Network Working Group
Request for Comments: 5547
Category: Standards Track

M. Garcia-Martin
Ericsson
M. Isomaki
Nokia
G. Camarillo
S. Loreto
Ericsson
P. Kyzivat
Cisco Systems

May 2009

A Session Description Protocol (SDP) Offer/Answer Mechanism to Enable File Transfer

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document provides a mechanism to negotiate the transfer of one or more files between two endpoints by using the Session Description Protocol (SDP) offer/answer model specified in RFC 3264. SDP is extended to describe the attributes of the files to be transferred. The offerer can describe either the files it wants to send or the files it would like to receive. The answerer can either accept or reject the offer separately for each individual file. The transfer of one or more files is initiated after a successful negotiation. The Message Session Relay Protocol (MSRP) is defined as the default mechanism to actually carry the files between the endpoints. The conventions on how to use MSRP for file transfer are also provided in this document.

Table of Contents

1. Introduction

The Session Description Protocol (SDP) offer/answer [RFC3264] provides a mechanism for two endpoints to arrive at a common view of a multimedia session between them. These sessions often contain real-time media streams such as voice and video, but are not limited to that. Basically, any media component type can be supported, as long as there is a specification how to negotiate it within the SDP offer/answer exchange.

The Message Session Relay Protocol (MSRP) [RFC4975] is a protocol for transmitting instant messages (IMs) in the context of a session. The protocol specification describes the usage of SDP for establishing an MSRP session. In addition to plain text messages, MSRP is able to carry arbitrary (binary) Multipurpose Internet Mail Extensions (MIME) [RFC2045] compliant content, such as images or video clips.

There are many cases where the endpoints involved in a multimedia session would like to exchange files within the context of that session. With MSRP, it is possible to embed files as MIME objects inside the stream of instant messages. MSRP also has other features that are useful for file transfer. Message chunking enables the sharing of the same transport connection between the transfer of a large file and interactive IM exchange without blocking the IM. MSRP relays [RFC4976] provide a mechanism for Network Address Translator (NAT) traversal. Finally, Secure MIME (S/MIME) [RFC3851] can be used for ensuring the integrity and confidentiality of the transferred content.

However, the baseline MSRP does not readily meet all the requirements for file transfer services within multimedia sessions. There are four main missing features:

- o The recipient must be able to distinguish "file transfer" from "file attached to IM", allowing the recipient to treat the cases differently.

- o It must be possible for the sender to send the request for a file transfer. It must be possible for the recipient to accept or decline it, using the meta information in the request. The actual transfer must take place only after acceptance by the recipient.

- o It must be possible for the sender to pass some meta information on the file before the actual transfer. This must be able to include at least content type, size, hash, and name of the file, as well as a short (human readable) description.

- It must be possible for the recipient to request a file from the sender, providing meta information about the file. The sender must be able to decide whether to send a file matching the request.

The rest of this document is organized as follows. Section 3 defines a few terms used in this document. Section 4 provides the overview of operation. Section 5 introduces the concept of the file selector. The detailed syntax and semantics of the new SDP attributes and conventions on how the existing ones are used are defined in Section 6. Section 7 discusses the file disposition types. Section 8 describes the protocol operation involving SDP and MSRP. Finally, some examples are given in Section 9.

2. Terminology

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 BCP 14, RFC 2119 [RFC2119].

3. Definitions

For the purpose of this document, the following definitions specified in RFC 3264 [RFC3264] apply:

- Answer

- Answerer

- Offer

- Offerer

Additionally, we define the following terms:

File sender: The endpoint that is willing to send a file to the file receiver.

File receiver: The endpoint that is willing to receive a file from the file sender.

File selector: A tuple of file attributes that the SDP offerer includes in the SDP in order to select a file at the SDP answerer. This is described in more detail in Section 5.

Push operation: A file transfer operation where the SDP offerer takes the role of the file sender and the SDP answerer takes the role of the file receiver.

Pull operation: A file transfer operation where the SDP offerer takes the role of the file receiver and the SDP answerer takes the role of the file sender.

4. Overview of Operation

An SDP offerer creates an SDP body that contains the description of one or more files that the offerer wants to send or receive. The offerer sends the SDP offer to the remote endpoint. The SDP answerer can accept or reject the transfer of each of those files separately.

The actual file transfer is carried out using the Message Session Relay Protocol (MSRP) [RFC4975]. Each SDP "m=" line describes an MSRP media stream used to transfer a single file at a time. That is, the transfer of multiple simultaneous files requires multiple "m=" lines and corresponding MSRP media streams. It should be noted that multiple MSRP media streams can share a single transport layer connection, so this mechanism will not lead to excessive use of transport resources.

Each "m=" line for an MSRP media stream is accompanied with a few attributes describing the file to be transferred. If the file sender generates the SDP offer, the attributes describe a local file to be sent (push), and the file receiver can use this information to either accept or reject the transfer. However, if the SDP offer is generated by the file receiver, the attributes are intended to characterize a particular file that the file receiver is willing to get (pull) from the file sender. It is possible that the file sender does not have a matching file or does not want to send the file, in which case the offer is rejected.

The attributes describing each file are provided in SDP by a set of new SDP attributes, most of which have been directly borrowed from MIME. This way, user agents can decide whether or not to accept a given file transfer based on the file's name, size, description, hash, icon (e.g., if the file is a picture), etc.

SDP direction attributes (e.g., 'sendonly', 'recvonly') are used to indicate the direction of the transfer, i.e., whether the SDP offerer is willing to send or receive the file. Assuming that the answerer accepts the file transfer, the actual transfer of the files takes place with ordinary MSRP. Note that the 'sendonly' and 'recvonly' attributes refer to the direction of MSRP SEND requests and do not preclude other protocol elements (such as 200 responses, REPORT requests, etc.).

> In principle the file transfer can work even with an endpoint supporting only regular MSRP without understanding the extensions defined herein, in a particular case where that endpoint is both the SDP answerer and the file receiver. The regular MSRP endpoint answers the offer as it would answer any ordinary MSRP offer without paying attention to the extension attributes. In such a scenario, the user experience would, however, be reduced, since the recipient would not know (by any protocol means) the reason for the session and would not be able to accept/reject it based on the file attributes.

5. File Selector

When the file receiver generates the SDP offer, this SDP offer needs to unambiguously identify the requested file at the file sender. For this purpose, we introduce the notion of a file selector, which is a tuple composed of one or more of the following individual selectors: the name, size, type, and hash of the file. The file selector can include any number of selectors, so all four of them do not always need to be present.

The purpose of the file selector is to provide enough information about the file to the remote entity, so that both the local and the remote entity can refer to the same file. The file selector is encoded in a 'file-selector' media attribute in SDP. The formal syntax of the 'file-selector' media attribute is described in Figure 1.

The file selection process is applied to all the available files at the host. The process selects those files that match each of the selectors present in the 'file-selector' attribute. The result can be zero, one, or more files, depending on the presence of the mentioned selectors in the SDP and depending on the available files in a host. The file transfer mechanism specified in this document requires that a file selector eventually results at most in a single file to be chosen. Typically, if the hash selector is known, it is enough to produce a file selector that points to exactly zero or one file. However, a file selector that selects a unique file is not always known by the offerer. Sometimes only the name, size, or type of file is known, so the file selector may result in selecting more than one file, which is an undesired case. The opposite is also true: if the file selector contains a hash selector and a name selector, there is a risk that the remote host has renamed the file, in which case, although a file whose computed hash equals the hash selector exists, the file name does not match that of the name selector. Thus, in this case, the file selection process will result in the selection of zero files.

This specification uses the Secure Hash Algorithm 1, SHA-1 [RFC3174]. If future needs require adding support for different hashing algorithms, they will be specified as extensions to this document.

Implementations according to this specification MUST implement the ’file-selector’ attribute and MAY implement any of the other attributes specified in this specification. SDP offers and answers for file transfer MUST contain a ’file-selector’ media attribute that selects the file to be transferred and MAY contain any of the other attributes specified in this specification.

The ’file-selector’ media attribute is also useful when learning the support of the file transfer offer/answer capability that this document specifies. This is further explained in Section 8.5.

6. Extensions to SDP

We define a number of new SDP [RFC4566] attributes that provide the required information to describe the transfer of a file with MSRP. These are all media-level-only attributes in SDP. The following is the formal ABNF syntax [RFC5234] of these new attributes. It is built above the SDP [RFC4566] grammar, RFC 2045 [RFC2045], RFC 2183 [RFC2183], RFC 2392 [RFC2392], and RFC 5322 [RFC5322].

```
attribute           =/ file-selector-attr / file-disp-attr /
                       file-tr-id-attr / file-date-attr /
                       file-icon-attr / file-range-attr
                       ; attribute is defined in RFC 4566

file-selector-attr   = "file-selector" [":" selector *(SP selector)]
selector             = filename-selector / filesize-selector /
                       filetype-selector / hash-selector

filename-selector    = "name:"  DQUOTE filename-string DQUOTE
                                   ; DQUOTE defined in RFC 5234
filename-string      = 1*(filename-char/percent-encoded)
filename-char        = %x01-09/%x0B-0C/%x0E-21/%x23-24/%x26-FF
                              ; any byte except NUL, CR, LF,
                              ; double quotes, or percent
percent-encoded      = "%" HEXDIG HEXDIG
                              ; HEXDIG defined in RFC 5234

filesize-selector    = "size:" filesize-value
```

```
   filesize-value       = integer        ;integer defined in RFC 4566

   filetype-selector    = "type:" type "/" subtype *(";" ft-parameter)
   ft-parameter         = attribute "=" DQUOTE value-string DQUOTE
                                      ; attribute is defined in RFC 2045
                        ; free insertion of linear-white-space is not
                        ; permitted in this context.
                        ; note: value-string has to be re-encoded
                        ; when translating between this and a
                        ; Content-Type header.
   value-string         = filename-string

   hash-selector        = "hash:" hash-algorithm ":" hash-value
   hash-algorithm       = token     ; see IANA Hash Function
                                    ; Textual Names registry
                                    ; only "sha-1" currently supported
   hash-value           = 2HEXDIG *(":" 2HEXDIG)
                                ; Each byte in upper-case hex, separated
                                ; by colons.
                                ; HEXDIG defined in RFC 5234

   file-tr-id-attr      = "file-transfer-id:" file-tr-id-value
   file-tr-id-value     = token

   file-disp-attr       = "file-disposition:" file-disp-value
   file-disp-value      = token

   file-date-attr       = "file-date:"  date-param *(SP date-param)

   date-param           = c-date-param / m-date-param / r-date-param
   c-date-param         = "creation:" DQUOTE date-time DQUOTE
   m-date-param         = "modification:" DQUOTE date-time DQUOTE
   r-date-param         = "read:" DQUOTE date-time DQUOTE
                             ; date-time is defined in RFC 5322
                             ; numeric timezones (+HHMM or -HHMM)
                             ; must be used
                             ; DQUOTE defined in RFC 5234 files.

   file-icon-attr       = "file-icon:" file-icon-value
   file-icon-value      = cid-url        ; cid-url defined in RFC 2392

   file-range-attr      = "file-range:" start-offset "-" stop-offset
   start-offset         = integer        ; integer defined in RFC 4566
   stop-offset          = integer / "*"
```

Figure 1: Syntax of the SDP extension

When used for capability query (see Section 8.5), the ’file-selector’ attribute MUST NOT contain any selector, because its presence merely indicates compliance to this specification.

When used in an SDP offer or answer, the ’file-selector’ attribute MUST contain at least one selector. Selectors characterize the file to be transferred. There are four selectors in this attribute: ’name’, ’size’, ’type’, and ’hash’.

The ’name’ selector in the ’file-selector’ attribute contains the filename of the content enclosed in double quotes. The filename is encoded in UTF-8 [RFC3629]. Its value SHOULD be the same as the ’filename’ parameter of the Content-Disposition header field [RFC2183] that would be signaled by the actual file transfer. If a file name contains double quotes or any other character that the syntax does not allow in the ’name’ selector, they MUST be percent-encoded. The ’name’ selector MUST NOT contain characters that can be interpreted as directory structure by the local operating system. If such characters are present in the file name, they MUST be percent-encoded.

> Note that the ’name’ selector might still contain characters that, although not meaningful for the local operating system, might still be meaningful to the remote operating system (e.g., ’\’, ’/’, ’:’). Therefore, implementations are responsible for sanitizing the input received from the remote endpoint before doing a local operation in the local file system, such as the creation of a local file. Among other things, implementations can percent-encode characters that are meaningful to the local operating system before doing file system local calls.

The ’size’ selector in the ’file-selector’ attribute indicates the size of the file in octets. The value of this attribute SHOULD be the same as the ’size’ parameter of the Content-Disposition header field [RFC2183] that would be signaled by the actual file transfer. Note that the ’size’ selector merely includes the file size, and does not include any potential overhead added by a wrapper, such as message/cpim [RFC3862].

The ’type’ selector in the ’file-selector’ attribute contains the MIME media and submedia types of the content. In general, anything that can be expressed in a Content-Type header field (see RFC 2045 [RFC2045]) can also be expressed with the ’type’ selectors. Possible MIME Media Type values are the ones listed in the IANA registry for MIME Media Types [IANA]. Zero or more parameters can follow. When translating parameters from a Content-Type header and a ’type’ selector, the parameter has to be re-encoded prior to its accommodation as a parameter of the ’type’ selector (see the ABNF syntax of ’ft-parameter’).

The ’hash’ selector in the ’file-selector’ attribute provides a hash computation of the file to be transferred. This is commonly used by file transfer protocols. For example, FLUTE [FLUTE-REV] uses hashes (called message digests) to verify the contents of the transfer. The purpose of the ’hash’ selector is two-fold: On one side, in pull operations, it allows the file receiver to identify a remote file by its hash rather than by its file name, providing that the file receiver has learned the hash of the remote file by some out-of-band mechanism. On the other side, in either push or pull operations, it allows the file receiver to verify the contents of the received file, or even avoid unnecessary transmission of an existing file.

> The address space of the SHA-1 algorithm is big enough to avoid any collision in hash computations in between two endpoints. When transferring files, the actual file transfer protocol should provide reliable transmission of data, so verifications of received files should always succeed. However, if endpoints need to protect the integrity of a file, they should use some other mechanism than the ’hash’ selector specified in this memo.

The ’hash’ selector includes the hash algorithm and its value. Possible hash algorithms are those defined in the IANA registry of Hash Function Textual Names [IANA]. Implementations according to this specification MUST add a 160-bit string resulting from the computation of US Secure Hash Algorithm 1 (SHA1) [RFC3174] if the ’hash’ selector is present. If need arises, extensions can be drafted to support several hashing algorithms. Therefore, implementations according to this specification MUST be prepared to receive SDP containing more than a single ’hash’ selector in the ’file-selector’ attribute.

The value of the ’hash’ selector is the byte string resulting from applying the hash algorithm to the content of the whole file, even when the file transfer is limited to a number of octets (i.e., the ’file-range’ attribute is indicated).

The ’file-transfer-id’ attribute provides a randomly chosen globally unique identification to the actual file transfer. It is used to distinguish a new file transfer request from a repetition of the SDP (or the fraction of the SDP that deals with the file description). This attribute is described in much greater detail in Section 8.1.

The ’file-disposition’ attribute provides a suggestion to the other endpoint about the intended disposition of the file. Section 7 provides further discussion of the possible values. The value of this attribute SHOULD be the same as the disposition type parameter of the Content-Disposition header field [RFC2183] that would be signaled by the actual file transfer protocol.

The ’file-date’ attribute indicates the dates on which the file was created, modified, or last read. This attribute MAY contain a combination of the ’creation’, ’modification’, and ’read’ parameters, but MUST NOT contain more than one of each type .

The ’creation’ parameter indicates the date on which the file was created. The value MUST be a quoted string that contains a representation of the creation date of the file in RFC 5322 [RFC5322] ’date-time’ format. Numeric timezones (+HHMM or -HHMM) MUST be used. The value of this parameter SHOULD be the same as the ’creation-date’ parameter of the Content-Disposition header field [RFC2183] that would be signaled by the actual file transfer protocol.

The ’modification’ parameter indicates the date on which the file was last modified. The value MUST be a quoted string that contains a representation of the last modification date to the file in RFC 5322 [RFC5322] ’date-time’ format. Numeric timezones (+HHMM or -HHMM) MUST be used. The value of this parameter SHOULD be the same as the ’modification-date’ parameter of the Content-Disposition header field [RFC2183] that would be signaled by the actual file transfer protocol.

The ’read’ parameter indicates the date on which the file was last read. The value MUST be a quoted string that contains a representation of the last date the file was read in RFC 5322 [RFC5322] ’date-time’ format. Numeric timezones (+HHMM or -HHMM) MUST be used. The value of this parameter SHOULD be the same as the ’read-date’ parameter of the Content-Disposition header field [RFC2183] that would be signaled by the actual file transfer protocol.

The ’file-icon’ attribute can be useful with certain file types such as images. It allows the file sender to include a pointer to a body that includes a small preview icon representing the contents of the file to be transferred, which the file receiver can use to determine whether it wants to receive such file. The ’file-icon’ attribute contains a Content-ID URL, which is specified in RFC 2392 [RFC2392]. Section 8.8 contains further considerations about the ’file-icon’ attribute.

The ’file-range’ attribute provides a mechanism to signal a chunk of a file rather than the complete file. This enables use cases where a file transfer can be interrupted and resumed, even perhaps changing one of the endpoints. The ’file-range’ attribute contains the "start offset" and "stop offset" of the file, separated by a dash "-". The "start offset" value refers to the octet position of the file where the file transfer should start. The first octet of a file is denoted by the ordinal number "1". The "stop offset" value refers to the octet position of the file where the file transfer should stop, inclusive of this octet. The "stop offset" value MAY contain a "*" if the total size of the file is not known in advance. The absence of this attribute indicates a complete file, i.e., as if the ’file-range’ attribute would have been present with a value "1-*". The ’file-range’ attribute must not be confused with the Byte-Range header in MSRP. The former indicates the portion of a file that the application would read and pass onto the MSRP stack for transportation. From the point of view of MSRP, the portion of the file is viewed as a whole message. The latter indicates the number of bytes of that message that are carried in a chunk and the total size of the message. Therefore, MSRP starts counting the delivered message at octet number 1, independently of the position of that octet in the file.

The following is an example of an SDP body that contains the extensions defined in this memo:

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.atlanta.example.com
s=
c=IN IP4 host.atlanta.example.com
t=0 0
m=message 7654 TCP/MSRP *
i=This is my latest picture
a=sendonly
a=accept-types:message/cpim
a=accept-wrapped-types:*
a=path:msrp://atlanta.example.com:7654/jshA7we;tcp
a=file-selector:name:"My cool picture.jpg" type:image/jpeg
  size:32349 hash:sha-1:
  72:24:5F:E8:65:3D:DA:F3:71:36:2F:86:D4:71:91:3E:E4:A2:CE:2E
a=file-transfer-id:vBnG916bdberum2fFEABR1FR3ExZMUrd
a=file-disposition:attachment
a=file-date:creation:"Mon, 15 May 2006 15:01:31 +0300"
a=file-icon:cid:id2@alicepc.example.com
a=file-range:1-32349
```

Figure 2: Example of SDP describing a file transfer

NOTE: The 'file-selector' attribute in the above figure is split in three lines for formatting purposes. Real implementations will encode it in a single line.

7. File Disposition Types

The SDP offer/answer for file transfer allows the file sender to indicate a preferred disposition of the file to be transferred in a new 'file-disposition' attribute. In principle, any value listed in the IANA registry for Mail Content Disposition Values [IANA] is acceptable; however, most of them may not be applicable.

There are two content dispositions of interest for file transfer operations. On one hand, the file sender may just want the file to be rendered immediately in the file receiver's device. On the other hand, the file sender may just want to indicate to the file receiver that the file should not be rendered at the reception of the file. The recipient's user agent may want to interact with the user regarding the file disposition or it may save the file until the user takes an action. In any case, the exact actions are implementation dependent.

To indicate that a file should be automatically rendered, this memo uses the existing 'render' value of the Content Disposition type in the new 'file-disposition' attribute in SDP. To indicate that a file should not be automatically rendered, this memo uses the existing 'attachment' value of the Content-Disposition type in the new 'file-disposition' attribute in SDP. The default value is 'render', i.e., the absence of a 'file-disposition' attribute in the SDP has the same semantics as 'render'.

> The disposition value 'attachment' is specified in RFC 2183 [RFC2183] with the following definition:
>
> > "Body parts can be designated 'attachment' to indicate that they are separate from the main body of the mail message, and that their display should not be automatic, but contingent upon some further action of the user."
>
> In the case of this specification, the 'attachment' disposition type is used to indicate that the display of the file should not be automatic, but contingent upon some further action of the user.

8. Protocol Operation

This section discusses how to use the parameters defined in Section 6 in the context of an offer/answer [RFC3264] exchange. Additionally, this section also discusses the behavior of the endpoints using MSRP.

A file transfer session is initiated by the offerer sending an SDP offer to the answerer. The answerer either accepts or rejects the file transfer session and sends an SDP answer to the offerer.

We can differentiate two use cases, depending on whether the offerer is the file sender or file receiver:

1. The offerer is the file sender, i.e., the offerer wants to transmit a file to the answerer. Consequently, the answerer is the file receiver. In this case, the SDP offer contains a 'sendonly' attribute, and accordingly the SDP answer contains a 'recvonly' attribute.

2. The offerer is the file receiver, i.e., the offerer wants to fetch a file from the answerer. Consequently, the answerer is the file sender. In this case, the SDP offer contains a session or media 'recvonly' attribute, and accordingly the SDP answer contains a session or media 'sendonly' attribute.

8.1. The 'file-transfer-id' Attribute

This specification creates an extension to the SDP offer/answer model [RFC3264], and because of that, it is assumed that the existing SDP behavior is kept intact. The SDP behavior requires, for example, that SDP is sent again to the remote party in situations where the media description or perhaps other SDP parameters have not changed with respect to a previous offer/answer exchange. Let's consider the SIP Session Timer (RFC 4028) [RFC4028], which uses re-INVITE requests to refresh sessions. RFC 4028 recommends to send unmodified SDP in a re-INVITE to refresh the session. Should this re-INVITE contain SDP describing a file transfer operation and occur while the file transfer was still going on, there would be no means to detect whether the SDP creator wanted to abort the current file transfer operation and initiate a new one or the SDP file description was included in the SDP due to other reasons (e.g., session timer refresh).

A similar scenario occurs when two endpoints have successfully agreed on a file transfer, which is currently taking place when one of the endpoints wants to add additional media streams to the existing session. In this case, the endpoint sends a re-INVITE request that contains the SDP. The SDP needs to maintain the media descriptions for the current ongoing file transfer and add the new media descriptions. The problem is that the other endpoint is not able to determine whether or not a new file transfer is requested.

In other cases, a file transfer was successfully completed. Then, if an endpoint resends the SDP offer with the media stream for the file transfer, then the other endpoint wouldn't be able to determine whether or not a new file transfer should start.

To address these scenarios, this specification defines the 'file-transfer-id' attribute, which contains a globally unique random identifier allocated to the file transfer operation. The file transfer identifier helps both endpoints to determine whether the SDP offer is requesting a new file transfer or it is a repetition of the SDP. A new file transfer is one that, in case of acceptance, will provoke the actual transfer of a file. This is typically the case of new offer/answer exchanges, or in cases where an endpoint wants to abort the existing file transfer and restart the file transfer once more. On the other hand, the repetition of the SDP does not lead to any actual file to be transferred, potentially because the file transfer is still going on or because it has already finished. This is the case of repeated offer/answer exchanges, which can be due to a number of reasons (session timer, addition/removal of other media types in the SDP, update in SDP due to changes in other session parameters, etc.).

Implementations according to this specification MUST include a 'file-transfer-id' attribute in SDP offers and answers. The SDP offerer MUST select a file transfer identifier according to the syntax and add it to the 'file-transfer-id' attribute. The SDP answerer MUST copy the value of the 'file-transfer-id' attribute in the SDP answer.

The file transfer identifier MUST be unique within the current session (never used before in this session), and it is RECOMMENDED to be unique across different sessions. It is RECOMMENDED to select a relatively big random identifier (e.g., 32 characters) to avoid duplications. The SDP answerer MUST keep track of the proposed file transfer identifiers in each session and copy the value of the received file transfer identifier in the SDP answer.

If a file transfer is suspended and resumed at a later time, the resumption is considered a new file transfer (even when the file to be transferred is the same); therefore, the SDP offerer MUST choose a new file transfer identifier.

If an endpoint sets the port number to zero in the media description of a file transfer, for example, because it wants to reject the file transfer operation, then the SDP answer MUST mirror the value of the 'file-transfer-id' attribute included in the SDP offer. This effectively means that setting a media stream to zero has higher precedence than any value that the 'file-transfer-id' attribute can take.

As a side effect, the ’file-transfer-id’ attribute can be used for aborting and restarting again an ongoing file transfer. Assume that two endpoints agree on a file transfer and the actual transfer of the file is taking place. At some point in time in the middle of the file transfer, one endpoint sends a new SDP offer, equal to the initial one except for the value of the ’file-transfer-id’ attribute, which is a new globally unique random value. This indicates that the offerer wants to abort the existing transfer and start a new one, according to the SDP parameters. The SDP answerer SHOULD abort the ongoing file transfer, according to the procedures of the file transfer protocol (e.g., MSRP), and start sending file once more from the initial requested octet. Section 8.4 further discusses aborting a file transfer.

If an endpoint creates an SDP offer where the ’file-transfer-id’ attribute value does not change with respect to a previously sent one, but the file selector changes so that a new file is selected, then this is considered an error, and the SDP answerer MUST abort the file transfer operation (e.g., by setting the port number to zero in the SDP answer). Note that endpoints MAY change the ’file-selector’ attribute as long as the selected file does not change (e.g., by adding a hash selector); however, it is RECOMMENDED that endpoints do not change the value of the ’file-selector’ attribute if it is requested to transfer the same file described in a previous SDP offer/answer exchange.

Figure 3 summarizes the relation of the ’file-transfer-id’ attribute with the file selector in subsequent SDP exchanges.

\ file selector ’file-transfer-id’ \	different file	same file
changed	new file transfer operation	new file transfer operation
unchanged	error	existing file transfer operation

Figure 3: Relation of the ’file-transfer-id’ attribute with the selector of the file in a subsequent SDP exchange

In another scenario, an endpoint that has successfully transferred a file wants to send an SDP due to other reasons than the transfer of a file. The SDP offerer creates an SDP file description that maintains the media description line corresponding to the file transfer. The SDP offerer MUST then set the port number to zero and MUST keep the same value of the 'file-transfer-id' attribute that the initial file transfer got.

8.2. Offerer's Behavior

An offerer who wishes to send or receive one or more files to or from an answerer MUST build an SDP [RFC4566] description of a session containing one "m=" line per file. When MSRP is used as the transfer mechanism, each "m=" line also describes a single MSRP session, according to the MSRP [RFC4975] procedures. Any "m=" lines that may have already been present in a previous SDP exchange are normally kept unmodified; the new "m=" lines are added afterwards (Section 8.6 describes cases when "m=" lines are reused). All the media line attributes specified and required by MSRP [RFC4975] (e.g., "a=path", "a=accept-types", etc.) MUST be included as well.

8.2.1. The Offerer Is a File Sender

In a push operation, the file sender creates an SDP offer describing the file to be sent. The file sender MUST add a 'file-selector' attribute media line containing at least one of the 'type', 'size', or 'hash' selectors in indicating the type, size, or hash of the file, respectively. If the file sender wishes to start a new file transfer, the file sender MUST add a 'file-transfer-id' attribute containing a new globally unique random identifier value. Additionally, the file sender MUST add a session or media 'sendonly' attribute to the SDP offer. Then the file sender sends the SDP offer to the file receiver.

> Not all the selectors in the 'file-selector' attribute might be known when the file sender creates the SDP offer, for example, because the host is still processing the file.
>
> The 'hash' selector in the 'file-selector' attribute contains valuable information for the file receiver to identify whether the file is already available and need not be transmitted.

The file sender MAY also add a 'name' selector in the 'file-selector' attribute, and 'file-icon', 'file-disposition', and 'file-date' attributes further describing the file to be transferred. The 'file-disposition' attribute provides a presentation suggestion (for example: the file sender would like the file receiver to render the file or not). The three date attributes provide the answerer with an indication of the age of the file. The file sender MAY also add a 'file-range' attribute indicating the start and stop offsets of the file.

When the file sender receives the SDP answer, if the port number of the answer for a file request is non-zero, the file sender starts the transfer of the file according to the negotiated parameters in SDP.

8.2.2. The Offerer Is a File Receiver

In a pull operation, the file receiver creates the SDP offer and sends it to the file sender. The file receiver MUST include a 'file-selector' attribute and MUST include, at least, one of the selectors defined for such attribute (i.e., 'name', 'type', 'size', or 'hash'). In many cases, if the hash of the file is known, that is enough to identify the file; therefore, the offerer can include only a 'hash' selector. However, particularly in cases where the hash of the file is unknown, the file name, size, and type can provide a description of the file to be fetched. If the file receiver wishes to start a new file transfer, it MUST add a 'file-transfer-id' attribute containing a new globally unique random value. The file receiver MAY also add a 'file-range' attribute indicating the start and stop offsets of the file. There is no need for the file receiver to include further file attributes in the SDP offer; thus, it is RECOMMENDED that SDP offerers do not include any other file attribute defined by this specification (other than the mandatory ones). Additionally, the file receiver MUST add a session or media 'recvonly' attribute in the SDP offer. Then, the file receiver sends the SDP offer to the file sender.

When the file receiver receives the SDP answer, if the port number of the answer for a file request is non-zero, then the file receiver should receive the file using the protocol indicated in the "m=" line. If the SDP answer contains a supported hashing algorithm in the 'hash' selectors of the 'file-selector' attribute, then the file receiver SHOULD compute the hash of the file after its reception and check it against the hash received in the answer. In case the computed hash does not match the one contained in the SDP answer, the file receiver SHOULD consider that the file transfer failed and SHOULD inform the user. Similarly, the file receiver SHOULD also verify that the other selectors declared in the SDP match the file properties, otherwise, the file receiver SHOULD consider that the file transfer failed and SHOULD inform the user.

8.2.3. SDP Offer for Several Files

An offerer that wishes to send or receive more than one file generates an "m=" line per file along with the file attributes described in this specification. This way, the answerer can reject individual files by setting the port number of their associated "m=" lines to zero, as per regular SDP [RFC4566] procedures. Similarly, the answerer can accept each individual file separately by setting the port number of their associated "m=" lines to non-zero value. Each file has its own file transfer identifier, which uniquely identifies each file transfer.

Using an "m=" line per file implies that different files are transferred using different MSRP sessions. However, all those MSRP sessions can be set up to run over a single TCP connection, as described in Section 8.1 of RFC 4975 [RFC4975]. The same TCP connection can also be reused for sequential file transfers.

8.3. Answerer's Behavior

If the answerer wishes to reject a file offered by the offerer, it sets the port number of the "m=" line associated with the file to zero, as per regular SDP [RFC4566] procedures. The rejected answer MUST contained a 'file-selector' and 'file-transfer-id' attributes whose values mirror the corresponding values of the SDP offer.

If the answerer decides to accept the file, it proceeds as per regular MSRP [RFC4975] and SDP [RFC4566] procedures.

8.3.1. The Answerer Is a File Receiver

In a push operation, the SDP answerer is the file receiver. When the file receiver gets the SDP offer, it first examines the port number. If the port number is set to zero, the file transfer operation is closed, and no more data is expected over the media stream. Then, if the port number is different than zero, the file receiver inspects the 'file-transfer-id' attribute. If the value of the 'file-transfer-id' attribute has been previously used, then the existing session remains without changes; perhaps the file transfer is still in progress, or perhaps it has concluded, but there are no changes with respect to the current status. In any case, independently of the port number, the SDP answerer creates a regular SDP answer and sends it to the offerer.

If the port number is different than zero and the SDP offer contains a new 'file-transfer-id' attribute, then it is signaling a request for a new file transfer. The SDP answerer extracts the attributes and parameters that describe the file and typically requests permission from the user to accept or reject the reception of the file. If the file transfer operation is accepted, the file receiver MUST create an SDP answer according to the procedures specified in RFC 3264 [RFC3264]. If the offer contains 'name', 'type', or 'size' selectors in the 'file-selector' attribute, the answerer MUST copy them into the answer. The file receiver copies the value of the 'file-transfer-id' attribute to the SDP answer. Then the file receiver MUST add a session or media 'recvonly' attribute according to the procedures specified in RFC 3264 [RFC3264]. The file receiver MUST NOT include ’file-icon’, ’file-disposition’, or ’file-date’ attributes in the SDP answer.

The file receiver can use the hash to find out if a local file with the same hash is already available, in which case, this could imply the reception of a duplicated file. It is up to the answerer to determine whether or not the file transfer is accepted in case of a duplicated file.

If the SDP offer contains a ’file-range’ attribute and the file receiver accepts to receive the range of octets declared in there, the file receiver MUST include a ’file-range’ attribute in the SDP answer with the same range of values. If the file receiver does not accept the reception of that range of octets, it SHOULD reject the transfer of the file.

When the file transfer operation is complete, the file receiver computes the hash of the file and SHOULD verify that it matches the hash declared in the SDP. If they do not match, the file receiver SHOULD consider that the file transfer failed and SHOULD inform the user. Similarly, the file receiver SHOULD also verify that the other selectors declared in the SDP match the file properties; otherwise, the file receiver SHOULD consider that the file transfer failed and SHOULD inform the user.

8.3.2. The Answerer Is a File Sender

In a pull operation the answerer is the file sender. In this case, the SDP answerer MUST first inspect the value of the ’file-transfer-id’ attribute. If it has not been previously used throughout the session, then acceptance of the file MUST provoke the transfer of the file over the negotiated protocol. However, if the value has been previously used by another file transfer operation within the session, then the file sender MUST NOT alert the user and MUST NOT start a new transfer of the file. No matter whether or not an actual file transfer is initiated, the file sender MUST create a proper SDP answer that contains the ’file-transfer-id’ attribute with the same value received in the SDP offer, and then it MUST continue processing the SDP answer.

The file sender MUST always create an SDP answer according to the SDP offer/answer procedures specified in RFC 3264 [RFC3264]. The file sender inspects the file selector of the received SDP offer, which is encoded in the ’file-selector’ media attribute line. Then the file sender applies the file selector, which implies selecting those files that match one by one with the ’name’, ’type’, ’size’, and ’hash’ selectors of the ’file-selector’ attribute line (if they are present). The file selector identifies zero or more candidate files to be sent. If the file selector is unable to identify any file, then the answerer MUST reject the MSRP stream for file transfer by setting the port number to zero, and then the answerer SHOULD also reject the SDP as per procedures in RFC 3264 [RFC3264], if this is the only stream described in the SDP offer.

If the file selector points to a single file and the file sender decides to accept the file transfer, the file sender MUST create an SDP answer that contains a 'sendonly' attribute, according to the procedures described in RFC 3264 [RFC3264]. The file sender SHOULD add a 'hash' selector in the answer with the locally computed SHA-1 hash over the complete file. If a hash value computed by the file sender differs from that specified by the file receiver, the file sender can either send the file without that hash value or reject the request by setting the port in the media stream to zero. The file sender MAY also include a 'type' selector in the 'file-selector' attribute line of the SDP answer. The answerer MAY also include 'file-icon' and 'file-disposition' attributes to further describe the file. Although the answerer MAY also include a 'name' and 'size' selectors in the 'file-selector' attribute, and a 'file-date' attribute, it is RECOMMENDED not to include them in the SDP answer if the actual file transfer protocol (e.g., MSRP [RFC4975]) can accommodate a Content-Disposition header field [RFC2183] with the equivalent parameters.

> The whole idea of adding file descriptors to SDP is to provide a mechanism where a file transfer can be accepted prior to its start. Adding any SDP attributes that are otherwise signaled later in the file transfer protocol would just duplicate the information, but will not provide any information to the offerer to accept or reject the file transfer (note that the offerer is requesting a file).

Last, if the file selector points to multiple candidate files, the answerer MAY use some local policy, e.g., consulting the user, to choose one of them to be defined in the SDP answer. If that choice cannot be done, the answerer SHOULD reject the MSRP media stream for file transfer (by setting the port number to zero).

> If the need arises, future specifications can provide a suitable mechanism that allows to either select multiple files or, e.g., resolve ambiguities by returning a list of files that match the file selector.

If the SDP offer contains a 'file-range' attribute and the file sender accepts to send the range of octets declared in there, the file sender MUST include a 'file-range' attribute in the SDP answer with the same range of values. If the file sender does not accept sending that range of octets, it SHOULD reject the transfer of the file.

8.4. Aborting an Ongoing File Transfer Operation

Either the file sender or the file receiver can abort an ongoing file transfer at any time. Unless otherwise noted, the entity that aborts an ongoing file transfer operation MUST follow the procedures at the media level (e.g., MSRP) and at the signaling level (SDP offer/answer), as described below.

Assume the scenario depicted in Figure 4 where a file sender wishes to abort an ongoing file transfer without initiating an alternative file transfer. Assume that an ongoing MSRP SEND request is being transmitted. The file sender aborts the MSRP message by including the '#' character in the continuation field of the end-line of a SEND request, according to the MSRP procedures (see Section 7.1 of RFC 4975 [RFC4975]). Since a file is transmitted as one MSRP message, aborting the MSRP message effectively aborts the file transfer. The file receiver acknowledges the MSRP SEND request with a 200 response. Then the file sender SHOULD close the MSRP session by creating a new SDP offer that sets the port number to zero in the related "m=" line that describes the file transfer (see Section 8.2 of RFC 3264 [RFC3264]). This SDP offer MUST conform with the requirements of Section 8.2.1. The 'file-transfer-id' attribute MUST be the same attribute that identifies the ongoing transfer. Then the file sender sends this SDP offer to the file receiver.

> Rather than close the MSRP session by setting the port number to zero in the related "m=" line, the file sender could also tear down the whole session, e.g., by sending a SIP BYE request.

Note that it is the responsibility of the file sender to tear down the MSRP session. Implementations should be prepared for misbehaviors and implement measures to avoid hang states. For example, upon expiration of a timer the file receiver can close the aborted MSRP session by using regular MSRP procedures.

A file receiver that receives the above SDP offer creates an SDP answer according to the procedures of the SDP offer/answer (RFC 3264 [RFC3264]). This SDP answer MUST conform with the requirements of Section 8.3.1. Then the file receiver sends this SDP answer to the file sender.

```
                        File sender            File receiver
                            |                        |
                            |\                       |
                            | \                      |
                            |  \                     |
                            |   \                    |
                            |    \                   |
                            |     \                  |
                     abort->|      \  MSRP SEND (#)  |
                            |       +--------------->|
                            | MSRP 200               |
                            |<-----------------------|
                            | re-INVITE (SDP offer)  |
                            |----------------------->|
                            | SIP 200 OK (SDP answer)|
                            |<-----------------------|
                            | SIP ACK                |
                            |----------------------->|
                            |                        |
```

Figure 4: File sender aborts an ongoing file transfer

When the file receiver wants to abort the file transfer, there are two possible scenarios, depending on the value of the Failure-Report header in the ongoing MSRP SEND request. Assume now the scenario depicted in Figure 5 where the MSRP SEND request includes a Failure-Report header set to a value different than "no". When the file receiver wishes to abort the ongoing file transfer, the file receiver generates an MSRP 413 response to the current MSRP SEND request (see Section 10.5 of RFC 4975 [RFC4975]). Then the file receiver MUST close the MSRP session by generating a new SDP offer that sets the port number to zero in the related "m=" line that describes the file transfer (see Section 8.2 of RFC 3264 [RFC3264]). This SDP offer MUST conform with the requirements expressed in Section 8.2.2. The 'file-transfer-id' attribute MUST be the same attribute that identifies the ongoing transfer. Then the file receiver sends this SDP offer to the file sender.

```
                     File sender            File receiver
                         |                        |
                         |\                       |
                         | \  MSRP SEND           |
                         |  \ Failure-Report: yes |
                         |   \                    |
                         |    \                   |
                         |     \                  |
                         |      \                 |
                         |       \                |
                         |        \               |
                         | MSRP 413               |<-abort
                         |<-----------------------|
                         |          \   (#)       |
                         |           +----------->|
                         | re-INVITE (SDP offer)  |
                         |<-----------------------|
                         | SIP 200 OK (SDP answer)|
                         |----------------------->|
                         | SIP ACK                |
                         |<-----------------------|
                         |                        |
```

Figure 5: File receiver aborts an ongoing file transfer. Failure-Report set to a value different than "no" in MSRP

In another scenario, depicted in Figure 6, an ongoing file transfer is taking place, where the MSRP SEND request contains a Failure-Report header set to the value "no". When the file receiver wants to abort the ongoing transfer, it MUST close the MSRP session by generating a new SDP offer that sets the port number to zero in the related "m=" line that describes the file transfer (see Section 8.2 of RFC 3264 [RFC3264]). This SDP offer MUST conform with the requirements expressed in Section 8.2.2. The 'file-transfer-id' attribute MUST be the same attribute that identifies the ongoing transfer. Then the file receiver sends this SDP offer to the file sender.

```
                     File sender            File receiver
                         |                        |
                         |\                       |
                         | \  MSRP SEND           |
                         |  \ Failure-Report: no  |
                         |   \                    |
                         |    \                   |
                         |     \                  |
                         |      \                 |
                         |       \                |
                         |        \               |
                         | re-INVITE (SDP offer)  |<-abort
                         |<-----------------------|
                         |          \   (#)       |
                         |           +----------->|
                         | MSRP 200               |
                         |<-----------------------|
                         | SIP 200 OK (SDP answer)|
                         |----------------------->|
                         | SIP ACK                |
                         |<-----------------------|
                         |                        |
```

Figure 6: File receiver aborts an ongoing file transfer. Failure-Report set to "no" in MSRP

A file sender that receives an SDP offer setting the port number to zero in the related "m=" line for file transfer, first, if an ongoing MSRP SEND request is being transmitted, aborts the MSRP message by including the ’#’ character in the continuation field of the end-line of a SEND request, according to the MSRP procedures (see Section 7.1 of RFC 4975 [RFC4975]). Since a file is transmitted as one MSRP message, aborting the MSRP message effectively aborts the file transfer. Then the file sender creates an SDP answer according to the procedures of the SDP offer/answer (RFC 3264 [RFC3264]). This SDP answer MUST conform with the requirements of Section 8.3.2. Then the file sender sends this SDP answer to the file receiver.

8.5. Indicating File Transfer Offer/Answer Capability

The SDP offer/answer model [RFC3264] provides provisions for indicating a capability to another endpoint (see Section 9 of RFC 3264 [RFC3264]). The mechanism assumes a high-level protocol, such as SIP [RFC3261], that provides a capability query (such as a SIP OPTIONS request). RFC 3264 [RFC3264] indicates how to build the SDP that is included in the response to such capability query. As such, RFC 3264 indicates that an endpoint builds an SDP body that contains an "m=" line containing the media type (message, for MSRP). An endpoint that implements the procedures specified in this document SHOULD also add a ’file-selector’ media attribute for the "m=message" line. The ’file-selector’ media attribute MUST be empty, i.e., it MUST NOT contain any selector. The endpoint MUST NOT add any of the other file attributes defined in this specification.

8.6. Reusage of Existing "m=" Lines in SDP

The SDP offer/answer model [RFC3264] provides rules that allow SDP offerers and answerers to modify an existing media line, i.e., reuse an existing media line with different attributes. The same is also possible when SDP signals a file transfer operation according to the rules of this memo. Therefore, the procedures defined in RFC 3264 [RFC3264], in particular those defined in Section 8.3, MUST apply for file transfer operations. An endpoint that wants to reuse an existing "m=" line to start the file transfer of another file creates a different ’file-selector’ attribute and selects a new globally unique random value of the ’file-transfer-id’ attribute.

If the file offerer resends an SDP offer with a port different than zero, then the ’file-transfer-id’ attribute determines whether a new file transfer will start or whether the file transfer does not need to start. If the SDP answerer accepts the SDP, then file transfer starts from the indicated octet (if a ’file-range’ attribute is present).

8.7. MSRP Usage

The file transfer service specified in this document uses "m=" lines in SDP to describe the unidirectional transfer of a file. Consequently, each MSRP session established following the procedures in Section 8.2 and Section 8.3 is only used to transfer a single file. So, senders MUST only use the dedicated MSRP session to send the file described in the SDP offer or answer. That is, senders MUST NOT send additional files over the same MSRP session.

File transfer may be accomplished using a new multimedia session established for the purpose. Alternatively, a file transfer may be conducted within an existing multimedia session, without regard for the media in use within that session. Of particular note, file transfer may be done within a multimedia session containing an MSRP session used for regular instant messaging. If file transfer is initiated within an existing multimedia session, the SDP offerer MUST NOT reuse an existing "m=" line that is still being used by MSRP (either regular MSRP for instant messaging or an ongoing file transfer). Rather, it MUST add an additional "m=" line or else reuse an "m=" line that is no longer being used.

Additionally, implementations according to this specification MUST include a single file in a single MSRP message. Notice that the MSRP specification defines "MSRP message" as a complete unit of MIME or text content, which can be split and delivered in more than one MSRP request; each of these portions of the complete message is called a "chunk". So, it is still valid to send a file in several chunks, but from the MSRP point of view, all the chunks together form an MSRP message: the Common Presence and Instant Messaging (CPIM) message that wraps the file. When chunking is used, it should be noticed that MSRP does not require to wait for a 200-class response for a chunk before sending the following one. Therefore, it is valid to send pipelined MSRP SEND requests containing chunks of the same MSRP message (the file). Section 9.1 contains an example of a file transfer using pipelined MSRP requests.

The MSRP specification [RFC4975] defines a 'max-size' SDP attribute. This attribute specifies the maximum number of octets of an MSRP message that the creator of the SDP is willing to receive (notice once more the definition of "MSRP message"). File receivers MAY add a 'max-size' attribute to the MSRP "m=" line that specifies the file, indicating the maximum number of octets of an MSRP message. File senders MUST NOT exceed the 'max-size' limit for any message sent in the resulting session.

In the absence of a 'file-range' attribute in the SDP, the MSRP file transfer MUST start with the first octet of the file and end with the last octet (i.e., the whole file is transferred). If a 'file-range' attribute is present in SDP, the file sender application MUST extract the indicated range of octets from the file (start and stop offset octets, both inclusive). Then the file sender application MAY wrap those octets in an appropriate wrapper. MSRP mandates implementations to implement the message/cpim wrapper [RFC3862]. Usage of a wrapper is negotiated in the SDP (see Section 8.6 in RFC 4975 [RFC4975]). Last, the file sender application delivers the content (e.g., the message/cpim body) to MSRP for transportation. MSRP will consider the delivered content as a whole message, and will start numbering bytes with the number 1.

Note that the default content disposition of MSRP bodies is 'render'. When MSRP is used to transfer files, the MSRP Content-Disposition header can also take the value 'attachment' as indicated in Section 7.

Once the file transfer is completed, the file sender SHOULD close the MSRP session and MUST behave according to the MSRP [RFC4975] procedures with respect to closing MSRP sessions. Note that MSRP session management is not related to TCP connection management. As a matter of fact, MSRP allows multiple MSRP sessions to share the same TCP connection.

8.8. Considerations about the 'file-icon' Attribute

This specification allows a file sender to include a small preview of an image file: an icon. A 'file-icon' attribute contains a Content-ID (CID) URL [RFC2392] pointing to an additional body that contains the actual icon. Since the icon is sent as a separate body along the SDP body, the file sender MUST wrap the SDP body and the icon bodies in a MIME multipart/related body. Therefore, implementations according to this specification MUST implement the multipart/related MIME type [RFC2387]. When creating a multipart/related MIME wrapper, the SDP body MUST be the root body, which according to RFC 2387 [RFC2387] is identified as the first body in the multipart/related MIME wrapper or explicitly identified by the 'start' parameter. According to RFC 2387 [RFC2387], the 'type' parameter MUST be present and point to the root body, i.e., the SDP body.

Assume that an endpoint behaving according to this specification tries to send a file to a remote endpoint that neither implements this specification nor implements multipart MIME bodies. The file sender sends an SDP offer that contains a multipart/related MIME body that includes an SDP body part and an icon body part. The file receiver, not supporting multipart MIME types, will reject the SDP offer via a higher protocol mechanism (e.g., SIP). In this case, it is RECOMMENDED that the file sender removes the icon body part, creates a single SDP body (i.e., without multipart MIME), and resends the SDP offer. This provides some backwards compatibility with file receives that do not implement this specification and increases the chances of getting the SDP accepted at the file receiver.

Since the icon is sent as part of the signaling, it is RECOMMENDED to keep the size of icons restricted to the minimum number of octets that provide significance.

9. Examples

9.1. Offerer Sends a File to the Answerer

This section shows an example flow for a file transfer scenario. The example assumes that SIP [RFC3261] is used to transport the SDP offer/answer exchange, although the SIP details are briefly shown for the sake of brevity.

Alice, the SDP offerer, wishes to send an image file to Bob (the answerer). Alice’s User Agent Client (UAC) creates a unidirectional SDP offer that contains the description of the file that she wants to send to Bob’s User Agent Server (UAS). The description also includes an icon representing the contents of the file to be transferred. The sequence flow is shown in Figure 7.

```
                   Alice’s UAC                 Bob’s UAS
                         |                        |
                         |(1) (SIP) INVITE        |
                         |----------------------->|
                         |(2) (SIP) 200 OK        |
                         |<-----------------------|
                         |(3) (SIP) ACK           |
                         |----------------------->|
                         |                        |
                         |(4) (MSRP) SEND (chunk) |
                         |----------------------->|
                         |(5) (MSRP) SEND (chunk) |
                         |----------------------->|
                         |(6) (MSRP) 200 OK       |
                         |<-----------------------|
                         |(7) (MSRP) 200 OK       |
                         |<-----------------------|
                         |                        |
                         |(8) (SIP) BYE           |
                         |----------------------->|
                         |(9) (SIP) 200 OK        |
                         |<-----------------------|
                         |                        |
                         |                        |
```

Figure 7: Flow diagram of an offerer sending a file to an answerer

F1: Alice constructs an SDP description of the file to be sent and attaches it to a SIP INVITE request addressed to Bob.

```
   INVITE sip:bob@example.com SIP/2.0
   To: Bob <sip:bob@example.com>
   From: Alice <sip:alice@example.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 1 INVITE
   Max-Forwards: 70
   Date: Sun, 21 May 2006 13:02:03 GMT
   Contact: <sip:alice@alicepc.example.com>
   Content-Type: multipart/related; type="application/sdp";
                 boundary="boundary71"
   Content-Length: [length]

   --boundary71
   Content-Type: application/sdp
   Content-Length: [length of SDP]

   v=0
   o=alice 2890844526 2890844526 IN IP4 alicepc.example.com
   s=
   c=IN IP4 alicepc.example.com
   t=0 0
   m=message 7654 TCP/MSRP *
   i=This is my latest picture
   a=sendonly
   a=accept-types:message/cpim
   a=accept-wrapped-types:*
   a=path:msrp://alicepc.example.com:7654/jshA7we;tcp
   a=file-selector:name:"My cool picture.jpg" type:image/jpeg
     size:4092 hash:sha-1:
     72:24:5F:E8:65:3D:DA:F3:71:36:2F:86:D4:71:91:3E:E4:A2:CE:2E
   a=file-transfer-id:Q6LMoGymJdh0IKIgD6wD0jkcfgva4xvE
   a=file-disposition:render
   a=file-date:creation:"Mon, 15 May 2006 15:01:31 +0300"
   a=file-icon:cid:id2@alicepc.example.com

   --boundary71
   Content-Type: image/jpeg
   Content-Transfer-Encoding: binary
   Content-ID: <id2@alicepc.example.com>
   Content-Length: [length of image]
   Content-Disposition: icon

   [...small preview icon of the file...]

   --boundary71--
```

Figure 8: INVITE request containing an SDP offer for file transfer

NOTE: The Content-Type header field and the ’file-selector’ attribute in the above figure are split in several lines for formatting purposes. Real implementations will encode it in a single line.

From now on we omit the SIP details for the sake of brevity.

F2: Bob receives the INVITE request, inspects the SDP offer and extracts the icon body, checks the creation date and file size, and decides to accept the file transfer. So Bob creates the following SDP answer:

```
v=0
o=bob 2890844656 2890844656 IN IP4 bobpc.example.com
s=
c=IN IP4 bobpc.example.com
t=0 0
m=message 8888 TCP/MSRP *
a=recvonly
a=accept-types:message/cpim
a=accept-wrapped-types:*
a=path:msrp://bobpc.example.com:8888/9di4ea;tcp
a=file-selector:name:"My cool picture.jpg" type:image/jpeg
  size:4092 hash:sha-1:
  72:24:5F:E8:65:3D:DA:F3:71:36:2F:86:D4:71:91:3E:E4:A2:CE:2E
a=file-transfer-id:Q6LMoGymJdh0IKIgD6wD0jkcfgva4xvE
```

Figure 9: SDP answer accepting the SDP offer for file transfer

NOTE: The ’file-selector’ attribute in the above figure is split in three lines for formatting purposes. Real implementations will encode it in a single line.

F4: Alice opens a TCP connection to Bob and creates an MSRP SEND request. This SEND request contains the first chunk of the file.

```
MSRP d93kswow SEND
To-Path: msrp://bobpc.example.com:8888/9di4ea;tcp
From-Path: msrp://alicepc.example.com:7654/iau39;tcp
Message-ID: 12339sdqwer
Byte-Range: 1-2048/4385
Content-Type: message/cpim

To: Bob <sip:bob@example.com>
From: Alice <sip:alice@example.com>
DateTime: 2006-05-15T15:02:31-03:00
Content-Disposition: render; filename="My cool picture.jpg";
                   creation-date="Mon, 15 May 2006 15:01:31 +0300";
                   size=4092
Content-Type: image/jpeg

... first set of bytes of the JPEG image ...
-------d93kswow+
```

Figure 10: MSRP SEND request containing the first chunk of actual file

F5: Alice sends the second and last chunk. Note that MSRP allows to send pipelined chunks, so there is no need to wait for the 200 (OK) response from the previous chunk.

```
MSRP op2nc9a SEND
To-Path: msrp://bobpc.example.com:8888/9di4ea;tcp
From-Path: msrp://alicepc.example.com:7654/iau39;tcp
Message-ID: 12339sdqwer
Byte-Range: 2049-4385/4385
Content-Type: message/cpim

... second set of bytes of the JPEG image ...
-------op2nc9a$
```

Figure 11: MSRP SEND request containing the second chunk of actual file

F6: Bob acknowledges the reception of the first chunk.

```
MSRP d93kswow 200 OK
To-Path: msrp://alicepc.example.com:7654/iau39;tcp
From-Path: msrp://bobpc.example.com:8888/9di4ea;tcp
Byte-Range: 1-2048/4385
-------d93kswow$
```

Figure 12: MSRP 200 OK response

F7: Bob acknowledges the reception of the second chunk.

```
MSRP op2nc9a 200 OK
To-Path: msrp://alicepc.example.com:7654/iau39;tcp
From-Path: msrp://bobpc.example.com:8888/9di4ea;tcp
Byte-Range: 2049-4385/4385
-------op2nc9a$
```

Figure 13: MSRP 200 OK response

F8: Alice terminates the SIP session by sending a SIP BYE request.

F9: Bob acknowledges the reception of the BYE request and sends a 200 (OK) response.

9.2. Offerer Requests a File from the Answerer and Second File Transfer

In this example, Alice, the SDP offerer, first wishes to fetch a file from Bob, the SDP answerer. Alice knows that Bob has a specific file she wants to download. She has learned the hash of the file by some out-of-band mechanism. The hash selector is enough to produce a file selector that points to the specific file. So, Alice creates an SDP offer that contains the file descriptor. Bob accepts the file transfer and sends the file to Alice. When Alice has completely received Bob’s file, she intends to send a new image file to Bob. Therefore, Alice reuses the existing SDP media line with different attributes and updates the description of the new file she wants to send to Bob’s User Agent Server (UAS). In particular, Alice creates a new file transfer identifier since this is a new file transfer operation. Figure 14 shows the sequence flow.

```
                   Alice's UAC                 Bob's UAS
                         |                        |
                         |(1) (SIP) INVITE        |
                         |----------------------->|
                         |(2) (SIP) 200 OK        |
                         |<-----------------------|
                         |(3) (SIP) ACK           |
                         |----------------------->|
                         |                        |
                         |(4) (MSRP) SEND (file)  |
                         |<-----------------------|
                         |(5) (MSRP) 200 OK       |
                         |----------------------->|
                         |                        |
                         |(6) (SIP) INVITE        |
                         |----------------------->|
                         |(7) (SIP) 200 OK        |
                         |<-----------------------|
                         |(8) (SIP) ACK           |
                         |----------------------->|
                         |                        |
                         |(9) (MSRP) SEND (file)  |
                         |----------------------->|
                         |(10) (MSRP) 200 OK      |
                         |<-----------------------|
                         |                        |
                         |(11) (SIP) BYE          |
                         |<-----------------------|
                         |(12) (SIP) 200 OK       |
                         |----------------------->|
                         |                        |
                         |                        |
```

Figure 14: Flow diagram of an offerer requesting a file from the answerer and then sending a file to the answer

F1: Alice constructs an SDP description of the file she wants to receive and attaches the SDP offer to a SIP INVITE request addressed to Bob.

```
INVITE sip:bob@example.com SIP/2.0
To: Bob <sip:bob@example.com>
From: Alice <sip:alice@example.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 1 INVITE
Max-Forwards: 70
Date: Sun, 21 May 2006 13:02:03 GMT
Contact: <sip:alice@alicepc.example.com>
Content-Type: application/sdp
Content-Length: [length of SDP]

v=0
o=alice 2890844526 2890844526 IN IP4 alicepc.example.com
s=
c=IN IP4 alicepc.example.com
t=0 0
m=message 7654 TCP/MSRP *
a=recvonly
a=accept-types:message/cpim
a=accept-wrapped-types:*
a=path:msrp://alicepc.example.com:7654/jshA7we;tcp
a=file-selector:hash:sha-1:
  72:24:5F:E8:65:3D:DA:F3:71:36:2F:86:D4:71:91:3E:E4:A2:CE:2E
a=file-transfer-id:aCQYuBRVoUPGVsFZkCK98vzcX2FXDIk2
```

Figure 15: INVITE request containing an SDP offer for file transfer

NOTE: The ’file-selector’ attribute in the above figure is split in two lines for formatting purposes. Real implementations will encode it in a single line.

From now on we omit the SIP details for the sake of brevity.

F2: Bob receives the INVITE request, inspects the SDP offer, computes the file descriptor, and finds a local file whose hash equals the one indicated in the SDP. Bob accepts the file transfer and creates an SDP answer as follows:

```
v=0
o=bob 2890844656 2890855439 IN IP4 bobpc.example.com
s=
c=IN IP4 bobpc.example.com
t=0 0
m=message 8888 TCP/MSRP *
a=sendonly
a=accept-types:message/cpim
a=accept-wrapped-types:*
a=path:msrp://bobpc.example.com:8888/9di4ea;tcp
a=file-selector:type:image/jpeg hash:sha-1:
  72:24:5F:E8:65:3D:DA:F3:71:36:2F:86:D4:71:91:3E:E4:A2:CE:2E
a=file-transfer-id:aCQYuBRVoUPGVsFZkCK98vzcX2FXDIk2
```

Figure 16: SDP answer accepting the SDP offer for file transfer

NOTE: The ’file-selector’ attribute in the above figure is split in two lines for formatting purposes. Real implementations will encode it in a single line.

F4: Alice opens a TCP connection to Bob. Bob then creates an MSRP SEND request that contains the file.

```
MSRP d93kswow SEND
To-Path: msrp://alicepc.example.com:7654/jshA7we;tcp
From-Path: msrp://bobpc.example.com:8888/9di4ea;tcp
Message-ID: 12339sdqwer
Byte-Range: 1-2027/2027
Content-Type: message/cpim

To: Bob <sip:bob@example.com>
From: Alice <sip:alice@example.com>
DateTime: 2006-05-15T15:02:31-03:00
Content-Disposition: render; filename="My cool photo.jpg";
               creation-date="Mon, 15 May 2006 15:01:31 +0300";
               modification-date="Mon, 15 May 2006 16:04:53 +0300";
               read-date="Mon, 16 May 2006 09:12:27 +0300";
               size=1931
Content-Type: image/jpeg

...binary JPEG image...
-------d93kswow$
```

Figure 17: MSRP SEND request containing the actual file

F5: Alice acknowledges the reception of the SEND request.

```
MSRP d93kswow 200 OK
To-Path: msrp://bobpc.example.com:8888/9di4ea;tcp
From-Path: msrp://alicepc.example.com:7654/jshA7we;tcp
Byte-Range: 1-2027/2027
-------d93kswow$
```

Figure 18: MSRP 200 OK response

F6: Alice reuses the existing SDP media line inserting the description of the file to be sent and attaches it to a SIP re-INVITE request addressed to Bob. Alice reuses the TCP port number for the MSRP stream, but changes the MSRP session and the ’file-transfer-id’ value according to this specification.

```
   INVITE sip:bob@example.com SIP/2.0
   To: Bob <sip:bob@example.com>;tag=1928323431
   From: Alice <sip:alice@example.com>;tag=1928301774
   Call-ID: a84b4c76e66710
   CSeq: 2 INVITE
   Max-Forwards: 70
   Date: Sun, 21 May 2006 13:02:33 GMT
   Contact: <sip:alice@alicepc.example.com>
   Content-Type: multipart/related; type="application/sdp";
                 boundary="boundary71"
   Content-Length: [length of multipart]

   --boundary71
   Content-Type: application/sdp
   Content-Length: [length of SDP]

   v=0
   o=alice 2890844526 2890844527 IN IP4 alicepc.example.com
   s=
   c=IN IP4 alicepc.example.com
   t=0 0
   m=message 7654 TCP/MSRP *
   i=This is my latest picture
   a=sendonly
   a=accept-types:message/cpim
   a=accept-wrapped-types:*
   a=path:msrp://alicepc.example.com:7654/iau39;tcp
   a=file-selector:name:"sunset.jpg" type:image/jpeg
     size:4096 hash:sha-1:
     58:23:1F:E8:65:3B:BC:F3:71:36:2F:86:D4:71:91:3E:E4:B1:DF:2F
   a=file-transfer-id:ZVE8MfI9mhAdZ8GyiNMzNN5dpqgzQlCO
   a=file-disposition:render
   a=file-date:creation:"Sun, 21 May 2006 13:02:15 +0300"
   a=file-icon:cid:id3@alicepc.example.com

   --boundary71
   Content-Type: image/jpeg
   Content-Transfer-Encoding: binary
   Content-ID: <id3@alicepc.example.com>
   Content-Length: [length of image]
   Content-Disposition: icon

   [..small preview icon...]

   --boundary71--
```

Figure 19: Reuse of the SDP in a second file transfer

NOTE: The Content-Type header field and the 'file-selector' attribute in the above figure are split in several lines for formatting purposes. Real implementations will encode it in a single line.

F7: Bob receives the re-INVITE request, inspects the SDP offer and extracts the icon body, checks the creation date and the file size, and decides to accept the file transfer. So Bob creates an SDP answer where he reuses the same TCP port number, but changes his MSRP session, according to the procedures of this specification.

```
v=0
o=bob 2890844656 2890855440 IN IP4 bobpc.example.com
s=
c=IN IP4 bobpc.example.com
t=0 0
m=message 8888 TCP/MSRP *
a=recvonly
a=accept-types:message/cpim
a=accept-wrapped-types:*
a=path:msrp://bobpc.example.com:8888/eh10dsk;tcp
a=file-selector:name:"sunset.jpg" type:image/jpeg
  size:4096 hash:sha-1:
  58:23:1F:E8:65:3B:BC:F3:71:36:2F:86:D4:71:91:3E:E4:B1:DF:2F
a=file-transfer-id:ZVE8MfI9mhAdZ8GyiNMzNN5dpqgzQlCO
a=file-disposition:render
```

Figure 20: SDP answer accepting the SDP offer for file transfer

NOTE: The 'file-selector' attribute in the above figure is split in three lines for formatting purposes. Real implementations will encode it in a single line.

F9: If a TCP connection towards Bob is already open, Alice reuses that TCP connection to send an MSRP SEND request that contains the file.

```
   MSRP d95ksxox SEND
   To-Path: msrp://bobpc.example.com:8888/eh10dsk;tcp
   From-Path: msrp://alicepc.example.com:7654/iau39;tcp
   Message-ID: 13449sdqwer
   Byte-Range: 1-2027/2027
   Content-Type: message/cpim

   To: Bob <sip:bob@example.com>
   From: Alice <sip:alice@example.com>
   DateTime: 2006-05-21T13:02:15-03:00
   Content-Disposition: render; filename="Sunset.jpg";
                        creation-date="Sun, 21 May 2006 13:02:15 -0300";
                        size=1931
   Content-Type: image/jpeg

   ...binary JPEG image...
   -------d95ksxox+
```

Figure 21: MSRP SEND request containing the actual file

F10: Bob acknowledges the reception of the SEND request.

```
   MSRP d95ksxox 200 OK
   To-Path: msrp://alicepc.example.com:7654/iau39;tcp
   From-Path: msrp://bobpc.example.com:8888/eh10dsk;tcp
   Byte-Range: 1-2027/2027
   -------d95ksxox$
```

Figure 22: MSRP 200 OK response

F11: Then Bob terminates the SIP session by sending a SIP BYE request.

F12: Alice acknowledges the reception of the BYE request and sends a 200 (OK) response.

9.3. Example of a Capability Indication

Alice sends an OPTIONS request to Bob (this request does not contain SDP). Bob answers with a 200 (OK) response that contain the SDP shown in Figure 24. The SDP indicates support for CPIM messages that can contain other MIME types. The maximum MSRP message size that the endpoint can receive is 20000 octets. The presence of the ’file-selector’ attribute indicates support for the file transfer offer/answer mechanism.

```
                   Alice’s UAC                 Bob’s UAS
                         |                        |
                         |(1) (SIP) OPTIONS       |
                         |----------------------->|
                         |(2) (SIP) 200 OK        |
                         |          with SDP      |
                         |<-----------------------|
                         |                        |
                         |                        |
```

Figure 23: Flow diagram of a capability request

```
   v=0
   o=bob 2890844656 2890855439 IN IP4 bobpc.example.com
   s=-
   c=IN IP4 bobpc.example.com
   t=0 0
   m=message 0 TCP/MSRP *
   a=accept-types:message/cpim
   a=accept-wrapped-types:*
   a=max-size:20000
   a=file-selector
```

Figure 24: SDP of the 200 (OK) response to an OPTIONS request

10. Security Considerations

The SDP attributes defined in this specification identify a file to be transferred between two endpoints. An endpoint can offer to send the file to the other endpoint or request to receive the file from the other endpoint. In the former case, an attacker modifying those SDP attributes could cheat the receiver making it think that the file to be transferred was a different one. In the latter case, the attacker could make the sender send a different file than the one requested by the receiver. Consequently, it is RECOMMENDED that integrity protection be applied to the SDP session descriptions carrying the attributes specified in this specification. Additionally, it is RECOMMENDED that senders verify the properties of the file against the selectors that describe it.

The descriptions of the files being transferred between endpoints may reveal information the endpoints may consider confidential. Therefore, it is RECOMMENDED that SDP session descriptions carrying the attributes specified in this specification are encrypted.

TLS and S/MIME are the natural choices to provide offer/answer exchanges with integrity protection and confidentiality.

When an SDP offer contains the description of a file to be sent or received, the SDP answerer MUST first authenticate the SDP offerer and then it MUST authorize the file transfer operation, typically according to a local policy. Typically, these functions are integrated in the high-level protocol that carries SDP (e.g., SIP), and in the file transfer protocol (e.g., MSRP). If SIP [RFC3261] and MSRP [RFC4975] are used, the standard mechanisms for user authentication and authorization are sufficient.

It is possible that a malicious or misbehaving implementation tries to exhaust the resources of the remote endpoint, e.g., the internal memory or the file system, by sending very large files. To protect from this attack, an SDP answer SHOULD first verify the identity of the SDP offerer, and perhaps only accept file transfers from trusted sources. Mechanisms to verify the identity of the file sender depend on the high-level protocol that carries the SDP, for example, SIP [RFC3261] and MSRP [RFC4975].

It is also RECOMMENDED that implementations take measures to avoid attacks on resource exhaustion, for example, by limiting the size of received files, verifying that there is enough space in the file system to store the file prior to its reception, or limiting the number of simultaneous file transfers.

File receivers MUST also sanitize all input, such as the local file name, prior to making calls to the local file system to store a file. This is to prevent the existence of meaningful characters to the local operating system that could damage it.

Once a file has been transferred, the file receiver must take care of it. Typically, file transfer is a commonly used mechanism for transmitting computer virus, spyware, and other types of malware. File receivers should apply all possible security technologies (e.g., anti-virus, anti-spyware) to mitigate the risk of damage at their host.

11. IANA Considerations

IANA has registered a number of SDP attributes according to the following.

11.1. Registration of New SDP Attributes

IANA has registered a number of media-level-only attributes in the Session Description Protocol Parameters registry [IANA]. The registration data, according to RFC 4566 [RFC4566], follows.

11.1.1. Registration of the file-selector Attribute

Contact: Miguel Garcia <miguel.a.garcia@ericsson.com>

Phone: +34 91 339 1000

Attribute name: file-selector

Long-form attribute name: File Selector

Type of attribute: media level only
This attribute is subject to the charset attribute

Description: This attribute unambiguously identifies a file by indicating a combination of the 4-tuple composed of the name, size, type, and hash of the file.

Specification: RFC 5547

11.1.2. Registration of the file-transfer-id Attribute

Contact: Miguel Garcia <miguel.a.garcia@ericsson.com>

Phone: +34 91 339 1000

Attribute name: file-transfer-id

Long-form attribute name: File Transfer Identifier

Type of attribute: media level only
This attribute is subject to the charset attribute

Description: This attribute contains a unique identifier of the file transfer operation within the session.

Specification: RFC 5547

11.1.3. Registration of the file-disposition Attribute

Contact: Miguel Garcia <miguel.a.garcia@ericsson.com>

Phone: +34 91 339 1000

Attribute name: file-disposition

Long-form attribute name: File Disposition

Type of attribute: media level only

This attribute is not subject to the charset attribute

Description: This attribute provides a suggestion to the other endpoint about the intended disposition of the file.

Specification: RFC 5547

11.1.4. Registration of the file-date Attribute

Contact: Miguel Garcia <miguel.a.garcia@ericsson.com>

Phone: +34 91 339 1000

Attribute name: file-date

Long-form attribute name:

Type of attribute: media level only
This attribute is not subject to the charset attribute

Description: This attribute indicates the dates on which the file was created, modified, or last read.

Specification: RFC 5547

11.1.5. Registration of the file-icon Attribute

Contact: Miguel Garcia <miguel.a.garcia@ericsson.com>

Phone: +34 91 339 1000

Attribute name: file-icon

Long-form attribute name: File Icon

Type of attribute: media level only
This attribute is not subject to the charset attribute

Description: For image files, this attribute contains a pointer to a body that includes a small preview icon representing the contents of the file to be transferred.

Specification: RFC 5547

11.1.6. Registration of the file-range Attribute

Contact: Miguel Garcia <miguel.a.garcia@ericsson.com>

Phone: +34 91 339 1000

Attribute name: file-range

Long-form attribute name: File Range

Type of attribute: media level only
This attribute is not subject to the charset attribute

Description: This attribute contains the range of transferred octets of the file.

Specification: RFC 5547

12. Acknowledgments

The authors would like to thank Mats Stille, Nancy Greene, Adamu Haruna, and Arto Leppisaari for discussing initial concepts described in this memo. Thanks to Pekka Kuure for reviewing initial versions of this document and providing helpful comments. Joerg Ott, Jiwey Wang, Amitkumar Goel, Sudha Vs, Dan Wing, Juuso Lehtinen, Remi Denis-Courmont, Colin Perkins, Sudhakar An, Peter Saint-Andre, Jonathan Rosenberg, Eric Rescorla, Vikram Chhibber, Ben Campbell, Richard Barnes, and Chris Newman discussed and provided comments and improvements to this document.

13. References

13.1. Normative References

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

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

[RFC2183] Troost, R., Dorner, S., and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, August 1998.

[RFC2392] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3851] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

[RFC3862] Klyne, G. and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format", RFC 3862, August 2004.

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

[RFC4975] Campbell, B., Mahy, R., and C. Jennings, "The Message Session Relay Protocol (MSRP)", RFC 4975, September 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

13.2. Informative References

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

[RFC4028] Donovan, S. and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)", RFC 4028, April 2005.

[RFC4483] Burger, E., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", RFC 4483, May 2006.

[RFC4976] Jennings, C., Mahy, R., and A. Roach, "Relay Extensions for the Message Sessions Relay Protocol (MSRP)", RFC 4976, September 2007.

[IANA] IANA, "Internet Assigned Numbers Authority", <http://www.iana.org>.

[FLUTE-REV] Luby, M., Lehtonen, R., Roca, V., and T. Paila, "FLUTE - File Delivery over Unidirectional Transport", Work in Progress, September 2008.

Appendix A. Alternatives Considered

The requirements are related to the description and negotiation of the session, not to the actual file transfer mechanism. Thus, it is natural that in order to meet them it is enough to define attribute extensions and usage conventions to SDP, while MSRP itself needs no extensions and can be used as it is. This is effectively the approach taken in this specification. Another goal has been to specify the SDP extensions in such a way that a regular MSRP endpoint that does not support them could still in some cases act as an endpoint in a file transfer session, albeit with a somewhat reduced functionality.

In some ways, the aim of this specification is similar to the aim of content indirection mechanism in the Session Initiation Protocol (SIP) [RFC4483]. Both mechanisms allow a user agent to decide whether or not to download a file based on information about the file. However, there are some differences. With content indirection, it is not possible for the other endpoint to explicitly accept or reject the file transfer. Also, it is not possible for an endpoint to request a file from another endpoint. Furthermore, content indirection is not tied to the context of a media session, which is sometimes a desirable property. Finally, content indirection typically requires some server infrastructure, which may not always be available. It is possible to use content indirection directly between the endpoints too, but in that case there is no definition for how it works for endpoints behind NATs. The level of requirements in implementations decides which solution meets the requirements.

Based on the argumentation above, this document defines the SDP attribute extensions and usage conventions needed for meeting the requirements on file transfer services with the SDP offer/answer model, using MSRP as the transfer protocol within the session.

> In principle, it is possible to use the SDP extensions defined here and replace MSRP with any other similar protocol that can carry MIME objects. This kind of specification can be written as a separate document if the need arises. Essentially, such a protocol should be able to be negotiated on an SDP offer/answer exchange (RFC 3264 [RFC3264]), be able to describe the file to be transferred in SDP offer/answer exchange, be able to carry MIME objects between two endpoints, and use a reliable transport protocol (e.g., TCP).

This specification defines a set of SDP attributes that describe a file to be transferred between two endpoints. The information needed to describe a file could be potentially encoded in a few different ways. The MMUSIC working group considered a few alternative approaches before deciding to use the encoding described in Section 6. In particular, the working group looked at the MIME 'external-body' type and the use of a single SDP attribute or parameter.

A MIME 'external-body' could potentially be used to describe the file to be transferred. In fact, many of the SDP parameters this specification defines are also supported by 'external-body' body parts. The MMUSIC working group decided not to use 'external-body' body parts because a number of existing offer/answer implementations do not support multipart bodies.

The information carried in the SDP attributes defined in Section 6 could potentially be encoded in a single SDP attribute. The MMUSIC working group decided not to follow this approach because it is expected that implementations support only a subset of the parameters defined in Section 6. Those implementations will be able to use regular SDP rules in order to ignore non-supported SDP parameters. If all the information was encoded in a single SDP attribute, those rules, which relate to backwards compatibility, would need to be redefined specifically for that parameter.

Authors' Addresses

Miguel A. Garcia-Martin
Ericsson
Calle Via de los Poblados 13
Madrid, ES 28033
Spain

EMail: miguel.a.garcia@ericsson.com

Markus Isomaki
Nokia
Keilalahdentie 2-4
Espoo 02150
Finland

EMail: markus.isomaki@nokia.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Salvatore.Loreto@ericsson.com

Paul H. Kyzivat
Cisco Systems
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

EMail: pkyzivat@cisco.com