Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 22, 2016

I. Bouazizi
Samsung Research America

March 21, 2016

MPEG Media Transport Protocol (MMTP)
draft-bouazizi-tsvwg-mmtp-01

Abstract

The MPEG Media Transport Protocol (MMTP) is a transport protocol that is designed to support download, progressive download, and streaming applications simultaneously. MMTP provides a generic media streaming mode by optimizing the delivery of generic media data encapsulated according to the ISO-Base Media File Format (ISOBMFF). In the file delivery mode, MMTP supports the delivery of any type of file. MMTP may used in IP unicast or multicast delivery and supports both Forward Error Correction (FEC) and retransmissions for reliable delivery of content.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Table of Contents

1. Introduction

The MMT protocol is an application layer transport protocol that is designed to efficiently and reliably transport multimedia data. MMTP can be used for both timed and non-timed media data. It supports

several features, such as media multiplexing and network jitter estimation. These features are designed to deliver content composed of various types of encoded media data more efficiently. MMTP may run on top of existing network protocols such as UDP and IP. In this specification, the carriage of data formatted differently than the MMTP payload format as specified in Section 4 by MMTP is not defined. The MMT protocol is designed to support a wide variety of applications and does not specify congestion control. The congestion control function is left for the application implementation. MMTP supports the multiplexing of different media data such as ISOBMFF files from various Assets over a single MMTP packet flow. It delivers multiple types of data in the order of consumption to the receiving entity to help synchronization between different types of media data without introducing a large delay or requiring large buffer. MMTP defines two packetization modes, Generic File Delivery mode as specified in Section 4.2 and the ISOBMFF mode as specified in Section 4.1. The former defines a mode for packetizing media data based on the size of the payload to be carried and the latter defines a mode for packetizing media data based on the type of data to be carried in the payload. MMTP supports simultaneous transmission of packets using the two different modes in a single delivery session. MMTP also provides means to calculate and remove jitter introduced by the underlying delivery network, so that constant end-to-end delay for data delivery can be achieved. By using the delivery timestamp field in the packet header, jitter can be precisely estimated without requiring any additional signalling information and protocols.

2. Rationale

MMTP provides a generic media transport protocol that inherently supports virtually any media type and codec. For this purpose, MMTP is designed to support a limited set of payload types agnostic to the media type or coding format, but providing generic information to serve the needs of different multimedia delivery services. The MMTP payload format is defined as a generic payload format for the packetization of media data. It is agnostic to media codecs used for encoded media data, so that any type of media data that are encapsulated in the ISOBMFF format can be packetized into MMTP payloads. The MMTP payload format also supports fragmentation and aggregation of data to be delivered. MMTP supports both streaming and download modes, where the streaming mode is optimized for packetized streaming of ISO-Base Media File formatted files (ISOBMFF mode) and the download mode allows for flexible delivery of generic files (GFD mode). In addition, MMTP delivers streaming support data such as Application Layer Forward Error Correction (AL-FEC) repair data.

2.1. Difference to RTP

The RTP protocol was initially designed to support multi-party real-timed communication conferencing over the Internet. Key concern at that time was scalability of RTP to a large number of participants and dealing with media synchronization. Consequently, the RTP protocol is a mixture of transport and presentation layer functions. RTP supports a wide range of media types and codecs through the definition of codec-specific payload formats.

A set of issues arise when deploying RTP for media delivery, some of which are provided in the following list:

Lack of Multiplexing: RTP usually requires two separate ports for every media session. Rich media services have several service components, each of which would require an RTP/RTCP port pair. Although some level of multiplexing is possible in RTP (i.e. RTP and RTCP multiplexing as defined in [RFC5761], it is not clear that all RTP implementations support it and this still does not solve the problem. This is one of the reasons the industry is moving towards HTTP-based streaming where a single port is used. MMTP uses a single port and multiplexes all media streams of a service as well as the related signaling and any non-real time objects into a single MMTP flow that is self-contained and self-describing.

Costly Server Maintenance One of the major issues with RTP is the costly operation of dedicated streaming servers that need to be updated to support any new media codecs. The server must be upgraded to support the new payload format for any new media codec that the service provider wishes to use. MMTP solves this issue by supporting a single payload format for media streaming based on the ISOBMFF file format. Any media codec that can be encapsulated into the ISOBMFF file format can be streamed without any modifications by an MMT server.

Coupling Delivery and Presentation RTP carries the presentation timestamp of the encapsulated media data, which corresponds to the sampling instant of the first media sample/sub-sample contained in the packet payload. As the delivery timestamp is not provided in RTP, it is often assumed that the presentation timestamp is equal to the delivery timestamp. This coupling may make sense for real-time conferencing use cases but is generally not useful for streaming of on-demand content as the receiver will not be aware of the exact delivery time and will usually use external media for controlling the presentation time (so that the RTP timestamp will only be use for intra-media synchronization). MMTP decouples media delivery and media presentation completely by carrying only

the delivery timestamp at the MMTP protocol level. The presentation time is controlled by external Presentation Information that may as well be carried as part of the MMTP flow, whereas the intra-synchronization is provided by the ISOBMFF file format. The delivery timestamp may be used for de-jittering, retransmissions, and other purposes.

3. Packet Header Field

The MMTP header is of variable size, where the size of the header may be deduced from the header flags. In the MMTP header, all integer fields are carried in "big-endian" or "network order" format, so that the most significant byte is first. Bits marked as "reserved" (r) MUST be set to 0 by the sender and ignored by receivers in this version of the specification. Unless otherwise noted, numeric constants in this specification are in decimal form (base 10). The format of the MMTP header is depicted in Figure 1.

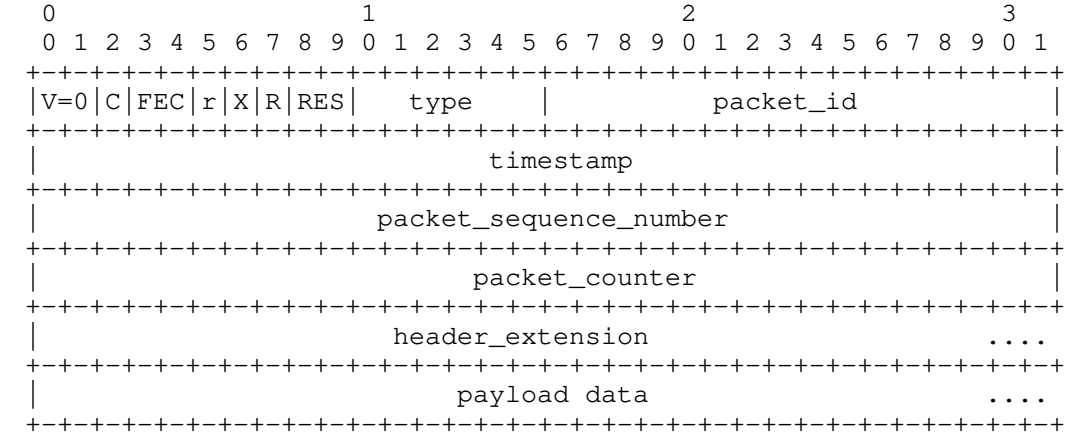


Figure 1: MMTP Header

The function and length of each field in the MMTP header is specified as follows:

version (V): 2 bits

indicates the version number of the MMTP protocol. This field shall be set to "00" to comply with this specification.

packet_counter_flag (C): 1 bit

"1" in this field indicates that the packet_counter field is present.

FEC_type (FEC): 2 bits

indicates whether the payload carries FEC source data or repair data. Valid values of this field are listed in Table 1 below. Depending on the FEC scheme, additional payload header may be added, for instance to identify the contained symbol(s).

reserved (r): 1 bit

reserved for future use.

extension_flag (X): 1 bit

when set to "1" this flag indicates that the header_extension field is present.

RAP_flag (R): 1 bit

when set to "1" this flag indicates that the payload contains a Random Access Point (RAP) to the data stream of that data type. The exact semantics of this flag are defined by the data type itself. The RAP_flag shall be set to mark data units of Fragment Type value "0" and "1" and for data units that contain a sync sample or a fragment thereof in the case of timed media and for the primary item of non-timed data.

reserved (RES): 2 bits

reserved for future use.

type: 6 bits

this field indicates the type of payload data. Payload type values are defined in Table 2.

packet_id: 16 bits

this field is an integer value that can be used to identify related media data, for example media that belong to the same media asset. The packet_id is unique throughout the lifetime of the delivery session and for all MMTP flows delivered by the same MMTP sender.

packet_sequence_number: 32 bits

an integer value that is used to distinguish between packets that have the same packet_id. The value of this field should start from an arbitrary value and shall be incremented by one for each

new MMTP packet. It wraps around to "0" after the maximum value is reached.

timestamp: 32 bits

specifies the time instance of MMTP packet delivery based on UTC. The format is the "short-format" as defined in clause 6 of [RFC5905], NTP version 4. This timestamp specifies the sending time at the first byte of MMTP packet. It is required that an MMTP sender should provide accurate time information that are synchronized with UTC.

packet_counter: 32 bits

an integer value for counting MMTP packets. It is incremented by 1 when an MMTP packet is sent regardless of its packet_id value. This field starts from arbitrary value and wraps around to "0" after its maximum value is reached.

header_extension:

this field contains user-defined information. The header extension mechanism is provided to allow for proprietary extensions to the payload format to enable applications and media types that require additional information to be carried in the payload format header. The header extension mechanism is designed in a such way that it may be discarded without impacting the correct processing of the MMTP payload. The header extension shall have the format as shown in Figure 2. This specification does not specify any particular header extension.

Value	Description
0	MMTP packet without AL-FEC protection
1	MMTP packet with AL-FEC protection (FEC source packet)
2	MMTP packet for repair symbol(s) (FEC repair packet)
3	Reserved for future use

Table 1: FEC Type

Value	Data type	Definition of data unit
0x00	ISOBMFF file	The packet carries a media-aware fragment of the ISOBMFF file
0x01	Generic object	The packet contains a generic object such as a complete ISOBMFF file or an object of another type or a chunk thereof.
0x02	signalling message	one or more signalling messages or a fragment of a signalling message. The syntax and semantics of signalling messages are out of scope of the current memo.
0x03	repair symbol	The packet carries a single complete FEC repair symbol
0x04-0x1F	reserved	reserved for ISO use
0x20-0x3F	reserved	reserved for private use

Table 2: Data type and definition of data unit

3.1. MMTP Header Extension

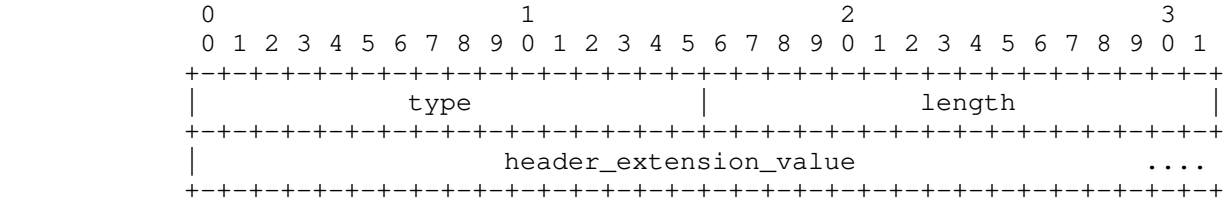


Figure 2: MMTP Header Extension

The function and length of each field in the MMTP header extension is as follows:

type: 16 bits

indicates the unique identification of the following header extension.

length: 16 bits

indicates the length of header_extension_value field in byte.

header_extension_value

provides the extension information. The format of this field is out of scope of this specification.

4. The MMTP payload

The MMTP payload is a generic payload format to packetize and carry media data such as ISOBMFF files, generic objects, and other information for consumption of a media service using the MMT protocol. The appropriate MMTP payload format shall be used to packetize ISOBMFF files, and generic objects. An MMTP payload may carry complete ISOBMFF files or fragments of ISOBMFF files, signalling messages, generic objects, repair symbols of AL-FEC schemes, etc. The type of the payload is indicated by the type field in the MMT protocol packet header. For each payload type, a single data unit for delivery as well as a type specific payload header are defined. For example, fragment of an ISOBMFF file (e.g. a data unit) is considered as a single data unit when MMTP payload carries ISOBMFF file fragments. The MMT protocol may aggregate multiple data units with the same data type into a single MMTP payload. It can also fragment a single data unit into multiple MMTP packets. The MMTP payload consists of a payload header and payload data. Some data types may allow for fragmentation and aggregation, in which case a single data unit is split into multiple fragments or a set of data units are delivered in a single MMTP packet. Each data unit may have its own data unit header depending on the type of the payload. For types that do not require a payload type specific header no payload type header is present and the payload data follows the MMTP header immediately. Some fields of the MMTP packet header are interpreted differently depending on the payload type. The semantics of these fields will be defined by the payload type in use.

4.1. The ISOBMFF Mode

The delivery of ISOBMFF files to MMT receivers using the MMT protocol requires a packetization and depacketization procedure to take place at the MMTP sender and MMTP receiver, respectively. The packetization procedure transforms an ISOBMFF file into a set of MMTP payloads that are then carried in MMTP packets. The MMTP payload format allows for fragmentation of the MMTP payload to enable the delivery of large payloads. It also allows for the aggregation of multiple MMTP payload data units into a single MMTP payload, to cater for smaller data units. At the receiving entity depacketization is performed to recover the original ISOBMFF file data. Several depacketization modes are defined to address the different requirements of the overlaying applications. It the payload type is 0x00, the ISOBMFF file is fragmented in a media aware way allowing the transport layer to identify the nature and priority of the fragment that is carried. A fragment of an ISOBMFF file may either

be ISOBMFF file metadata, a Movie Fragment metadata, a data unit, or a non-timed media data item.

4.1.1. MMTP payload header for ISOBMFF mode

The payload type specific header is provided in Figure 3.

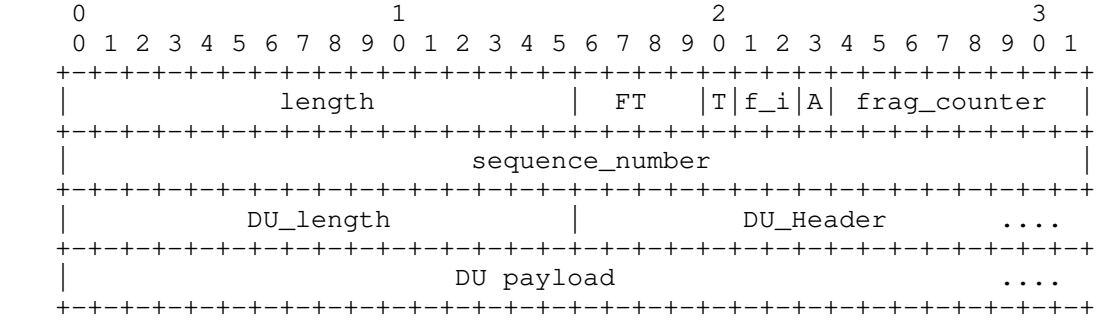


Figure 3: Structure of the MMTP payload header for the ISOBMFF mode

For payload that carries a data unit, the DU header is specified depending on the value of the T flag indicating wether the carried data is timed or non-timed media. For timed media (i.e. when the value of T is set to "1") the DU header fields are shown in Figure 4. For non-timed media (T is set to "0"), the DU header is defined as shown in Figure 4.

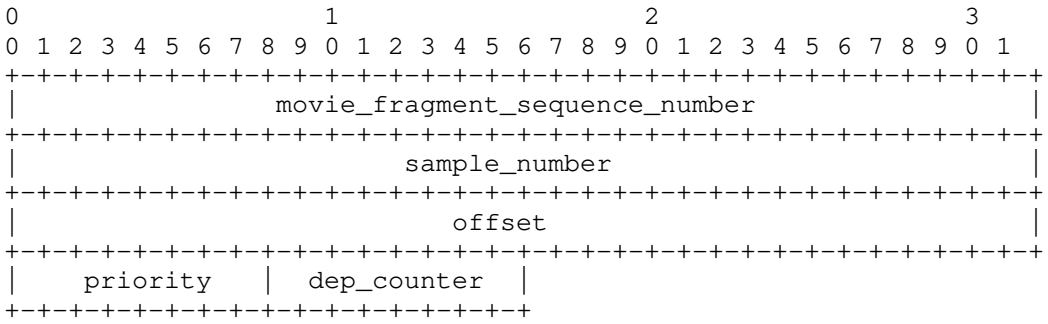


Figure 4: The DU header for a timed-media data unit

For non-timed media, the DU header fields are shown in Figure 5.

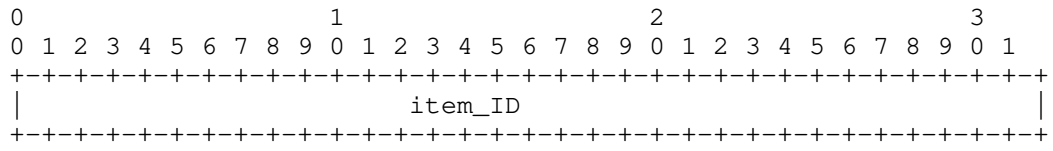


Figure 5: The DU header for a non-timed media data unit

length: 16 bits indicates the length of payload excluding this field in byte.

Fragment Type (FT): 4 bits this field indicates the fragment type and its valid values are shown in Table 3.

Timed Flag (T): 1 bit this flag indicates if the fragment is from an ISOBMFF file that carries timed (value 1) or non-timed media (value 0).

Fragmentation Indicator (f_i) : 2 bits this field indicates the fragmentation indicator contains information about fragmentation of data unit in the payload. The four values are listed in Table 4. If the value is set to "00", the aggregation_flag can be presented.

FT	Description	Content
0	ISOBMFF metadata	contains the ftyp, mmpu, moov, and meta boxes as well as any other boxes that appear in between.
1	Movie fragment metadata	contains the moof box and the mdat box, excluding all media data inside the mdat box.
2	a data unit	contains a sample or sub-sample of timed media data or an item of non-timed media data.
3~15	Reserved for private use	reserved

Table 3: Data type and definition of data unit

fragmentation indicator	Description
00	Payload contains one or more complete data units.
01	Payload contains the first fragment of data unit
10	Payload contains a fragment of data unit that is neither the first nor the last part.
11	Payload contains the last fragment of data unit.

Table 4: Values for fragmentation indicator

The following flags are used to indicate the presence of various information carried in the MMTP payload. Multiple bits can be set simultaneously.

aggregation_flag (A: 1 bit)

when set to "1" indicates that more than 1 data unit is present in the payload, i.e. multiple data units are aggregated.

fragment_counter (frag_count: 8 bits)

this field specifies the number of payload containing fragments of same data unit succeeding this MMTP payload. This field shall be "0" if aggregation_flag is set to "1".

sequence_number (32 bits)

the sequence number of the ISOBMFF to which this fragment belongs.

DU_length (16 bits)

this field indicates the length of the data following this field. When aggregation_flag is set to "0", this field shall not be present. When aggregation_flag is set to "1", this field shall appear as many times as the number of the data units aggregated in the payload and preceding each aggregated data unit.

DU_Header

The header of the data unit, which depends on the FT field. A header is only defined for the media unit fragment type, with different semantics for timed and non-timed media as identified by the T flag.

movie_fragment_sequence_number (32 bits)

the sequence number of the movie fragment to which the media data of this data unit belongs. (see [isopart12] sub-clause 8.5.5)

sample_number (32 bits)

the sample number of the sample to which the media data of the data unit. (see [isopart12] sub-clause 8.8.8)

offset (32 bits)

offset of the media data of this data unit inside the referenced sample.

subsample_priority (priority: 8 bits)

provides the priority of the media data carried by this data unit compared to other media data of the same ISOBMFF file. The value of subsample_priority shall be between "0" and "255", with higher values indicating higher priority.

dependency_counter (dep_counter: 8 bits)

indicates the number of data units that depend on their media processing upon the media data in this data unit.

Item_ID (32 bits)

the identifier of the item that is carried as part of this data unit.

For the FT types "0" and "1", no additional DU header is defined.

4.2. Generic File Delivery Mode

MMTP also supports the transport of generic files and Assets and uses payload type "0x01" as defined in Table 3. An Asset consists of one or more files that are logically grouped and share some commonality for an application, e.g. Segments of a Dynamic Adaptive Streaming over HTTP (DASH) Representation, a sequence of ISOBMFF files, etc. In the generic file delivery (GFD) mode, an Asset is transported by using MMTP"s GFD payload type. Each file delivered using the GFD mode requires association of transport delivery information. This includes, but is not limited to information such as the transfer length. Each file delivered using the GFD mode may also have associated content specific parameters such as Name, Identification, and Location of file, media type, size of the file, encoding of the

file or message digest of the file. In alignment with HTTP/1.1 protocol as defined in [RFC2616], each file within one generic Asset may have assigned any meta-information about the entity body, i.e. the delivered file. The details are also defined in Section 4.2.1.

4.2.1. GFD Information

In the GFD mode, each file gets assigned the following parameters:

- the asset to which each object belongs to. Objects that belong to the same asset are considered as logically connected, e.g. all DASH segments of a Representation and also across Representations that extend over multiple DASH Periods and which carry pieces of the same content.
- Each object is associated with a unique identifier within the scope of the packet_id.
- each object is associated with a CodePoint. A CodePoint associates a specific object and object transport properties. Packets with the same TOI shall have the same CodePoint value. For more details see 0.

4.2.1.1. GFD Table

The GFD table provides a list of CodePoints as defined in Section 4.2.1.2. Each CodePoint gets dynamically assigned a CodePoint value. Table 5 shows the structure and semantics of the GFD table.

Element or Attribute Name	Use	Description
GFDTable CodePoint	1..N	The element carries a GFDTable defines all CodePoints in the MMTP session

Table 5: GFD Table

Legend: For attributes: M=Mandatory, O=Optional, OD=Optional with Default Value, CM=Conditionally Mandatory. For elements: minOccurs..maxOccurs (N=unbounded) Elements are bold; attributes are non-bold and preceded with an @

4.2.1.2. CodePoints

A CodePoint value can be used to obtain following information:

- the maximum transfer length of any object delivered with this CodePoint signalling

In addition, a CodePoint may include following information

- the actual transfer length of the objects
- any information that may be present in the entity-header as defined in [RFC2616] section 7.1.
- A Content-Location-Template as defined in Section 4.2.1.3 using the TOI and packet_id parameter, if present. The TOI and packet_id may be used to generate the Content-Location for each TOI and packet_id. If such a template is present, the processing in Section 4.2.1.3 shall be used to generate the Content-Location and the value of the URI shall be treated as the Content-Location field in the entity-header.
- Specific information on the content, for example how the content is packaged, etc.

Within one session, at most 256 CodePoints may be defined. The definition of CodePoints is dynamically setup in the MMTP Session Description. The CodePoint semantics are described in Table 6.

Element or Attribute Name	Use	Description
@value	M	defines the value of the CodePoint in the MMTP session as provided in the CodePoint value of the MMTP packet header containing the GFD payload. The value shall be between 1 and 255. The value 0 is reserved.
@fileDeliveryMode	M	specifies the file delivery mode according to Section 4.2.
@maximumTransferLength	M	specifies the maximum transfer length in bytes of any object delivered with this CodePoint in this MMTP session.
@constantTransferLength	OD default: 'false'	specifies if all objects delivered by this CodePoint have constant transfer length. If this attribute is set to TRUE, all objects shall have transfer length as specified in the @maximumTransferLength attribute.
@contentLocationTemplate	O	specifies a template to generate the Content-Location of the entity header.
EntityHeader	0..1	specifies a full entity header in the format as defined in [RFC2616] section 7.1. The entity header applies for all objects that are delivered with the value of this CodePoint.

Table 6: CodePoint Semantics

Legend: For attributes: M=Mandatory, O=Optional, OD=Optional with Default Value, CM=Conditionally Mandatory. For elements:

minOccurs..maxOccurs (N=unbounded) Elements are bold; attributes are non-bold and preceded with an @

4.2.1.3. Content-Location Template

A CodePoint may include a @contentLocationTemplate attribute. The value of @contentLocationTemplate attribute may contain one or more of the identifiers listed in Table 7. In each URL, the identifiers from Table 7 shall be replaced by the substitution parameter defined in Table 7. Identifier matching is case-sensitive. If the URL contains unescaped $ symbols which do not enclose a valid identifier then the result of URL formation is undefined. The format of the identifier is also specified in Table 7. Each identifier may be suffixed, within the enclosing "$" characters following this prototype: %0[width]d The width parameter is an unsigned integer that provides the minimum number of characters to be printed. If the value to be printed is shorter than this number, the result shall be padded with zeros. The value is not truncated even if the result is larger. The @contentLocationTemplate shall be authored such that the application of the substitution process results in valid URIs. Strings outside identifiers shall only contain characters that are permitted within URLs according to [RFC3986].

$Identifier$	Substitution parameter	Format
$$	Is an escape sequence, i.e. "$$" is replaced with a single "$"	not applicable
$PacketID$	This identifier is substituted with the value of the packet_id of the associated MMT flow.	The format tag may be present.If no format tag is present, a default format tag with width=1 shall be used.
TOI	This identifier is substituted with the Object Identifier of the corresponding MMTP packet containing the GFDpayload.	The format tag may be present. If no format tag is present, a default format tag with width=1 shall be used.

Table 7: Identifiers for URL templates

4.2.1.4. File metadata

Files can be transported using the GFD mode of the MMT protocol. Furthermore, the GFD mode can also be used to transport entities where an entity is defined according to section 7 of [RFC2616]. An entity consists of meta-information in the form of entity-header fields and content in the form of an entity-body (the file), as described in section 7 of [RFC2616]. This enables that files may get assigned information by inband delivery in a dynamic fashion. For example, it enables the association of a Content-Location, the Content-Size, etc. The file delivery mode shall be signaled in the CodePoint.

Value $Identifier$	Description	Definition
1	The transport object is a file	in Section 4.2.1.4.1
2	The delivered object is an entity consisting of an entity-header and the file	in Section 4.2.1.4.2

Table 8: File Delivery Modes for GFD

4.2.1.4.1. Regular File

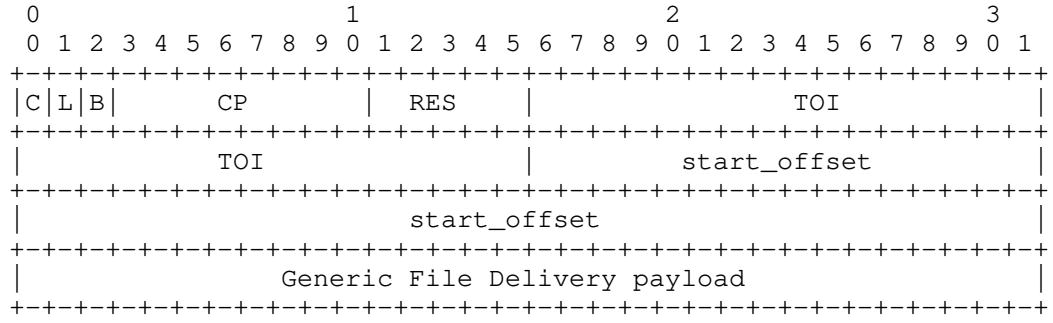
In case of the regular file, the object represents a file. If the CodePoint defined in the GFD table contains entity-header fields or entity-header fields can be generated, then all of these entity-header fields shall apply to the delivered file.

4.2.1.4.2. Regular Entity

In case of the regular entity, the object represents an entity as defined in section 7 of [RFC2616]. An entity consists of entity-header fields and an entity-body. If the CodePoint defined in the GFD table contains entity-header fields or entity-header fields can be generated, then all of these entity-header fields apply to the delivered file. If the entity-header field is present in both locations, then the entity header field in the entity-header delivered with the object overwrites the one in the CodePoint.

4.2.1.5. MMTP payload header for GFD mode

The GFD mode of MMTP delivers regular files. When delivering regular files, the object represents a file. If the CodePoint defined in the MMTP Session description contains entity-header fields or entity-header fields can be generated, then all of these entity-header fields shall apply to the delivered file. The payload packets sent using MMTP shall include a GFD payload header and a GFD payload as shown in Figure 6. In some special cases a MMTP sender may need to produce packets that do not contain any payload. This may be required, for example, to signal the end of a session.



MMTP payload header for GFD mode

Figure 6

The GFD payload header as shown in Figure 6 and has a variable size. Bits designated as "padding" or "reserved" (r) MUST by set to 0 by MMTP sender s and ignored by receivers. Unless otherwise noted, numeric constants in this specification are in decimal form

C (1 bit)

indicates that this is the last packet for this session.

L (1 bit)

indicates that this is the last delivered packet for this object.

B (1 bit)

indicates that this packet contains the last byte of the object.

CodePoint (CP: 8 bits)

An opaque identifier that is passed to the packet payload decoder to convey information on the packet payload. The mapping between the CodePoint and the actual codec is defined on a per session basis and communicated out-of-band as part of the session description information.

RES (5 bits)

a reserved field that should be set to "0".

Transport Object Identifier (TOI: 32 bits)

The object identifier should be set to a unique identifier of the generic object that is being delivered. The mapping between the object identifier and the object information (such as URL and MIME type) may be done explicitly or implicitly. For example a sequence of DASH segments may use the segment index as the object identifier and a numerical representation identifier as the packet_id. This mapping may also be performed using a signalling message

start_offset (48 bits)

the location of the current payload data in the object.

5. Protocol Operation

In this section, we describe the behavior of an MMTP receiver and of an MMTP sender when operating the MMTP protocol using different payload types.

5.1. General Operation

An MMTP session consists of one MMTP transport flow. MMTP transport flow is defined as all packet flows that are delivered to the same destination and which may originate from multiple MMTP senders. In the case of IP, destination is the IP address and port number. A single Package may be delivered over one or multiple MMTP transport flows. A single MMTP transport flow may deliver data from multiple Packages. An MMTP transport flow may carry multiple Assets. Each Asset is associated with a unique packet_id within the scope of the MMTP session. MMTP provides a streaming-optimized mode (the ISOBMFF mode) and a file download mode (the GFD mode). The Asset is delivered as a set of related objects denoted as an object flow. Object may either be an ISOBMFF file, file or signalling message. Each object flow shall either be carried in ISOBMFF mode or GFD mode, however, the delivery of one Package may be performed using a mix of the 2(two) modes, i.e. some Assets may be delivered using the ISOBMFF

mode and others using the GFD mode. The MMTP packet sub-flow is the subset of the packets of an MMTP packet flow that share the same packet_id. The object flow is transported as an MMTP packet sub-flow. The ISOBMFF mode supports the packetized streaming of an ISOBMFF file. The GFD mode supports flexible file delivery of any type of file or sequence of files. MMTP is suitable for unicast as well as multicast media distribution. To ensure scalability in multicast/ broadcast environments, MMTP relies mainly on FEC instead of retransmissions for coping with packet error. Before joining the MMTP session, the MMTP receiver should obtain sufficient information to enable reception of the delivered data. This minimum required information is specified in Section 6. MMTP requires MMTP receivers to be able to uniquely identify and de-multiplex MMTP packets that belong to a specific object flow. In addition, MMT receivers are required to be able to identify packets carrying AL-FEC repair packets by interpreting the type field of the MMTP packet header. The MMTP receiver shall be able to simultaneously receive, de-multiplex, and reconstruct the data delivered by MMTP packets of different types and from different object flows. A single MMTP packet shall carry exactly one MMTP payload.

5.2. Delivery ISOBMFF objects

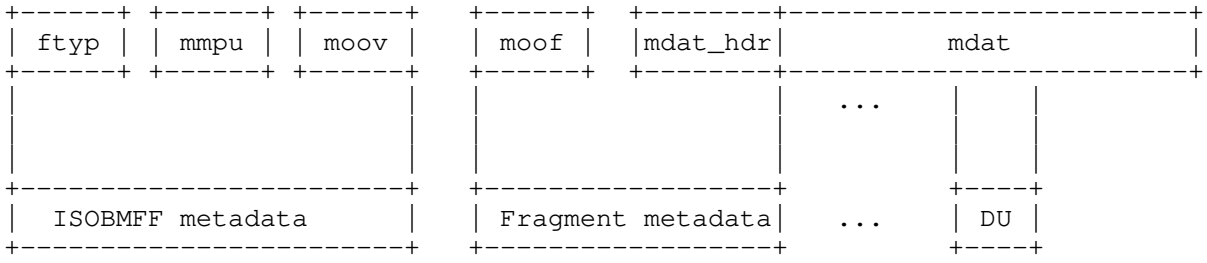
The ISOBMFF mode is used to transport ISOBMFF files sent by a sending entity to a receiving entity.

5.2.1. MMTP sender operation

5.2.1.1. Timed Media Data

The packetization of an ISOBMFF file that contains timed media may be performed in a ISOBMFF file format aware mode or ISOBMFF file format agnostic mode. In the media format agnostic mode, the ISOBMFF file is packetized into data units of equal size (except for the last data unit, of which the size may differ) or predefined size according to the size of MTU of the underlying delivery network by using GFD mode as specified in Section 4.2. It means that the packetization of the ISOBMFF file format agnostic mode only consider the size of data to be carried in the packet. The type field of MMTP packet header specified in Section 4.1 is set to "0x00" to indicate that the packetization is format agnostic mode. In the format agnostic mode the packetization procedure takes into account the boundaries of different types of data in ISOBMFF file to generate packets by using ISOBMFF mode as specified in Section 4.1. The resulting packets shall carry delivery data units of either ISOBMFF file metadata, movie fragment metadata, or a data unit. The resulting packets shall not carry more than two different types of delivery data units. The delivery data unit of ISOBMFF file metadata consists of the "ftyp"

box, the "mmpu" box, the "moov" box, and any other boxes that are applied to the whole ISOBMFF file. The FT field of the MMTP payload carrying a delivery data unit of the ISOBMFF file metadata is set to "0x00". The delivery data unit of movie fragment metadata consists of the "moof" box and the "mdat" box header (excluding any media data). The FT field of the MMTP payload carrying a delivery data unit of movie fragment metadata is set to "0x01". The media data, data units in "mdat" box of the ISOBMFF file, is then split into multiple delivery data units in a format aware way. This may for example be performed with the help of the MMT hint track. The FT field of the MMTP payload carrying a delivery data unit is set to "0x02". Each data unit is prepended with a data unit header, which has the syntax and semantics as defined in section Section 4.1.1. It is followed by the media data of the data unit. This procedure is described by Figure 7.



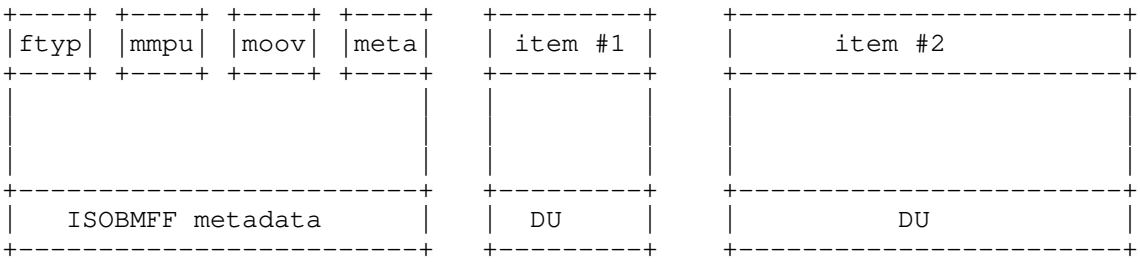
Payload generation for timed media

Figure 7

5.2.1.2. Non-Timed Media Data

The packetization of non-timed media data may also be performed in two different modes. In the ISOBMFF file format agnostic mode, the ISOBMFF file is packetized into delivery data units of equal size (except for the last data unit, of which the size may differ) or or predefined size according to the size of MTU of the underlying delivery network by using GFD mode as specified in Section 4.2. The type field of MMTP packet header specified in Figure 1 is set to "0x00" to indicate that the packetization is format agnostic mode. In the format agnostic mode, the ISOBMFF file shall be packetized into the packet containing delivery data units of either ISOBMFF file metadata or data unit by using ISOBMFF mode as defined in Section 4.1. The delivery data unit of the ISOBMFF file metadata contains the "ftyp" box, the "moov" box, and the "meta" box and any other boxes that are applied to the whole ISOBMFF file. The FT field of the MMTP payload carrying a delivery data unit of the ISOBMFF file metadata is set to "0x01". Each delivery data unit contains a single

item of the non-timed media. The FT field of the MMTP payload carrying a delivery data unit is set to "0x02". Each item of the non-timed data is then used to build a data unit. Each data unit consists of a data unit header and the item's data. The data unit header is defined in Section 4.1.1.



Payload generation for non-timed media

Figure 8

5.2.2. MMTP receiver operation

The depacketization procedure is performed at an MMTP receiver to rebuild the transmitted ISOBMFF file. The depacketization procedure may operate in one of the following modes, depending on the application needs:

ISOBMFF mode:

in the ISOBMFF mode, the depacketizer reconstructs the full ISOBMFF file before forwarding it to the application. This mode is appropriate for non-time critical delivery, i.e. the ISOBMFF file's presentation time as indicated by the presentation information document is sufficiently behind its delivery time.

Fragment mode:

in the Fragment mode, the depacketizer reconstructs a complete fragment including the fragment metadata and the "mdat" box with media samples before forwarding it to the application. This mode does not apply to non-timed media. This mode is suitable for delay-sensitive applications where the delivery time budget is limited but is large enough to recover a complete fragment.

Media unit mode:

in the media unit mode, the depacketizer extracts and forwards media units as fast as possible to the application. This mode is

applicable for very low delay media applications. In this mode, the recovery of the ISOBMFF file is not required. The processing of the fragment media data is not required but may be performed to resynchronize. This mode tolerates out of order delivery of the fragment metadata data units, which may be generated after the media units are generated. This mode applies to both timed and non-timed media. Using the data unit sequence numbers, it is relatively easy for the receiver to detect missing packets and apply any error correction procedures such as ARQ to recover the missing packets. The payload type may be used by the MMTP sender to determine the importance of the payload for the application and to apply appropriate error resilience measures.

5.3. Delivering Generic Objects

The files delivered using the GFD mode may have to be provided to an application, for example Presentation Information documents or a Media Presentation Description as defined in ISO/IEC 23009-1 may refer to the files delivered using MMTP as GFD objects. The file shall be referenced through the URI provided or derived from Content-Location, either provided in-band as part of an entity header or as part of a GFDT. In certain cases, the files have an availability start time in the application. In this case the MMTP session shall deliver the files such that the last packet of the object is delivered such that it is available latest at the receiver at the availability start time as announced in the application. Applications delivered through the GFD mode may impose additional and stricter requirements on the sending of the files within a MMTP session.

5.3.1. MMTP sender operation

If more than one object is to be delivered using the GFD mode, then the MMTP sender shall use different TOI fields. In this case each object shall be identified by a unique TOI scoped by the packet_id, and the MMTP sender shall use that TOI value for all packets pertaining to the same object. The mapping between TOIs and files carried in a session is either provided in-band or in a GFDT. The GFD payload header as defined in Section 4.2.1.5 shall be used. The GFD payload header contains a CodePoint field that shall be used to communicate to a MMTP receiver the settings for information that is established for the current MMTP session and may even vary during a MMTP session. The mapping between settings and Codepoint values is communicated in the a GFDT as described in Section 4.2.1.1. Let T > 0 be the Transfer-Length of any object in bytes. The data carried in the payload of a packet consists of a consecutive portion of the object. Then for any arbitrary X and any arbitrary Y > 0 as long as

X + Y is at most T, an MMTP packet may be generated. In this case the followings shall hold:

1. The data carried in the payload of a packet shall consist of a consecutive portion of the object starting from the beginning of byte X through the beginning of byte X + Y.

2. The start_offset field in the GFD payload header shall be set to X and the payload data shall be added into the packet to send.

3. If X + Y is identical to T,

    * the payload header flag B shall be set to "1".

    * else

    * the payload header flag B shall be set to "0".

The following procedure is recommended for a MMTP sender to deliver an object to generate packets containing start_offset and corresponding payload data.

1. Set the byte offset counter X to "0"

2. For the next packets to be delivered set the length in bytes of a payload to a value Y, which is

    * reasonable for a packet payload (e.g., ensure that the total packet size does not exceed the MTU), and

    * such that the sum of X and Y is at most T, and

    * such that it is suitable for the payload data included in the packet

3. Generate a packet according to the rules a to c from above

4. If X + Y is equal to T,

    * set the payload header flag B to "1"

    * else

    * set the payload header flag B to "0"

    * increment X = X + Y

    * goto 2

The order of packet delivery is arbitrary, but in the absence of other constraints delivery with increasing start_offset number is recommended. Note that the transfer length may be unknown prior to sending earlier pieces of the data. In this case, T may be determined later. However, this does not affect the sending process above. Additional packets may be sent following the rules in 1 to 3 from above. In this case the B flag shall only be set for the payload that contains the last portion of the object.

5.3.2. GFD Payload

The bytes of the object are referenced such that byte 0 is the beginning of the object and byte T-1 is the last byte of the object with T is the transfer length (in bytes) of the object. The data carried in the payload of an MMTP packet shall consist of a consecutive portion of the object starting from the beginning of byte X and ending at the beginning of byte X + Y where

1. X is the value of start_offset field in the GFD payload header

2. Y is the length of the payload in bytes

Note that Y is not carried in the packet, but framing shall be provided by the underlying transport protocol.

5.3.3. GFD Table Delivery

When GFD mode is used, the GFD table (GFDT) shall be provided. A file that is delivered using the GFD mode, but not described in the GFD table is not considered a 'file' belonging to the MMTP session. Any object received with an unmapped CodePoint should be ignored by a MMTP receiver. Other ways of delivery the GFD table may possible, but out of scope of this specification.

5.3.4. MMTP receiver operation

The GFDT may contain one or multiple CodePoints identified by different CodePoint values. Upon receipt of each GFD payload, the receiver proceeds with the following steps in the order listed.

1. The MMTP receiver shall parse the GFD payload header and verify that it is a valid header. If it is not valid, then the GFD payload shall be discarded without further processing.

2. The MMTP receiver shall parse the CodePoint value and verify that the GFDT contains a matching CodePoint. If it is not valid, then the GFD payload shall be discarded without further processing.

3. The MMTP receiver should process the remainder of the payload, including interpreting the other payload header fields appropriately, and using the source_offset and the payload data to reconstruct the corresponding object as follows:

    1. The MMT receiving can determine from which object a received GFD payload was generated by using the GFDT., and by the TOI carried in the payload header.

    2. Upon receipt of the first GFD payload for an object, the MMTP receiver uses the Maximum Transfer Length received as part of the GFDT to determine the maximum length T' of the object.

    3. The MMTP receiver allocates space for the T' bytes that the object may require.

    4. The MMTP receiver also computes the length of the payload, Y, by subtracting the payload header length from the total length of the received payload.

    5. The MMTP receiver allocates a Boolean array RECEIVED[0..T'-1] with all T entries initialized to false to track received object symbols. The MMTP receiver keeps receiving payloads for the object block as long as there is at least one entry in RECEIVED still set to false or until the application decides to give up on this object.

    6. For each received GFD payload for the object (including the first payload), the steps to be taken to help recover the object are as follows:

    7. Let X be the value of the source_offset field in the GFD payload header of the MMTP packet. and let Y be the length of the payload, Y, computed by subtracting the MMTP packet and GFD payload header lengths from the total length of the received packet.

    8. The MMTP receiver copies the data into the appropriate place within the space reserved for the object and sets RECEIVED[X ... X+Y-1] = true.

    9. If all T entries of RECEIVED are true, then the receiver has recovered the entire object.

    10. Once the MMTP receiver receives a GFD payload with the B flag set to 1, it can determine the transfer length T of the

object as X+Y of the corresponding GFD payload and adjust the boolean array RECEIVED[0..T'-1] to RECEIVED[0..T-1].

6. Session Description information

The MMTP session description information may be delivered to receivers in different ways to accommodate different deployment environments. Before a receiver is able to join an MMTP session, the receiver needs to obtain the following information:

The destination information. In an IP environment, the destination IP address and port number.

An indication that the session is an MMTP session

The version number of the MMT protocol used in the MMTP session

Additionally, the MMTP session description information should contain the following information:

The start and end time of the MMTP session.

7. Congestion Control

All transport protocols used on the Internet are required to address congestion control. MMTP provides for means to the sender to adjust its sending rate to the available bandwidth. Feedback mechanisms from the client, sent as part of MMT signaling, give the sender the necessary information to estimate the available bandwidth. A description file of the content that is being streamed is also available at the sender to assist with the selection of alternative representations and in stream thinning through selection of an appropriate operation point. The MMTP sender SHALL make use of this available information to timely react to congestion.

8. IANA Considerations

This internet draft includes no request to IANA.

9. Security Considerations

Lower layer protocols may eventually provide all the security services that may be desired for applications of MMTP, including authentication, integrity, and confidentiality. These services have been specified for IP in [RFC4301].

10. References

10.1. Normative References

[isopart12] ISO/IEC, "Information technology High efficiency coding and media delivery in heterogeneous environments Part 12: File Format", 2008, <http://www.mpeg.org/>.

[mmt] ISO/IEC, "Information technology High efficiency coding and media delivery in heterogeneous environments Part 1: MPEG media transport (MMT)", 2014, <http://www.mpeg.org/>.

[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, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

10.2. Informative References

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <http://www.rfc-editor.org/info/rfc5761>.

Author's Address

Imed Bouazizi
Samsung Research America
Richardson, TX
US

Phone: +1 972 761 7023
Email: i.bouazizi@samsung.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: September 2015

Attila Mihaly
Szilveszter Nadas
Ericsson
March 9, 2015

Enablers for Transport Layer Protocol Evolution
draft-mihaly-enablers-for-tlp-evolution-00.txt

Abstract

In this document we collected requirements for TLP evolution. These requirements are the consequence of removing obstacles of TLP evolution. This results in a higher variety of expected TLP implementations and lower trust level in these. Confidentiality of communication and security is more and more important. Middleboxes which today are one of the obstacles of the evolution shall be taken into account and incentivized to cooperate in the future landscape. Resulting from the requirements we propose areas for further investigation.

Table of Contents

1. Introduction

In this document we collect requirements for TLP evolution. These requirements are the consequence of removing obstacles of evolution. Note that we are not after the requirements on the TLP development as such, but rather the requirements on the eco-system that allow fast evolution and deployment of new TLPs. When deriving the requirements we consider the interest of all actors: users, application developers, network operators, equipment vendors as well as operating system vendors to result in an infrastructure where cooperation between the parties becomes possible.

The premise of all the discussion is that TLPs need to evolve, and potentially at a faster pace than it is allowed today. A few use cases are collected to demonstrate the potential need for many application and access network related TLP features that should be available in a flexible way[FLEX].

Based on the requirements, in the second part we discuss the new features that are needed and provide ideas to meet these requirements.

1.1. Main obstacles to TLP evolution

The following obstacles were discussed at a related IAB workshop [SEMI].

1.1.1. Kernel-based TLP implementations

One problem with the innovation in the TLP area is that the transport protocols are currently implemented in the kernel. Kernel updates are generally slow. The proprietary TLP implementations that would require modifications of the existing transport protocols in the clients are therefore not usable. Each new modification needs to undergo a long standardization process, followed by a period until it is deployed in the operating systems. This process may take years.

The solution to this problem, which is also observed today in some of the new TLP designs (QUIC, WebRTC), is that, instead of the kernel, they are implemented in the application space on the top of UDP. This is a way to speed up the innovation on the transport protocol layer in the current eco-system.

1.1.2. Middleboxes

The ubiquitous deployment of middleboxes is another main reason of what is called the 'ossification' of the transport layer. Firewalls and NATs often inspect packets beyond the IP header and often drop packets based on port numbers or other identified protocol and application characteristics or, more commonly, because of non-identified protocol characteristics. The only 100 percent 'safe' protocol to be transmitted through middleboxes is HTTP and HTTPs over TCP. Even the UDP-based traffic is often dropped or policed.

Note that there may be means to implement improved TLPs with framing format embedded over the TCP layer to avoid potential problems with middleboxes. However, this introduces additional complexity and overhead.

Another, often neglected reason why the above approach is not advisable is that middleboxes and the policies they enforce have an important role in the eco-system. The internet is a critical infrastructure a number of services are using from emergency services to banking systems, which should therefore be protected against misbehaving protocols. Middleboxes that filter out unknown TLP implementations together with miss-behaving kernel-based TLP implementations are the means to keep the Internet stable.

In conclusion, a solution is needed that lets the middleboxes not block user space TLP implementations but in a way that protects against miss-behaving TLPs.

2. Requirements on the TLP framework

Overcoming the abovementioned obstacles need changes in the TLP framework. Also, when the above obstacles are overcome new requirements result from the changed ecosystem.

2.1. Enforce expected TLP behavior

One conclusion that relates to both user space TLP implementations and new TLP behavior is that protection is needed against buggy and/or malicious TLP implementations. This serves the interest of all parties and has two aspects:

- Protecting the other flows of the same user.
- Protecting the other customers using the network domain from the potential negative effect of other traffic. One can always gain a lot of capacity in a shared network just by being

aggressive. If there are no regulating network mechanisms then both the overall network utility and fairness of assessing network resources are compromised.

The behavior that needs to be verified in proprietary TLP implementations is e.g., congestion control aggressiveness, packet size, packet shaping.

2.2. Enable application access to TLPs

Introducing a proprietary TLP causes a 'walled-garden' effect, i.e., the new protocol with the new features is only available for the services provided by the developer of the transport protocol. Commercialization of the new TLPs and re-usage of open-source implementations both support fast TLP innovation, but require that the new TLPs SHOULD be selectable by different applications.

2.3. Allow for consistent TLP selection

While the previous requirement 2.2. relates to TLP selection within the host, this requirement states that the framework SHOULD allow for a consistent TLP selection for a given connection towards a certain remote host through certain legacy paths. That is, the finally selected TLP should be supported by the other endpoint as well as the network domains on the path. This is to guarantee interoperability during gradual deployment of new TLPs and their support in the network.

2.4. Allow the path influencing TLP selection

Note that middleboxes on path may explicitly be part of the TLP selection process. They may favor the usage of certain TLPs because of some enhancement functions that they could offer (see discussion in 2.6.)

2.5. Ensure expected TLP performance characteristics

By TLP performance characteristics we mean the non-functional, i.e., performance-related TLP features. Interactions needed for the selection and usage of proper TLPs (including within-the-host interactions as well as interactions with the other party and the network domain) SHOULD not result in significant degradation of expected TLP performance characteristics. For example, if a transport protocol has been designed for low setup latency then this should be preserved by introducing a framework that achieves the above goals.

Note that the above requirement is strictly valid for the case when communicating to a known party through a known network domain. In other cases a more complex setup procedure (including e.g., the exchange of security credentials with the other party or the middlebox) may be needed. In some cases it MAY be also allowed that the very first session is delayed until a proper TLP is downloaded if one of the parties does not possess the required TLP. But such cases should be as rare as possible, e.g., a browser downloading the unsupported TLP once when first demand is encountered and then storing it for future usage.

2.6. Ensure that the access providers can be part of the value chain

The framework SHALL allow access providers (ISPs and Mobile operators) provide a cooperative middlebox environment for user space TLP implementations. Middleboxes can generate value for the hosts in a number of different scenarios; we direct the reader to our recent position paper where we gave a summary of potential middlebox roles for Mobile Broadband [MBC].
Naturally, content providers and users who are not willing to participate in this cooperation and want to avoid any unwanted traffic manipulation by middleboxes MUST be able to do that. Also, the cooperation with middleboxes SHOULD be possible on different trust levels. When there is a high trust level in a middlebox, the middlebox should be able to access and manipulate higher layers, i.e., TLP or application layers.

2.7. Ensure confidentiality of end-to-end communication

A valid demand from the end-users and content providers is that the eco-system SHALL allow for confidentiality of end-to-end communication. A trend in data communication is to encrypt everything including the transport layer and above. However, if middlebox communication requires access and manipulation of the TLP fields, then object security solutions are needed to guarantee confidentiality.

2.8. Ensure security of end-to-end communication

The framework SHOULD also make all possible actions to avoid that a hostile entity along the path cannot cause any harm to the hosts by exploiting the flaws in the user space TLP implementations (at least not by using reasonable amount of processing resources).
Unfortunately it is very hard to prepare TLP implementations to be robust against any harmful change in the protocol fields. Middlebox interaction may however require that some of the TLP fields can be accessed or modified by trusted middleboxes.

2.9. Ensure user control

It SHOULD be possible allow the users to control TLP behavior. On the host side this means that the user may keep control on what TLP should be selected for a certain application. Moreover, it SHOULD be possible to control the transmission resource sharing between the different applications or streams. Otherwise, specific TLP behavior can override user expectations on the transmission resource sharing and this may cause unwanted effect on the user experience. Note that such resource sharing problems are also apparent today e.g., file sharing using a broadband access may cause bad Skype or video streaming experience.
The above requirement also has relevance for Middlebox communication, i.e., the user SHOULD have the possibility to decide what information is sent out related to a certain application or TLP.

3. Ideas for framework evolution

3.1. API definitions for TLP selection within the host

New APIs are needed to support a consistent TLP selection as specified in 2.3. Figure 1 depicts the new interfaces (shown by question marks) that a user space TLP suite brings in, in addition to the existing selection alternatives.

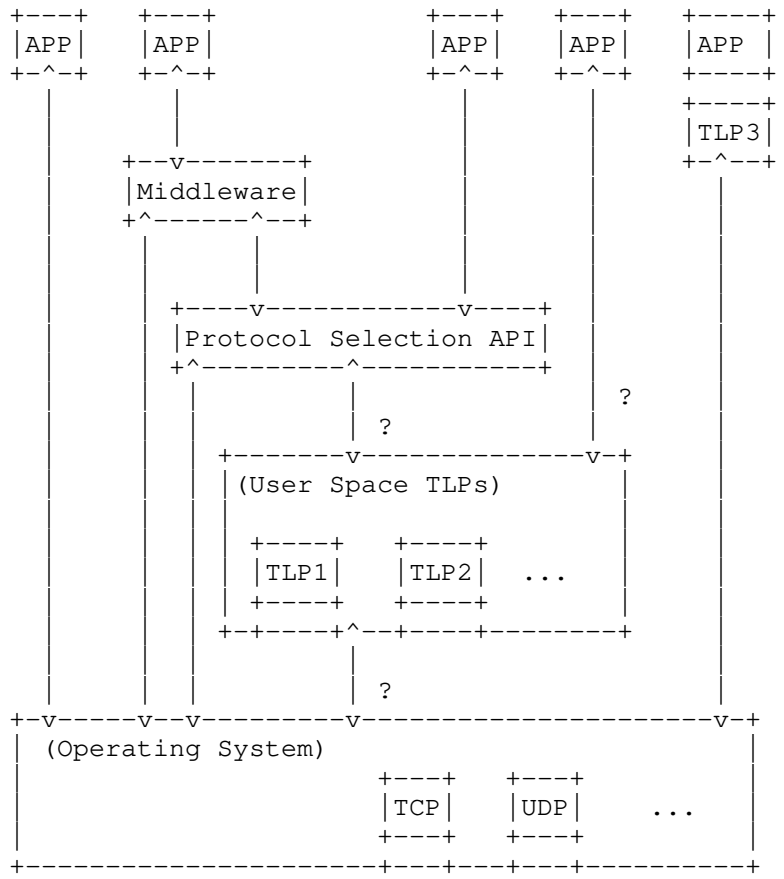


Figure 1 : Transport protocol selection alternatives with User Space transport protocol implementations

It is apparent that using middleware and high-level TLP selection APIs (like that proposed in [TAPS]) is advantageous for a framework involving user space TLPs since they separate application development from TLP development and make new, innovative TLP solutions possible also for legacy applications. However, direct selection of TPLs by the applications should also be possible. Low-level APIs to the new functions residing in the operating system are also needed (see discussion in 3.2.)

3.2. Generic TLP related functions and APIs

It is apparent that there is some generic functionality needed (i.e., transport layer functionality that applies on the top of the different user space TLP flavors) in order to fulfill the requirements in section 2. :

- Resource control component for the total transmission resources in the host, to fulfill requirement 2.9.

- Trust API that enforces a certain trust level, i.e., controls what different features and APIs a certain TLP has access to in the operating system. For example, some TLPs may control the resource control parameters, others not. This is needed because TLPs may be designed by different parties and not every TLP should be allowed to use all kernel functions.

- API in the operating system for time-critical processing, e.g., packet shaping. Time-critical processing is one functionality that may only be done efficiently in the operating system

- A common policing function (needed to fulfill the requirements in 2.1. and 2.9. on host side)
- Congestion control communication component with standard framing on congestion control 'classes' (needed to fulfill the requirements in 2.1. and 2.9. on the network side)
- Other middlebox communication component for exchanging information about the 'service' required (to support requirement 2.6.)
- A security function that verifies the different TLP implementations requested and downloaded from remote sources

Note that there may be other common features and APIs required besides those listed above. It is FFS to identify all necessary functions.

3.3. Mechanisms for consistent TLP selection

There is a need for fast TLP identification and negotiation mechanisms, which is derived from the requirements in 2.2. and 2.5. , respectively. The selection mechanisms may potentially involve middleboxes in the path. There can be many alternatives for this which is outside the scope of this document.
The selection may include communication with a trusted part of code at the other end, which provides some proof about the TLP implementation.

3.4. Security solutions for multiple trust domains

As stipulated in 2.8. , it is important that the TLP fields should not be easily changeable by nodes along the path because this can help exploiting bugs in a TLP implementation. A potential solution is to also authenticate the transport layer. However, network side enhancements on different layers could lead to improved end-to-end quality, as we discuss in 2.6. In some cases, this implies that the middleboxes should be able to modify information on the transport layer or above: transcoder proxies, parental control, TLP proxies translating between an 'old' and a 'new' TLP, etc.

The conclusion from the above is that security solutions are needed that enable that different entities have access to the information at the different layers. If there is a hint on the expected capabilities of the other end on a certain layer, the setup messages for the different layers can be grouped together in order to speed up the connection establishment, in the spirit of requirement 2.5.

3.5. In-band protocol for Middlebox communication

The smooth and fair networking so far has been highly aided by the TCP/IP "narrow-waist", i.e., the assumption that all sources use TCP-like congestion avoidance mechanisms. Misbehaving sources have been possible to identify because of the fact that TCP wire format (ACKs and sequence numbers) allows to identify both the congestion signal and the congestion avoidance action, i.e., how many data is sent upon the detection of packet losses. The proliferation of UDP-based traffic in the network with many new features with unknown and untested behavior could make even the simple traffic management that guarantees that all users get their share quite troublesome.

Middleboxes may need to enforce expected behavior in the network domain because of conflicting interests, as specified in the requirement 2.1. A potential help in that is the identification of the expected congestion behavior of TLPs (e.g., congestion control used, constant bitrate flow, or chatty flow not adapting etc.) on the network side. We recognize that any information from hosts that would represent negative discrimination is useless, since all hosts would evidently use the indication that implies the best treatment. However this 'best treatment' is not obvious and the preferred congestion treatment depends on application needs. For example, an application using CBR-type of traffic would 'sacrifice' additional bandwidth (e.g., by a rate-limiter in the network) for zero packet loss provided by the network up to a certain congestion level. To fully avoid misuse of certain congestion indicators, the network should be able to ensure some kind of fairness for the different

congestion control mechanisms, to avoid scenarios where a certain congestion behavior indicated would take more resources than others over arbitrary period of time.

By default, when there is no indication on the congestion control behavior used, the network could enforce a TCP-friendly behavior for the flow, without requiring any specific knowledge about TLP specifics (or apply a traffic treatment that result in comparable congestion 'volumes'. However, we foresee that new TLPs providing features tailored for specific application needs will have different congestion control behavior. For these flows the network may enforce a different congestion control behavior than default, given it receives the needed information.
Middlebox communication shall be made in a way that does not impede TLP evolution, which is likely faster than middlebox evolution. This requires that the framework should not require that all middleboxes should know all details (e.g., frame structure) of the TLPs. This hints towards an in-band signaling solution as proposed in [SPUD], where a new independent layer is introduced for middlebox communication and where the parameters conveyed are independent from the specific TLP used, but rather standardized information pointing to a well-defined congestion behavior.

Communication from the middlebox towards the hosts is motivated by the consistent TLP selection requirement 2.3. The end hosts may be informed about middlebox capability on coping with a certain TLP to be able to select a TLP that is supported by the network middleboxes too.

In addition to communication about TLP congestion behavior, other TLP capabilities could be exchanged and negotiated with a middlebox in order to make use of some TLP enhancement functions. Such a middlebox signaling layer also allows the endpoint convey additional IP-layer (delay, loss targets, etc.) or application-layer (proxy, transcoding feature negotiation etc.) information about their traffic (if they want to) in order to enable additional services in the spirit of the requirement 2.6. However, to support these features, the framework should allow middleboxes to setup a signaling connection in-band using the already setup user plane connection in the end-to-end communication in an authenticated way to advertise their services and communicate to the endpoints in a secure way.

Requirement 2.5. implies that the performance overhead of the above in-band signaling protocol should be minimal in the generic case, and the communication setup should be fast. Certain scenarios may be

exception to the rule, e.g., setting up communication to a new middlebox along a path, exchanging context to another middlebox due to handovers, etc.

4. On open source TLP implementations

Open source implementation may facilitate TLP evolution. The user space TLP (feature) implementations become faster, allowing smaller players leverage on the existing features that are designed and validated by bigger players. This may be further facilitated by pre-agreed software design rules for TLPs.

However, the eco-system should allow also for non-open source. Firstly, because nothing would prevent big players come out with their own user space implementations. Further, it creates a competitive eco-system allowing e.g., start-ups to design and commercialize new proprietary TLP implementations. This helps innovation.

In conclusion, open source does not bring in any specific requirements to the eco-system compared to other user space implementations. However, there could be TLP designs that facilitate re-use of open source and thus contribute to faster TLP evolution.

5. Related work

There is an on-going activity and WG in IETF targeting the definition of a transport-layer interface exposed to the applications, on which, instead of specifying a protocol, a 'transport service' is specified [TAPS]. The primary scope for the activity is to make it possible to select the best transport protocol implementation for the applications, i.e., to use the existing transport protocol implementations in the clients, which are currently never of seldom used.

There is also an EU horizon 2020 activity proposal with similar scope proposing to solve issues for NAT traversal, Path MTU discovery and falling back to a semantically-equivalent service in the selection layer, i.e., completely hidden from the application [NEAT] . As mentioned above, TLP selection layers are advantageous since they separate application development from TLP development. However, they should consider the framework extensions identified in this document.

The recently formed IAB program [SEMI] and resulting BoF discussions [SPUD] started from the very same understanding that middleboxes in the network have expectations on the packet and flow structures,

which block TLP evolution. The workshop held in January 2015 identified that a mechanism is needed for applications at the end as well as boxes along the path to explicitly declare their assumptions and intentions. The design goals identified in this document for such a communication are intended to serve as input to the protocol discussion.

6. Conventions used in this document

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

7. Security Considerations

We argue in 2.7. 2.8. and 3.4. that other means that end-to-end encryption mechanisms involving the transport layer are necessary to address middlebox communication and also ensure the confidentiality and integrity of end-to-end communication.

Exposing additional information to kernel functions and also to the network middleboxes raises a number of security questions about what should be exposed, how should be exposed etc. The present document does not address these issues, but they should be talked in future framework discussions.

8. IANA Considerations

This draft presently does not pose any requirements to IANA.

9. Conclusions

In this document we collected requirements for TLP evolution. These requirements are the consequence of removing obstacles of evolution. This results in a higher variety of expected TLP implementations and lower trust level in these. With the evolved TLPs even better performance is expected. Confidentiality of communication and security is more and more important. Middleboxes which today are one of the obstacles of the evolution shall be taken into account and incentivized to cooperate in the future landscape.

Resulting from the requirements we have identified the following ideas for further investigation.

. API definitions for TLP selection within the host

. Generic TLP related functions and APIs

. Mechanisms for consistent TLP selection

. Security solutions for multiple trust domains
. In-band protocol for Middlebox communication

We would like to invite the community to complete this analysis based on potential missing pieces of the problem space to address, further requirements, or further design goals that are useful for fast TLP evolution.

10. References

10.1. Normative References

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

10.2. Informative References

[FLEX] Flexibility of Transport Layer Protocols: Problem Statement draft-mihaly-TP-flexibility-00.txt

[TAPS] Transport Services Charter, https://datatracker.ietf.org/doc/charter-ietf-taps/

[NEAT] A New, Evolutive API and Transport Architecture for the Internet (NEAT), project proposal to Call ICT 5-2014 'Smart Networks and Novel Internet Architectures", on the Horizon 2020 Work programme

[SEMI] IAB Workshop on Stack Evolution in a Middlebox Internet (SEMI), http://www.iab.org/activities/workshops/semi/

[SPUD] A new BoF called 'Session Protocol for User Datagrams' (SPUD) is proposed for IETF92, see http://trac.tools.ietf.org/bof/trac/#Transport Stack. The background of it has been discussed in a recent IAB workshop 'Evolution in a Middlebox Internet' (SEMI), see https://www.iab.org/activities/workshops/semi/

[MBC] Nadas, S. and Loreto, S: Middleboxes in Cellular Networks, position paper to SEMI WorkShop, https://www.iab.org/wp-content/IAB-uploads/2014/12/semi2015_nadas.pdf

11. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Attila Mihaly
Ericsson
H-1117 Budapest, Irinyi Jozsef u 4-20, Hungary

Email: Attila.Mihaly@ericsson.com

Szilveszter Nadas
Ericsson
H-1117 Budapest, Irinyi Jozsef u 4-20, Hungary

Email: Szilveszter.Nadas@ericsson.com