

- to send the meta-data in the EXT_OMD header extension of ALC packets chosen periodically, for instance once per second, or every 1000 ALC packets. This strategy enables a receiver to be able to recover the meta-data of an object quickly, for instance on average every 0.5 seconds when the EXT_OMD is sent once per second. The client can then decide whether or not to decode the object, and whether or not to keep the already received packets of the object;

- to send the meta-data only in the object, appended to this later. For instance, it makes sense when sending small objects, since the meta-data size can have a size comparable to that of the object. In that case, it is assumed that receiving the object is not a problem to clients. Sending separately the object and the meta-data would be in that case meaningless, and would only create inter dependencies that can easily be avoided by concatenating object and meta-data.

- to send the meta-data both appended to the object and /in some/in most of/in all/periodically in/ ALC packets with the EXT_OMD header extension. In some cases, the information contained in the EXT_OMD can be a subset of the meta-data, for instance containing the information required by the clients to take the decision of receiving the object or discarding the associated packets, but not the pieces of information that are required for the processing of the object once received. In some cases, the meta-data appended to the object and the meta-data contained in the EXT_OMD header extension of the ALC packets can complement (without creating duplications) one another.

  What information should be put in the EXT_OMD header extension versus should be appended to the object is not specified here and depends on the target use case. In some cases, the client must be able to select the objects he's interested in after receiving some high level indication on the content, e.g. the file's name and encoding, plus a human readable description of the content. In that case, these pieces of information are included in the EXT_OMD. In other target use cases, the client might decide to drop the objects whose size exceeds a given threshold, because of insufficient memory resources for instance. In that case, the EXT_OMD contains the object's transfer length.

2.3 Carousel and Transmissions

Transmissions are performed within a carousel that sends all the objects that have been scheduled for transmission, for a given number of cycles. At a sender, the following operations take place:

1. the user (or another application) selects a set of objects (often files) to deliver and submits them to the FCAST application. Along with each object, the user can specify how many times each object should be sent in this carousel. Said differently, if objects have similar lengths, assigning them a different number of transmissions leads to define different transmission priorities to each of them;

2. the user (or another application) then informs FCAST that all the objects of the set have been submitted. If no new object will be submitted later to FCAST, i.e. if the session's content is now fully defined, the user also informs FCAST;

3. the FCAST application creates the list of all objects that are part of the carousel instance, and explicitly (e.g. as explained in [6]) or implicitly (e.g. deciding to send them in sequence) defines a transmission schedule of the objects and the associated ALC packets.

4. the FCAST application can optionally create a `Carousel Instance` object, that lists the set of objects that are part of the current carousel instance, or that have been removed from the current carousel instance. This `Carousel Instance` object *does not include any meta-data for the objects that are part of the carousel instance*. It provides the TOI of the objects that are in the current carousel instance. Implicitely, all objects that are not in this list are considered as not being part of the carousel instance, even if they were present in the previous carousel instance.

5. the FCAST application starts the carousel, for the number of transmissions specified for each object. Deciding whether to use the EXT_OMD and/or appending meta-data to each object is left to the sender. The FCAST application also decides whether or not to use and send a `Carousel Instance` object to provide the list of objects that are part of the carousel instance. If FCAST knows that no new object will be submitted and if FCAST uses a carousel instance object, then this latter includes the `Complete` keyword (usually an Element's attribute in case of an XML formatting) to inform all clients that no object in addition to the ones specified in this carousel instance will be sent;

6. then transmissions take place until:

- each object has been transmitted the desired number of times, or
- the user (or another application) wants to add one or several new objects to the carousel, or on the opposite wants to remove them from the carousel. In that case a new carousel instance must be created, i.e. we continue at step 1 above.

Using a `Carousel Instance` object can be useful for the clients to know what are the objects that are officially part of the current carousel. Yet it is not mandatory and sessions with a fixed set of objects can omit the `Carousel Instance` object, as well as sessions composed of a small number of objects. In that case, the client will progressively learn what files are part of the carousel by receiving ALC packets with new TOIs. Yet using the `Carousel Instance` object has the benefit of letting the clients know when they can leave the session, i.e. when they have received all the objects that are part of the delivery session, thanks to the `Complete` keyword.

In case of a session with a dynamic set of objects, the sender can easily inform receiver that some objects have been removed from the carousel by using the `CLOSE object` mechanism of ALC/LCT, even if no `Carousel Instance` object is used. Yet this requires that the clients receive at least one of the ALC packets containing the ALC `CLOSE object` flag, and a client with an intermittent connectivity will not necessarily be informed. It is therefore recommended in case of an FCAST session with dynamic content to use `Carousel Instance` object.

The decision of how often and when the `Carousel Instance` object should be sent within an FCAST session is left to the sender and depends on many parameters, including the target use case, and the session dynamicity. For instance, in case of an FCAST session in PUSH mode, then it makes sense to send the `Carousel Instance` object before that objects actually start to be sent (and with a low frequency after the start to enable late receivers to catch up, if feasible). In case of an FCAST session that is highly dynamic, `Carousel Instance` object will probably be sent at the begining of a new carousel instance, and then periodically. The period depends on the desired maximum latency that could be experienced by late receivers who join the FCAST session in the middle of a carousel instance transmission cycle, and who therefore missed the initial `Carousel Instance` object transmission.

3 Control Information Formats

3.1 Object Meta-Data ALC/LCT Header Extension, EXT_OMD

The EXT_OMD header extension format is the following. This is an ALC PI Specific header extension (i.e. $HET \leq 127$).

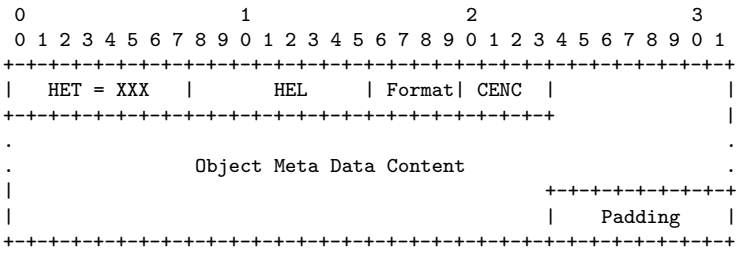

Figure 1: Object Meta-Data (EXT_OMD) header extension format.

In figure 1, the `Format` field defines the format of the `Object Meta Data Content` field (e.g. XML), while the `CENC` field defines the optional content encoding of the `Object Meta Data Content` field (e.g. gzip). Because of the `HEL` semantic, the EXT_OMD size cannot exceed: $255 \times 4 = 1020 bytes$. Of course, the resulting ALC packet should remain below the PMTU for higher efficiency. This constraint can lead the sender to use the EXT_OMD extension in separate control ALC packets, containing no data or repair symbol.

3.2 Object Trailer with Meta-Data

To each object being transmitted, a trailer is appended that might or not include meta-data. The format of the trailer is the following:

In figure 2, the `Format` and `CENC` fields have the same meaning as for EXT_OMD. If no object meta data is appended to the object, then these two fields are not present. The `Trailer Length` field contains the

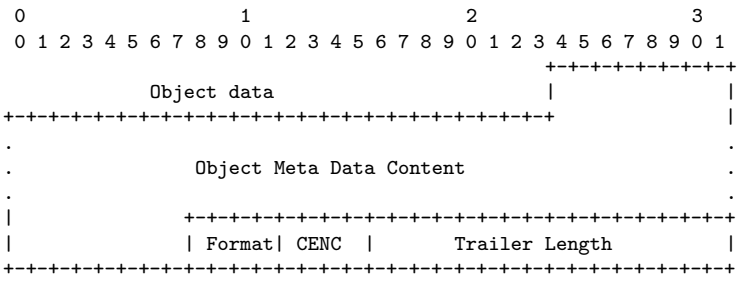

Figure 2: Object trailer with meta-data format.

number of bytes used by all the trailer fields, from the `Object Meta Data Content` field up to and including the `Trailer Length` field. A value of 2 means that the trailer is only composed of the `Trailer Length` field.

3.3 Carousel Instance Object

The `Carousel Instance` object format is the following. It basically consist of two lists:

- a list of TOIs included in the current carousel instance, specified either a the individual TOIs of each object, or as a TOI span (e.g. `10..100` means that all objects whose TOI is in the range 10 to 100 inclusive are part of the carousel instance), or both. In all cases, a TOI in the list is included unless otherwise specified by the exclude list;
- a list of TOIs to exclude from the current carousel instance, even if they are part of the include list. Here also the TOIs are specified either a the individual TOIs to exclude, or as a TOI span to exclude, or both. In all cases, the priority of the exclude list is higher than the priority of the include list: if a TOI is on both list, then the TOI is considered not being part of the carousel instance.

The TOI 0 is reserved for the Carousel Instance object. The various instances of this object are identified by the Carousel Instance ID.

XXX: format TDB

The `Carousel Instance ID` identifies the carousel instance. It starts from 0 and is incremented by 1 for each new carousel instance. Wrapping of the Carousel Instance ID can happen. In that case, IDs that wrapped to 0 must be considered higher than the IDs used before the wrapping.

The `Expires` attribute identifies the validity period of this carousel instance. This validity period is expressed as an NTP time. The validity period should enable the transmission, the reception and processing of all the objects that belong to this carousel instance.

4 Conclusions

This document introduces the FCAST massively scalable, object delivery application on top of unidirectional transport. It has been designed with flexibility in mind in order to enable its specialization to a large variety of target use cases. More information on the project can be found in the Planète-bcast web site [5].

References

[1] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. *Asynchronous Layered Coding (ALC) protocol instantiation*, Dec. 2002. IETF Request for Comments, RFC3450.

[2] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, M. Handley, and J. Crowcroft. *Layered Coding Transport (LCT) building block*, Dec. 2002. IETF Request for Comments, RFC3451.

[3] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. *FLUTE - File Delivery over Unidirectional Transport*, Oct. 2004. IETF RMT Working Group, Request For Comments, RFC 3926.

[4] V. Roca and al. *MCLv3: an Open Source GNU/GPL Implementation of the FLUTE/ALC and NORM Reliable Multicast Protocols*. URL: http://planete-bcast.inrialpes.fr/.

[5] V. Roca and al. *Planète-BCAST: Tools for large scale content distribution*. URL: http://planete-bcast.inrialpes.fr/.

[6] V. Roca and B. Mordelet. Improving the efficiency of a multicast file transfer tool based on alc. Research Report 4411, INRIA, Mar. 2002.