SIMPLE WG
Internet-Draft
Expires: August 9, 2004

C. Jennings
R. Mahy
J. Garg
Cisco Systems, Inc.

February 9, 2004

SIMPLE Instant Messaging Sessions (SIMS)
draft-jennings-simple-sims-00.txt

Abstract

This document defines a protocol for conveying binary MIME content in near-real time, peer-to-peer or through one or more relays, with the opportunity for store and forward. SIMS (SIMPLE Instant Messaging Sessions) can be used as a standalone protocol, or in conjunction with a rendezvous or session setup protocol such as SIP.

While SIMS was originally envisioned as an alternative to the Media Session Relay Protocol (MSRP), one section of this document describes how these ideas could be applied as MSRP extensions for features such as chunking, relay connection multiplexing, and prevention of head-of-line blocking.

Table of Contents

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

Below we list several definitions important to SIMS:

'SIMS node:' A host that implements the SIMS protocols as a Client or a Relay

'SIMS Client:' A SIMS role which is the initial sender or final target of messages and delivery status.

'SIMS Relay:' A SIMS role which forwards messages and delivery status and may provide policy enforcement. Relays MAY fragment and reassemble portions of messages.

'Message-Taker:' A SIMS Client which persistently stores messages on behalf of specific users or resources

'message:' arbitrary MIME content which one client wishes to send to another. For the purposes of this specification, a complete MIME body as opposed to a portion of a complete message.

'message fragment:' a portion of a complete message carried in a message/byteranges MIME type.

'message:' binary MIME content of an arbitrary type. Each message has a unique message-id. In SIMS, messages may be broken up into pieces and sent in separate CHUNK requests.

'parcel:' a SIMS request or response. CHUNK request parcels typically contain a portion of a complete message.

'end-to-end:' delivery of data from the initiating client to the final target client

'hop:' delivery of data between one SIMS node and an adjacent node.

'transaction:' a request and response as seen from a single SIMS node. Each transaction has a locally significant transaction identifier.

2. Introduction and Requirements

The IETF SIMPLE Working Group has identified a number of scenarios where using a separate protocol for bulk messaging is desirable. In particular, the SIMPLE WG will use this facility to handle a sequence of messages as a session of media initiated using SIP [2], just like any other media type. The SIMPLE community has investigated many options for sessions of messages (Jabber [27], SIP [28], IMTP [29], and AMSX [30]), the most recent of these called MSRP [19].

While the wireless community has responded favorably to MSRP for point-to-point usage, the authors feel that MSRP does not sufficiently address the relay requirements of the Enterprise and Consumer IM community. Indeed, the most recent version of MSRP has completely removed any normative discussion about building relays at all. This proposal attempts to capture the benefits of MSRP (especially peer-to-peer operation) and also address these additional requirements. SIMS instead borrows heavily from the relay capabilities of IMTP. Section 4 discusses how the concepts in SIMS could be implemented as MSRP extensions.

The rest of this document describes SIMS as a separate protocol for conveying arbitrary MIME [3] content in near-real time through zero or more relays, with the opportunity for store and forward. SIMS (SIMPLE Instant Messaging Sessions) can be used as a standalone protocol, or in conjunction with a rendezvous or session setup protocol such as SIP. As with MSRP, all SIMS traffic is sent over reliable, congestion-safe transports.

SIMS was designed to allow SIMS clients to communicate directly, or through an arbitrary number of relays. Each client is responsible for identifying any relays acting on its behalf and providing appropriate credentials.

The Goals of SIMS are listed below:

- convey arbitrary binary MIME data
- operate as a standalone protocol or as a session media protocol
- support client to client operation (no servers required)
- operate through an arbitrary number of relays for policy enforcement
- allow each client to control which relays are traversed on its behalf

- o prevent unsolicited messages (spam), "open relays", and denial of service amplification

- o allow relays to use one or a small number of TCP or TLS [4] connections to carry messages for multiple sessions, recipients, and senders

- o allow large messages to be sent over a slow connection without causing head-of-line blocking problems

- o allow transmission of a large message to be interrupted and resumed in place when network connectivity is lost and later reestablished

- o offer end-to-end notification of message receipt

- o provide notification of message storage (desirable)

- o easy to implement

- o allow relays to delete state after a short amount of time

3. Protocol Overview

SIMS defines the concept of clients and relays. Clients send messages to relays and other clients. Relays forward messages and message delivery status to clients and other relays. Clients which can open TCP connections to each other without intervening policy restrictions, can communicate directly with each other. Clients who are behind a firewall or who need to use an intermediary for policy reasons can use the services of a relay. Each client is responsible for enlisting the assistance of one or more relays for its half of the communication.

SIMS also defines the special role of a Message-Taker, which is a client that can receive messages and store them persistently on behalf of a user. Note that these roles can be co-resident.

Clients which use a relay operate by first opening a connection with a relay and authenticating. When clients wish to send a short message, they send a CHUNK request with the entire contents of the message.

```
 CHUNK sims:bob.example.net SIMS/1.0
 Via: TCP/SIMS-TLS/1.0 alice.example.org;received=10.1.1.1:9000
       ;branch=3847873847083047
 Message-Id: 12313513
```

```
    Route: <sims:example.org:9000;transport=tls+tcp>,
           <sims:magic-cookie@example.net:9000;transport=tls+tcp>
    Content-Type: text/plain

    Hi Bob, I'm about to send you "The Lord of the Rings".
```

Each hop (relay or recipient client) that receives a CHUNK request acknowledges receipt of the request before forwarding. For larger messages, each CHUNK request may contain only a portion of the complete message. To avoid confusion and ambiguity, each request or response is called a "parcel". When Alice sends Bob a 4GB file called "The Lord of the Rings.mpeg", she will sends several CHUNK requests (parcels) each with one part of the complete message. Relays can repack parcels en-route. As individual parts of the complete message arrive at the final destination client, the receiving client sends INFORM requests indicating delivery status.

```
          Typical flow with no relays
          (peer-to-peer client communication).

          Alice                      Bob

            |                         |
            |      CHUNK              |   "Hey dude! I think your IM
            |------------------------>|    client is spewing chunks!"
            |                         |
            |      200 OK             |
            |<------------------------|
            |      INFORM             |
            |<------------------------|    Message displayed
            |      200 OK             |
            |------------------------>|
            |                         |
```

When a client uses a relay, it first opens a TLS connection to its first relay and authenticates using an AUTH request which can contain Digest Authentication credentials. In a successful AUTH response, the relay provides a SIMS URI associated with the path to the client that the client can give to other clients for end-to-end message delivery.

SIMS nodes can send individual portions of a complete message in multiple CHUNK requests. Each parcel uses the message/byteranges MIME type defined in RFC 2616 [5] to correlate that part to the complete message. As each CHUNK request is received, the next hop acknowledges the request. As relays receive parcels they can reassemble or re-fragment them as long as each chunk is sent in order. Once a chunk or complete message arrives at the destination

client, the destination sends an INFORM request indicating that a chunk arrived end-to-end. This request travels back along the reverse path of the CHUNK request. Unlike the CHUNK request which is acknowledged along every hop, only the sender of the INFORM request responds to an INFORM. Relays then forward the INFORM response back to the recipient of the original CHUNK.

```
                    Typical flow involving two relays

Alice              a.example.org       b.example.net             Bob
  |                      |                    |                     |
  |                      |                    |                     |
  |--- AUTH ----------->|                    |<-- AUTH ------------|
  |<-- 401 Auth---------|                    |--- 401 Auth-------->|
  |--- AUTH ----------->|                    |<-- AUTH ------------|
  |<-- 200 OK-----------|                    |--- 200 OK---------->|
  |                      |                    |                     |
        ....              time passes            ....
  |                      |                    |                     |
  |--- CHUNK 0-3 ------>|                    |                     |
  |<-- 200 OK ----------|                    |  (slow link)        |
  |--- CHUNK 4-7 ------>|--- CHUNK 0-5 ----->|                     |
  |<-- 200 OK ----------|<-- 200 OK ---------|--- CHUNK 0-3 ------>|
  |--- CHUNK 8-10 ----->|--- CHUNK 6-10 ---->|               ....>|
  |<-- 200 OK ----------|<-- 200 OK ---------|                 ..>|
  |                      |                    |<-- 200 OK ----------|
  |                      |                    |<-- INFORM 0-3 ------|
  |                      |<-- INFORM 0-3 -----|--- CHUNK 4-7 ------>|
  |<-- INFORM 0-3 ------|                    |                ...>|
  |--- 200 OK --------->|                    |                 ..>|
  |                      |--- 200 OK -------->|                     |
  |                      |                    |--- 200 OK --------->|
  |                      |                    |<-- INFORM 4-7 ----->|
  |                      |<-- INFORM 4-7 -----|--- CHUNK 8-10 ----->|
  |<-- INFORM 4-7 ------|                    |                 ..>|
  |--- 200 OK --------->|                    |<-- 200 OK ----------|
  |                      |<-- INFORM done-----|<-- INFORM done -----|
  |<-- INFORM done -----|--- 200 OK -------->|                     |
  |--- 200 OK --------->|                    |--- 200 OK --------->|
  |                      |--- 200 OK -------->|                     |
  |                      |                    |--- 200 OK --------->|
  |                      |                    |                     |
```

Relays only keep transaction state for a short period of time for each chunk. Delivery of each hope should take no more than 32 seconds after the last byte of data is sent. Clients applications define their own implementation-dependent timers for end-to-end message delivery.

In some cases the end user node may not have its own client or that client or node may be unavailable. In this case, a message-taker can take receipt of the message or fragment and deliver an INFORM back to the sender indicating that the message or fragment was successfully stored.

For client to client communication, the sender of a message typically opens a new TCP connection if one is needed. Relays reuse existing connections first, but can open new connections (typically to another relay) to deliver a CHUNK request. INFORM requests are only delivered over an existing connection.

4. Building SIMS as extensions to MSRP

While SIMS is described as a standalone protocol in the bulk of this document, this proposal could be applied to MSRP while preserving the energy the SIMPLE working group has invested in discussing MSRP.

4.1 Changes Required to the core MSRP spec

If a SIMS-inspired relay extension to MSRP is implemented, a number of changes need to be made to the core MSRP specification. Specifically, many changes are needed when the requirements of multiplexing and no head-of-line blocking are introduced.

The most significant of these deals with the elimination of the VISIT command and with connection oriented media. The authors propose that the offerer initiate any needed TCP or TLS connections and immediately use a SEND to send the first message or portion of a message.

SEND requests will require a new mandatory header field which correlates a message or chunk with the session responsible for that session. Likewise for some conferencing applications, it may be necessary to include the identity of the original sender of the request.

Instead of relying on port numbers, connection identifiers or connection handles would be needed in an MSRP URI so that a client can provide enough information for a relay to forward over an existing TCP or TLS connection.

To prevent head-of-line blocking, it is necessary for clients to be able to stop sending large messages midstream and chunk messages using the message/byteranges MIME type. (Since using multiple connections as described in section 5.1 of MSRP is undesirable in a relay environment). Portions of messages conveyed with SEND need a corresponding message identifier to correlate them. Similarly the

length value in the start line of each MSRP request should be replaced with a MIME boundary. The end of that boundary marker would signal the end of a request.

TLS and TCP on the same port with no STARTTLS command would be an unacceptable implementation burden for relay providers. Either two port numbers of a STARTTLS command should be introduced. Further, it is unacceptable and of questionable usefulness to switch from TCP to TLS at any time other than immediately at connection establishment.

4.2 MSRP extensions for using relays

Other features of SIMS could be introduced as an extension to MSRP or even as a separate protocol. It is desirable for example to add an optional Route header in MSRP which clients can use to direct their request through specific relays. The "hop" SDP attribute could be added to convey this information in SIP offers and answers.

Because introducing relays which can repack messages changes the way chunks are acknowledged, an end-to-end message delivery mechanism such as INFORM would be needed.

A mechanism to authenticate with relays to prevent open relay and DoS amplification is needed. A mechanism similar to AUTH can be added.

5. SIMS parcel structure

5.1 Basic parcel organization

SIMS defines the concept of a parcel, which is analogous to a "message" (a request or response) in HTTP, SIP, and RTSP [20]. In SIMS, a message is a complete MIME document with a single Message-ID. Since messages can be arbitrarily large, a message can be sent in one or more piece, each piece carried in its own parcel.

SIMS parcels can be either requests or responses. Like HTTP messages, SIMS Parcels consist of a start line, headers and an optional body. Requests contain a method name and the Request-URI in the start line. Responses contain a response code and response phrase in the start line.

The Request-URI in a SIMS request is typically a SIMS URI. A SIMS URI takes the form sims:userinfo@hostport;param=value. For example: sims:r13-9dELHJ@server.example.com:9000;transport=tls+tcp

SIMS defines three types of requests, the CHUNK request, the INFORM request, and the AUTH request. The semantics of each of these methods is described in turn.

The CHUNK method is used to send a chunk of a message. CHUNK requests contain a Message-ID used to associate all the chunks of a message. In addition, an optional Thread-ID and Call-ID can correlate the chunk with a specific thread or session respectively. CHUNK requests are sent one hop at a time. Once a CHUNK request is received by a hop, that hop immediately generates a response parcel. This kind of request and response is called a per-hop transaction. CHUNK requests are per-hop for two reasons: 1) SIMS relays may pack or rechunk any message in a different set of chunks as long as they preserve ordering, and 2) since the amount of bandwidth available between each hop may be radically different, there is no way to set a sensible timer for the success or failure of a chunk delivered end-to-end.

```
CHUNK->
<- 200 OK
         CHUNK ->
         <- 200 OK
```

Chunks of messages are managed using the message/byterangesmessage/ byteranges MIME container defined in RFC 2616. Each CHUNK parcel MAY contain a complete MIME body, or it MAY contain a chunk, described using message/byterange. It is not necessary to know the length of a message or a chunk before sending, although setting one or both of these can help SIMS clients receiving a message, display progress information (for example, a progress thermometer).

INFORM requests are sent to indicate delivery status of a chunk. INFORM requests contain a Message-ID header with the same value as the corresponding CHUNK requests. INFORM requests are typically sent by the final recipient to indicate the delivery status of a chunk. (Note that the INFORM may provide status for a different sized chunk than sent in any of the original CHUNK requests). Other INFORM requests can be sent to indicate a forwarding delay or error condition. Unlike CHUNK transactions, INFORM transactions are multi-hop. Only the sender of the original message responds to an INFORM request. Relays forward responses to an INFORM back to the sender of the INFORM.

```
           <-- INFORM
<-- INFORM
200 OK -->
           200 OK -->
```

Finally, AUTH requests are used by clients with ephemeral addresses to create a handle they can use to receive incoming requests. AUTH requests can also contain credentials used to authenticate a client, and authorization policy used to block Denial of Service attacks.

AUTH requests do not contain a Message-ID header. AUTH requests are discussed in more detail in Section XXX TODO.

SIMS responses contain a 3-digit response code. Responses in the range 200-299 indicate a successful transaction. Responses in the ranges 400-499 and 500-599 indicate client and server errors respectively. Responses in the 600-699 range indicate that the receiver of a request has declined the request. Unlike in HTTP and SIP there are no redirection responses and no provisional responses.

5.2 SIMS Headers

SIMS parcels contain a number of header fields. Many header fields can contain an ordered list of multiple header field values separated by commas or printed on several lines with the same header name. For example, the following two Accept header fields are semantically identical (they contain the same header field values in the same order.

```
Accept: message/cpim
Accept: text/plain

Accept: message/cpim, text/plain
```

Note that for many headers fields, the order of header field values is significant and must be preserved (for example, see the discussion on the Via and Route header fields).

5.2.1 Essential Headers

There are three addresses which work in concert to properly route parcels. The Request-URI and the Route header work together to route SIMS requests: the Request-URI is the final target (Client) of the request) , and the Route header contains a list of relays (if any) which must be visited before contacting the Request-URI. The Via header contains a list of SIMS nodes used to route responses back to the sender of the request.

The Via header indicates the path taken by a request so far and the path that should be followed to route responses. The "branch" parameter contains a transaction identifier which allows SIMS nodes to correlate responses with requests. [blah blah]

The Route header contains a list of SIMS relays through which a request must traverse to reach a specific destination. A Route header MAY appear in any request. In a request, the top-most Route header is contacted according to the rules in [seciton foo] until the Route list is exhausted. Then the Request-URI is contacted. In

addition, a Route header MUST appear in any 2xx response to an AUTH request. This indicates the list of URIs that the client should advertise for requests targetted to the client.

The Max-Forwards header contains an integer value of the maximum number of nodes the current request may pass through, before a 483 Too Many Hops error is generated. The Max-Forwards header prevents infinite message forwarding loops. When a client sends a request for the first time, it sets the Max-Forwards header to the default starting value of 20.

5.2.2 Message-Specific headers

The Message-ID header contains a identifier unique to each message. The Message-ID header MUST be present in CHUNK and INFORM requests. In CHUNK requests it is used to associate multiple portions of a message (sent in several CHUNK requests) for reassembly. In INFORM requests it is used to correlate delivery status with the appropriate message. The Message-ID header MUST NOT be sent in responses.

The Thread-ID header is an optional header which can contain a unique identifier for threading related messages which do not share a common session (for example in a conference, group chat, or data collaboration).

The Call-ID header is optional in CHUNK and INFORM requests to correlate a message with a session identifier from other protocols such as SIP.

The Delivery-Status header contains the status of delivery of a portion of a message. The status is indicated by one of the following tokens. The portion of the message is identified by a byterange. [need more!] Copy a bunch of the values from RFC xxxx on message delivery disposition. These include dispositions such as displayed, dispatched, processed, deleted, denied, failed. the delivery status can indicate a portion of the relevant message was received (with the range parameter), whether the status was caused by human or automatic action, and can include an additional 3-digit error code.

The Message-Context header contains ... text-message or multimedia-message = email. instant-message and page-message are instant.

5.2.3 Headers related to MIME Content

The Accept header contains a list of the MIME types that the sender of the parcel supports. Note that SIMS mandatory to implement types

do not need to be included in this list. An empty list implies support for only the mandatory to implement types.

The Accept-Language header contains a list of preferred languages for reason phases, message bodies, delivery status, and other textual information. The "q" parameter specifies the relative preference among the listed languages, with the default value of 1.0 the most preferred.

The Content-Disposition header described how the content of the body is to be interpreted. This header is copied from RFC 2183. The value "inline" means to render the content immediately, while "attachment" means to store the attached MIME type as a file. An instant-message with Content-Disposition of attachment is a bit like a file transfer.

The Content-Language header describes the language of the contents of the body. It is optional.

The Content-Length header describes the length of the content of the body. It's use is optional when there is no body, or if there is a body which has natural MIME boundaries.

The Content-Type header describes the MIME type of the content. The Content-Type header MUST be present if a body is present. The Content-Type header MUST be present in CHUNK requests, even if no body is present.

The Message-Context header defined in RFC 3458 [6] describes the context of a message (for example: fax-message, voice-message, page-message, instant-message). This specification extends this header with two additional context values: instant-message, and file-delivery.

5.2.4 Headers used for extensibility

The Allow header contains a list of method names supported by the sender of the parcel.

The Require header contains a list of option tags which the other client must support. In a request, this indicates a list which the target client MUST support for the request to succeed. If the target client does not support these options it returns a 420 "Unsupported Extension" error response and includes a list of the option tags it does not understand in an Unsupported header field. In a 421 "Extension Required" response, this indicates a list of option tags which the responder expected the requester to advertise in a Supported header field value in the request.

The Supported header lists all the extensions supported by the sender of a parcel. The Supported header MAY included in any request, but it MUST be included in any 420 response.

The Unsupported header lists all the extensions in a request which where not supported or understood by the sender of a parcel. The Unsupported header is only sent in a 420 "Bad Extension" response.

5.2.5 Authentication headers

The Authentication-Info header provides optional information for HTTP Digest authentication. This header MAY be included in the response to an AUTH request. Semantics of the header are described in RFC 2617

The Authorization header contains authentication credentials for HTTP Digest authentication in an AUTH request. Section [x.y] .   Note that the parameters of this header are separated by commas instead of semicolons. The presence of commas in this header does not imply that there is more than one header field value for this header field (only one header field value is allowed). Semantics of the header are described in RFC 2617. This header MUST NOT appear in any parcel other than an AUTH request.

The WWW-Authenticate header [more]

5.2.6 Time-related headers

The Date header contains the date and time in RFC 1123 format. In SIMS, the date and time are always expressed in the "GMT" timezone.

The Expires header in a provides a relative time after which the action implied by the method of the request is no longer of interest. In a request, the Expires header indicates how long the sender would like to . In a response, the Expires header indicates how long the responder considers this information relevant (if the responder [more]

The Min-Expires header contains the minimum duration a server will permit in an Expires header. It is sent only in 423 "Interval Too Brief" responses.

The Retry-After header [more]

5.2.7 Error-related headers

The Error-Info header provides a pointer to additional information about an error-code in a response, or delivery error (conveyed in an

INFORM request).

The Warning header [snore, maybe we should delete this one]

5.2.8 The Server and User-Agent headers

The Server header contains information about the software used to handle the request. Use of this header is useful for debugging and troubleshooting, but can also reveal potentially private information.

The User-Agent header contains information about the software used to initiate the request. Use of this header is useful for debugging and troubleshooting, but can also reveal potentially private information.

5.2.9 Table of header fields

The following table explains which headers are optional (o), mandatory (m), or not appropriate (-) for requests and responses to each method defined in this specification. For the requests, a specific 3-digit code indicates that the header is only meaningful for that specific code. The code 4xx indicates that the header is valid in any 400-class response.

	Requests				Responses			
	CHUNK	INFORM	AUTH	???	CHUNK	INFORM	AUTH	???
Accept	o	o	o	o	4xx	4xx	4xx	4xx
Accept-Language	o	o	o	o	4xx	4xx	4xx	4xx
Allow	o	o	o	o	-	-	405	405
Authentication-Info	-	-	-	-	-	-	o	-
Authorization	-	-	o	-	-	-	-	-
Call-ID	o	o	-	o	-	-	-	-
Content-Disposition	o	o	o	o	o	o	o	o
Content-Language	o	o	o	o	o	o	o	o
Content-Length	o	o	o	o	o	o	o	o
Content-Type	m	o	o	o	o	o	o	o
Date	o	o	o	o	o	o	o	o
Delivery-Status	-	m	-	-	-	-	-	-
Error-Info	-	o	-	-	4xx	4xx	4xx	4xx
Expires	o	o	o	o	o	o	o	o
Max-Forwards	m	m	m	m	-	-	-	-
Message-Context	o	-	-	-	-	-	-	-
Message-ID	m	m	-	o	-	-	-	o
Min-Expires	-	-	-	-	423	423	423	423
Require	o	o	o	o	421	421	421	421
Retry-After	-	o	-	o	501	501	501	501
Route	o	o	o	o	-	-	2xx	-

Server	-	-	-	-	o	o	o	o
Supported	o	o	o	o	o	o	o	o
Thread-ID	o	o	-	o	-	-	-	-
Unsupported	-	-	-	-	420	420	420	420
User-Agent	o	o	o	o	-	-	-	-
Via	m	m	m	m	m	m	m	m
Warning	-	o	-	-	4xx	4xx	4xx	4xx
WWW-Authenticate	-	-	-	-	-	-	401	-

All parcels MUST contain a Via header field. Clients and relays set the Via header when sending requests and consume the Via on the return to route responses.

The Route header is used to provide a list of relays to traverse before visiting the Request-URI.

The Message-ID header is used in CHUNK and INFORM requests to refer to a specific message.

The Delivery-Status header is used in INFORM requests to indicate the status of a chunk or an entire message. Some examples:

```
Delivery-Status: ok;range=0-131071
Delivery-Status: ok;range=*
Delivery-Status: stored
Delivery-Status: failure;error=disk-full
```

Other Optional headers (temporal relevance, priority)

Note Expire header must look like Expire: 3600 meaning expires 3600 seconds in future. Absolute times are not supported.

5.3 SIMS Responses

Response codes semantically convey the success or failure of a request. These meaning of each response code is described briefly.

200 OK indicates that the request was successful. 202 Accepted indicates that the request was accept for further processing.

[TODO: fill-in semantics]
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable

408 Request Timeout
409 Puzzle Required
410 Gone
413 Request Entity Too Large
414 Request-URI Too Large
415 Unsupported Media Type
416 Unsupported URI Scheme
420 Bad Extension
421 Extension Required
423 Interval Too Brief
480 Temporarily not available
481 Message/Transaction Does Not Exist
482 Loop Detected
483 Too Many Hops
488 Not Acceptable Here
491 Request Pending
493 Undecipherable

500 Internal Server Error
501 Not Implemented
503 Service Unavailable
504 Server Time-out

603 Decline indicates that the request was declined due to user or administrator policy

5.4 SIMS bodies

Body handling and use of message/byteranges

```
CHUNK
Content-type: multipart/byteranges; boundary=------bound123456

-------bound123456
Content-type: text/plain
Content-range: bytes 0-2/8

hi
-------bound123456--
```

The "0" indicates that the data in this body starts is for byte location 0 in the complete message. The "2" is a hint of the byte position of the last byte in this chunk but MUST be ignored if the actual size is different. The "8" indicates the size of the total parcel. If it is unknown, a * would be used.

An important feature of the way the bodies are defined is that a network element sending a message, can decide to change the size of

what it is sending after it starts sending. For example, say that an element has 500 bytes of a message that start at location 1000 to 2000. It expects to send all 500 bytes but after sending the first 5 bytes that contain the the word hello, the element discovers there is a higher priority message that it needs to send over the same link. It closes off the first messages. The receiver will get something that looks like:

```
CHUNK
Content-type: multipart/byteranges; boundary=-----bound123456

-------bound123456
Content-type: text/plain
Content-range: bytes 1000-1499/8000

12345
-------bound123456--
```

If a relay has selected a boundary marker of "bound1234" and encounters the string "bound1234" in the data it is sending. It can just close off the current parcel and start a new one so there is no need to escape any of the data inside of the multipart bodies.

The multipart boundaries are constructed in a special way to allow for simple high speed parsing of them. In addition to the two dashes (-) that are normally before a boundary, the boundary itself MUST start with five additional dashes followed by a string that MUST have at least 16 bits of randomness in it. For example, a valid boundary would be "boundary=-----6ea7" where the 6ea7 was a randomly chosen four digit hexadecimal number.

The advantage of this is there will always be several "-" in a row in the boundaries that the scanner is searching for. This guarantees that 4 of then will be aligned on a 32 bit boundary and the scanner can quickly look for them by just looking for a 32 bit value that is equal to the "----". Once this word is found, the scanner can carefully check and see if this is the boundary it is looking for or just some random data.

All SIMS clients and relays must support multipart/related, multipart/mixed, message/byteranges, and multipart/signed MIME types. It is not required to check the signatures if they don't support S/MIME but they still need to be able to receive the content in a multipart/signed messages. Any MIME type that is acceptable for content (such as text/plain) must also be supported inside any supported MIME container.

6. Procedures

6.1 Client behavior

6.1.1 Sending requests

To send a new request, clients start by setting the Request-URI to the final target (the URI of the receiving client) and the method of the request (ex: CHUNK, AUTH, INFORM). The client also includes a Max-Forwards header with the default value (20), and a Via identifying itself. If the requests needs to be routed through any relays, those relay should be listed in a Route header field. If a body is present in the request, the appropriate Content-* headers need to be present (for example: Content-Type, Content-Disposition, Content-Length). If the attached content is large as defined by local policy, the outermost MIME container SHOULD be of the type message/byteranges. If any extensions involving option-tags are required, the client includes these in a Require header field. The client also includes any method-specific headers and any optional headers desired.

When a new request is ready to send, the client MUST determine the next-hop target URI by taking the URI in the topmost Route header field value if one exists or the Request-URI if no Route header field values exist. Once the next-hop URI is determined, the client MUST use the resolution rules described in Section 8 to find the appropriate address, port, and transport to use. Next the client MUST check if there is already an existing suitable connection to the next-hop target. If so, the client MUST send the request over the most suitable connection. Suitability MAY be determined by a variety of factors such as measured load and local policy, however in most simple implementations a connection will be suitable if it exists and is in an active state.

If the client wants to interrupt sending a request after the request headers have been sent (while sending the body contents) to deliver another parcel, the client SHOULD close the MIME boundary associated with the outermost request body, and therefore complete the request early. Clients MUST NOT interrupt sending parcel start lines (the request or response line) or parcel headers. In addition, clients SHOULD chunk messages based on the amount of data sent in a configurable amount of time. The default time for a chunk is one minute.

After the last byte of the request is sent, the client MUST set a timer for 32 seconds. If a response to that request is not received within 32 seconds, the client will consider that the request failed. When receiving a response, all SIMS nodes MUST verify that the top

Via header field value corresponds to the node receiving the response, and that the branch tag matches a valid transaction for that node. If either case is not true the client SHOULD silently discard the response. If the branch tag matches a valid transaction, the client MUST mark the transaction completed.

If the client receives a success response, it should continue sending any additional portions of the relevant outstanding message. If the client receives a recoverable error (for example a 416 Not Acceptable response), the client SHOULD try to resubmit the request if it is capable after modifying the request to address the nature of the error. Note that any resubmitted request MUST have a different transaction identifier than the original request.

When sending a CHUNK request, the client MUST include a Message-ID header, and MAY add Thread-ID, Call-ID, Content-Disposition, and Message-Context headers to further identify the handling of the content of the message. If the client wishes to convey that the parcel is no longer relevant after some time period, it can include an Expires header field indicating when the chunk should no longer be forwarded.

When sending an INFORM request, the client MUST include a Message-ID header and a Delivery-Status header. The client MAY also include Error-Info, Retry-After, and Warning headers if the Delivery-Status does not indicate successful delivery.

When sending an AUTH request, the client MAY add an Expires header to request a SIMS URI that is valid for no longer that the provided interval. If an AUTH request returns a 401 Unauthorized request, the client SHOULD fetch the Digest challenge from the WWW-Authenticate header in the response and retry the AUTH request, including an Authorization header with the Digest response. Unlike in HTTP and SIP, Digest authentication in SIMS is only permitted for AUTH requests.

6.1.2 Receiving Requests

Upon receiving a valid SIMS request, SIMS clients add a "received" parameter to the topmost Via to indicate to the client the connection handle over which the request arrived. Clients MUST verify the Request-URI corresponds to an address managed by the client. (A collocated client and relay would handle the request as a relay). If the request is unacceptable for any reason, the client creates an appropriate error response and returns it over the connection from which the request arrived.

To form a request, a client deletes all the headers from the response

except for the Via headers. If an extension is required in the response, the client includes the required option-tags in a Require header. If a body is present (typically one is not), include the appropriate Content-* headers. If an error occurred, the client SHOULD include any headers mentioned in the description of the corresponding response code. (For example the Accept header should be included in a 416 Not Acceptable response). The receiving client MAY also include Retry-After, Error-Info, and/or Warning header fields. If the request was successful, the client returns a 200 or 202 response and may optionally include an Expires header indicating the actual time after which the receiving client will ignore the contents of the request.

When a client receives a CHUNK request, it SHOULD send an INFORM request to the client which initiated the content indicating the delivery status of the corresponding message.

6.1.3 Receiving CHUNK requests

A SIMS client that receives a CHUNK request MUST respond with a final response immediately. A 200-class response indicates the successful delivery of the message fragment to the final hop, but does not mean that the message has been read by the user.

The final response to the CHUNK MUST be sent to the previous hop, which could be a SIMS relay or the sender of the CHUNK.

The 2xx response to the CHUNK MUST NOT contain a body. A 4xx or 5xx response indicates that the message was not delivered successfully. A 6xx response means it was delivered successfully, but refused.

The client SHOULD reconstruct the original message sent by combining the message fragments that it receives in different CHUNK requests with the same messageID. It SHOULD not display or store the message until the entire message has been reconstructed.

After the final response has been sent, the client MUST send back an INFORM to the sender of the CHUNK request,indicating the successful end-to-end delivery of the message fragment. For more details on constructing the INFORM request, see section Section 6.1.4.

After the message has received fully, the client may display the message to the user. If the CHUNK expires before the client is able to present the message to the user, the client SHOULD handle the message based on local policy. Example policies include: deleting the message without displaying it, displaying to the user with an indication that the message is expired, or some other policy. If the message is displayed, the client SHOULD clearly indicate to the user

that the message has expired.

6.1.4 Sending INFORM requests

When a client or a note taker receives a message parcel, it MUST send an INFORM request that indicates the byte range that has been received. The route header for this INFOM message is formed by looking at the Via headers of the CHUNK request that was received. If an error response is received when sending an INFOM, it is not retried.

A relay can also send an INFORM to indicate that some error happened when sending sending a parcel. It is possible to get INFORM requests a long time after the original message was sent. If a client receives an INFORM for a message it knows nothing about, it can discard the INFORM.

6.1.5 Sending AUTH requests

Clients can be configured (typically through discovery or manual provisioning) with a list of relays they need to use. They MUST be able to form a connection to each relay and send an AUTH command to get a URI that can be used in route headers. The client can authenticate the relay by looking at the relay's TLS certificate. The relay MUST authenticate the client using digest authentication.

The relay will return a URI, or list of URIs, in the Route header of the response. When using a session-protocol such as SIP, these URI can be used by the client in the route set that is sent in the SDP to setup the session. The same URI can be used for multiple session to send to the client.

Example with two relays on one side. Need to AUTH to first, then use the supplied route header to AUTH to second thought the first.

NOTE - only auth not auth-int is needed because TLS provides integrity

When a client wishes to use more than one relay, they must AUTH to each relay they wish to use. Consider a client A, that whishes messages to flow from A to the first relays, R1, then on to a second relays, R2. This client with do a normal AUTH with R1. It will then do an AUTH transaction with R2 that is routed through R1. The client will form this AUTH messages by setting the request URI to R2 and adding a route header with the URI learned from R1 then sending this message to R1. R1 will forward this like a INFORM request is forwarded to R2.

When the client sends an AUTH request, it may set the Expires header a relative time. The relay will return a URI that is only valid for that periods of time.

6.1.6 Managing Connections

Clients should open connection whenever they wish to deliver a request and no suitable connection exists. For client to client connections, a client should close a connection when there are no longer any sessions associated with the connection. For connections to relays, the client should leave a connection up until no sessions are using the connection for a locally defined period of time, which defaults to 5 minutes for foreign relays and one hour for the client's relays.

6.2 Relay behavior

6.2.1 Generic request behavior

Like clients receiving requests, relays receiving requests MUST add a "received" parameter to the top most Via header. Relays then examine the topmost Route header field value and remove this if it matches a URI corresponding to the relay. If no Route header field value is present, the relay examines the Request-URI to determine if the Request-URI corresponds to the relay itself.

6.2.2 Forwarding CHUNK requests

A SIMS relay that receives a CHUNK request MUST respond with a final response immediately. A 200-class response indicates the successful delivery of the message fragment, but does not mean that the message has been forwarded on to its next hop.

The final response to the CHUNK MUST be sent to the previous hop, which could be a SIMS relay or the sender of the CHUNK.

The 2xx response to the CHUNK MUST NOT contain a body. A 4xx or 5xx response indicates that the message was not delivered successfully. A 6xx response means it was delivered successfully, but refused.

The SIMS relay MAY further break up the message fragment received in the CHUNK request into smaller fragments and forward them to the next hop in separate CHUNK requests. It MAY also combine message fragments received before or after this CHUNK request, and forward them out in a single CHUNK request to the next hop identified in the Route header. The SIMS relay MUST NOT combine message fragments from CHUNK requests with different messageIDs.

The SIMS relay MAY choose whether to further fragment the message, or combine message fragments, or send the message as is, based on some policy which is administered, or based on the network speed to the next hop, or any other mechanism.

If the SIMS relay has knowledge of the byte range that it will transmit to the next hop, it SHOULD update the message/byteranges parameter in the CHUNK request appropriately.

Before forwarding the CHUNK request to the next hop, the SIMS relay MUST inspect the URI in the topmost Route header field value. If it indicates this relay, the relay removes it from the Route header field. It MUST then delete all the Via headers from the new request. Then it MUST insert a Via header into the request for itself.

If the SIMS relay fails to forward the CHUNK on to the next hop, it SHOULD return an INFORM back to the sender of the CHUNK indicating the reason for failure. [how? example. see section]

6.2.3 Receiving AUTH requests

When a relay receives an AUTH request, it must digest challenge the request. Once the challenge is complete, it MUST provide a URI that can be used in future route headers. When the route URI is received in future messages. It MUST verify that this URI was issues by this relay. It MUST ensure that the message is either being forwarded from an entity that did the AUTH request that resulted in this URI or it is being forwarded to the the entity that did the AUTH request that resulted in this URI.

The relay does not necessarily needs to save state to meet these requirements. One way that a relay could implement this is the following. When an AUTH request arrives, the relay concatenates the current time, the identity of the sender of the AUTH request, the identity of the previous hop the request came from. It then takes the concatenates string and encrypts it with a key only the relay knows and uses this for form the user portion of the sims URI that it returns. Later when it receives a URI, it can decrypt this information and use it to decide if the request should be forwarded or not. If the relay is actually several servers that share a DNS name, the URI may also encrypt which server actually has the connection to the client.

When a relay receive an AUTH request, it must authenticate the client that sent it with digest, it must also authenticate the previous hop that send the message to it. When previous hop was a relay this is done with the mutual TLS while when the previous hop was a client mutual TLS MAY be used it is available or the client authorization

from the digest is used. The relay will generate a URI that it a token that allows messages to be forwarded to and from this client. If the previous hop was authenticated by mutual TLS, then the URI MUST be valid to route across any connection the relay has to the previous hop relay. If the previous hop was not authenticated by mutual TLS, then the URI MUST only be valid to route across the same connection that the AUTH was received on. If this connection is closed then reopened, the URI MUST NOT be valid. Valid to route means that when the relay receives a messages that contains this URI, if the message it going to element that was the previous hop in the AUTH, then the relay can forward it and if the messages is coming from previous hop in the AUTH, then the relay can forward it to any location, otherwise the RELAY must discard the message and MAY send a INFORM indicating the auth URI was bad. If the AUTH request contains an Expires header, then the relay MUST ensure that the URI is not valid to route after the expiry time.

It is possible to implement all of the above requirements without the relay saving any state. When a relay starts up it could pick a crypto random 128 bit password (K) and 128 bit initialization vector (IV). If the relay was actually a NDS farm, all the machines in the farm would need to share the same K. When an ATUH request was received the relay form a string that contains: the expiry time of the URI, an indication if the previous hop was mutual TLS authenticated or not and it it was, the name of the previous hop, if it was not the identifier for the connection which received the AUTH request. This string would be padded by appending a byte with the value 0x80 then adding zero or more bytes with the value of 0x00 until the string length is a multiple of 16 bytes long. A new random IV vector would be selected (it needs to change because it forms the salt) and the padded string would be encrypted using AES-CBC with a key of K. The IV and encrypted data and an SPI (security parameter index) that changed each time K changed would be base 64 encoded and form the user portion of the request URI. The SPI allows the key to be changed and for the system to know which K should be used. Later when the relay received this URI, it could decrypt it and check the current time was before the expiry time and check that the messages was coming from or going to the connection or location specified in the URI. Integrity protection is not required because it is extremely unlikely that random data that was decrypted would result in a valid location that was the same as the messages was routing to or from. When implementing something like this, implementers should be careful not to use a scheme like EBE that would allows portion of encrypted tokens to be cut and paste into others.

Note: A successful AUTH response returns a Route header which contains a base SIMS URI that the client can use to create a number of different URIs which are all associated with the current

connection.

6.2.4 Forwarding INFORM requests

A SIMS relay that receives an INFORM request, MUST inspect the URI in the topmost Route header field value. If it indicates this relay, the relay removes it from the Route header field. It MUST then insert a Via header into the request. Then, it MUST forward the INFORM request on to the next hop listed in the Route Header.

6.2.5 Forwarding Responses

Relays forward responses by first verifying the topmost Via corresponds to the Via and that the response matches a valid transaction. Then the relay sends the request over the connection which corresponds to the handle in the received tag of the next Via header field value. If this connection has closed, then the response is silently discarded.

A SIMS relay can distinguish between responses for an INFORM and a CHUNK request based on the transaction ID of the request (the branch tag in the Via)

6.2.6 Managing Connections

Relays should keep connection open as long as possible. If a connection has not been used in a significant time (many minutes) it could be closed. If the relay runs out of resource and must close connections, it should first stop accepting new connections from clients then start closing connections on a least recently used basis.

6.2.7 Forwarding unknown requests

Requests with an unknown method are forwarded as if they were INFORM requests.

6.3 Acting as a Message Taker

A Message Taker merely acts like a Client which returns different INFORM responses.

TODO - how do I let the message taker know to send all the requests it saved for me to me. I assume I still send INFOMS to the original sender as well as the message take to let them know I got the message.

7. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [7]. Section 6.1 of RFC 2234 defines a set of core rules that are used by this specification, and not repeated here. Implementers need to be familiar with the notation and content of RFC 2234 in order to understand this specification. Certain basic rules are in uppercase, such as SP, LWS, HTAB, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions to clarify the use of rule names.

The use of square brackets is redundant syntactically. It is used as a semantic hint that the specific parameter is optional to use.

The following rules are used throughout this specification to describe basic parsing constructs. Also, several rules are incorporated from RFC 2396 [5] but are updated to make them compliant with RFC 2234 [10]. These include:

```
      alphanum  =  ALPHA / DIGIT

      reserved    =  ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
                     / "$" / ","
      unreserved  =  alphanum / mark
      mark        =  "-" / "_" / "." / "!" / "~" / "*" / "'"
                     / "(" / ")"
      escaped     =  "%" HEXDIG HEXDIG
```

The most frequently-used production in SIMS is the token. Unless otherwise stated, tokens are case- insensitive. Non-token characters MUST be in a quoted string to be used within a parameter value.

```
      token       =  1*(alphanum / "-" / "." / "!" / "%" / "*"
                     / "_" / "+" / "`" / "'" / "~" )
```

A string of text is parsed as a single word if it is quoted using double-quote marks. In quoted strings, quotation marks (") and backslashes (\) need to be escaped. The backslash character (\) MAY be used as a single-character quoting mechanism only within quoted-string and comment constructs. Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this mechanism to avoid conflict with line folding and header separation.

```
      quoted-string  =  SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
      qdtext         =  LWS / %x21 / %x23-5B / %x5D-7E
                        / UTF8-NONASCII
      quoted-pair  =  "\" (%x00-09 / %x0B-0C / %x0E-7F)
```

Unlike SIP/2.0 and HTTP/1.1 which allow line folding, line folding in SIMS is not allowed. In SIMS header field values, all unquoted linear white space has the same semantics as SP. A recipient MAY replace any unquoted linear white space with a single SP before interpreting the field value or forwarding the message downstream.

The SWS construct is used when linear white space is optional, generally between tokens and separators. When tokens are used or separators are used between elements, whitespace is often allowed before or after the characters below.

```
      LWS = 1*WSP
      SWS = [LWS]

      HCOLON = SWS ":" SWS

      EQUAL   =  SWS "=" SWS ; equal
      LPAREN  =  SWS "(" SWS ; left parenthesis
      RPAREN  =  SWS ")" SWS ; right parenthesis
      RAQUOT  =  ">" SWS ; right angle quote
      LAQUOT  =  SWS "<"; left angle quote
      COMMA   =  SWS "," SWS ; comma
      SEMI    =  SWS ";" SWS ; semicolon
      LDQUOT  =  SWS DQUOTE; open double quotation mark
      RDQUOT  =  DQUOTE SWS ; close double quotation mark
```

The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 charset (RFC 2279 [7]). The TEXT-UTF8-TRIM rule is used for descriptive field contents that are n t quoted strings, where leading and trailing LWS is not meaningful. In this regard, SIMS differs from HTTP, which uses the ISO 8859-1 character set.

```
      TEXT-UTF8-TRIM  =  1*TEXT-UTF8char *(*LWS TEXT-UTF8char)
      TEXT-UTF8char   =  %x21-7E / UTF8-NONASCII
      UTF8-NONASCII   =  %xC0-DF 1UTF8-CONT
                      /  %xE0-EF 2UTF8-CONT
                      /  %xF0-F7 3UTF8-CONT
                      /  %xF8-Fb 4UTF8-CONT
                      /  %xFC-FD 5UTF8-CONT
      UTF8-CONT       =  %x80-BF


SIMS-URI         =  "sims:" [ userinfo ] hostport
                    uri-parameters
```

```
   userinfo         =  user  "@"
   user             =  1*( unreserved / escaped / user-unreserved )
   user-unreserved  =  "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
   hostport         =  host [ ":" port ]
   host             =  hostname / IPv4address / IPv6reference
   hostname         =  *( domainlabel "." ) toplabel [ "." ]
   domainlabel      =  alphanum
                       / alphanum *( alphanum / "-" ) alphanum
   toplabel         =  ALPHA / ALPHA *( alphanum / "-" ) alphanum

   IPv4address    =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
   IPv6reference  =  "[" IPv6address "]"
   IPv6address    =  hexpart [ ":" IPv4address ]
   hexpart        =  hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
   hexseq         =  hex4 *( ":" hex4)
   hex4           =  1*4HEXDIG
   port           =  1*DIGIT

   uri-parameters    =  *( ";" uri-parameter)
   uri-parameter     =  transport-param /  method-param / other-param
   transport-param   =  "transport="
                        ( "tcp" / "tls+tcp" / other-transport)
   other-transport   =  token
   method-param      =  "method=" Method
   other-param       =  pname [ "=" pvalue ]
   pname             =  1*paramchar
   pvalue            =  1*paramchar
   paramchar         =  param-unreserved / unreserved / escaped
   param-unreserved  =  "[" / "]" / "/" / ":" / "&" / "+" / "$"


   SIMS-parcel    =  Request / Response
   Request        =  Request-Line
                     *( parcel-header )
                     CRLF
                     [ parcel-body ]
   Request-Line   =  Method SP Request-URI SP SIMS-Version CRLF
   Request-URI    =  SIMS-URI / anyURI
   anyURI         =  scheme ":" *uric
   scheme         =  ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )
   uric           =  reserved / unreserved / escaped
   SIMS-Version   =  "SIMS" "/" 1*DIGIT "." 1*DIGIT

   parcel-header  = ( Accept
                   /  Accept-Language
                   /  Allow
                   /  Authentication-Info
                   /  Authorization
```

```
                  /  Call-ID
                  /  Content-Disposition
                  /  Content-Language
                  /  Content-Length
                  /  Content-Type
                  /  Date
                  /  Delivery-Status
                  /  Error-Info
                  /  Expires
                  /  Max-Forwards
                  /  Message-Context
                  /  Message-Id
                  /  Min-Expires
                  /  Require
                  /  Retry-After
                  /  Route
                  /  Server
                  /  Supported
                  /  Thread-ID
                  /  Unsupported
                  /  User-Agent
                  /  Via
                  /  Warning
                  /  WWW-Authenticate
                  /  extension-header ) CRLF


   CHUNKm          = %x43.48.55.4E.4B        ; CHUNK in caps
   INFORMm         = %x49.4E.46.4F.52.4D     ; INFORM in caps
   AUTHm           = %x41.55.54.48           ; AUTH in caps
   Method          = CHUNKm / INFORMm / AUTHm
                          / extension-method

   extension-method  =  token

   Response          =  Status-Line
                        *( message-header )
                        CRLF
                        [ message-body ]

   Status-Line     =  SIMS-Version SP Status-Code SP Reason-Phrase CRLF
   Status-Code     =  Success
                  /   Client-Error
                  /   Server-Error
                  /   Global-Failure
                  /   extension-code
   extension-code  =  3DIGIT
```

```
   Reason-Phrase   =  *(reserved / unreserved / escaped
                      / UTF8-NONASCII / UTF8-CONT / SP / HTAB)

   Success  =  "200"  ;  OK
            /  "202"  ;  Accepted

   Client-Error  =  "400"  ;  Bad Request
                 /   "401"  ;  Unauthorized
                 /   "402"  ;  Payment Required
                 /   "403"  ;  Forbidden
                 /   "404"  ;  Not Found
                 /   "405"  ;  Method Not Allowed
                 /   "406"  ;  Not Acceptable
                 /   "408"  ;  Request Timeout
                 /   "409"  ;  Puzzle Required
                 /   "410"  ;  Gone
                 /   "413"  ;  Request Entity Too Large
                 /   "414"  ;  Request-URI Too Large
                 /   "415"  ;  Unsupported Media Type
                 /   "416"  ;  Unsupported URI Scheme
                 /   "420"  ;  Bad Extension
                 /   "421"  ;  Extension Required
                 /   "423"  ;  Interval Too Brief
                 /   "480"  ;  Temporarily not available
                 /   "481"  ;  Message/Transaction Does Not Exist
                 /   "482"  ;  Loop Detected
                 /   "483"  ;  Too Many Hops
                 /   "488"  ;  Not Acceptable Here
                 /   "491"  ;  Request Pending
                 /   "493"  ;  Undecipherable

   Server-Error  =  "500"  ;  Internal Server Error
                 /   "501"  ;  Not Implemented
                 /   "503"  ;  Service Unavailable
                 /   "504"  ;  Server Time-out

   Global-Failure = "603"  ;  Decline


   Accept         =  "Accept" HCOLON
                      [ accept-range *(COMMA accept-range) ]
   accept-range   =  media-range *(SEMI accept-param)
   media-range    =  ( "*/*"
                     / ( m-type "/" "*" )
                     / ( m-type "/" m-subtype )
                     ) *( SEMI m-parameter )
   accept-param   =  ("q" EQUAL qvalue) / generic-param
   qvalue         =  ( "0" [ "." 0*3DIGIT ] )
```

```
                  / ( "1" [ "." 0*3("0") ] )
   generic-param  =  token [ EQUAL gen-value ]
   gen-value      =  token / host / quoted-string

   Accept-Language  =  "Accept-Language" HCOLON
                        [ language *(COMMA language) ]
   language         =  language-range *(SEMI accept-param)
   language-range   =  ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) / "*" )

   Allow            =  "Allow" HCOLON [Method *(COMMA Method)]

   Authentication-Info  =  "Authentication-Info" HCOLON ainfo
                            *(COMMA ainfo)
   ainfo                =  nextnonce / message-qop
                             / response-auth / cnonce
                             / nonce-count
   nextnonce            =  "nextnonce" EQUAL nonce-value
   response-auth        =  "rspauth" EQUAL response-digest
   response-digest      =  LDQUOT *LHEX RDQUOT

   Authorization     =  "Authorization" HCOLON credentials
   credentials       =  ("Digest" LWS digest-response)
                        / other-response
   digest-response   =  dig-resp *(COMMA dig-resp)
   dig-resp          =  username / realm / nonce / digest-uri
                         / dresponse / algorithm / cnonce
                         / opaque / message-qop
                         / nonce-count / auth-param
   username          =  "username" EQUAL username-value
   username-value    =  quoted-string
   digest-uri        =  "uri" EQUAL LDQUOT digest-uri-value RDQUOT
   digest-uri-value  =  rquest-uri ; Equal to request-uri as specified
                        by HTTP/1.1
   message-qop       =  "qop" EQUAL qop-value
   cnonce            =  "cnonce" EQUAL cnonce-value
   cnonce-value      =  nonce-value
   nonce-count       =  "nc" EQUAL nc-value
   nc-value          =  8LHEX
   dresponse         =  "response" EQUAL request-digest
   request-digest    =  LDQUOT 32LHEX RDQUOT
   auth-param        =  auth-param-name EQUAL
                        ( token / quoted-string )
   auth-param-name   =  token
   other-response    =  auth-scheme LWS auth-param
                        *(COMMA auth-param)
   auth-scheme       =  token
   LHEX              =  DIGIT / %x61-66 ;lowercase a-f
   ;   Some elements (authentication) force hex alphas to be lower case.
```

```
   Call-ID            =  "Message-ID" HCOLON msgid
   msgid              =  token [ "@" token ]

   Content-Disposition   =  "Content-Disposition" HCOLON
                            disp-type *( SEMI disp-param )
   disp-type             =  "render" / "status" /
                            disp-extension-token
   disp-param            =  handling-param / generic-param
   handling-param        =  "handling" EQUAL
                            ( "optional" / "required"
                            / other-handling )
   other-handling        =  token
   disp-extension-token  =  token

   Content-Language   =  "Content-Language" HCOLON
                         language-tag *(COMMA language-tag)
   language-tag       =  primary-tag *( "-" subtag )
   primary-tag        =  1*8ALPHA
   subtag             =  1*8ALPHA

   Content-Length   =  "Content-Length" HCOLON 1*DIGIT
   Content-Type     =  "Content-Type" HCOLON media-type
   media-type       =  m-type "/" m-subtype *(SEMI m-parameter)
   m-type           =  discrete-type / composite-type
   discrete-type    =  "text" / "image" / "audio" / "video"
                       / "application" / extension-token
   composite-type   =  "message" / "multipart" / extension-token
   extension-token  =  ietf-token / x-token
   ietf-token       =  token
   x-token          =  "x-" token
   m-subtype        =  extension-token / iana-token
   iana-token       =  token
   m-parameter      =  m-attribute EQUAL m-value
   m-attribute      =  token
   m-value          =  token / quoted-string

   Date          =  "Date" HCOLON rfc1123-date
   rfc1123-date  =  wkday "," SP date1 SP time SP "GMT"
   date1         =  2DIGIT SP month SP 4DIGIT
                    ; day month year (e.g., 02 Jun 1982)
   time          =  2DIGIT ":" 2DIGIT ":" 2DIGIT
                    ; 00:00:00 - 23:59:59
   wkday         =  "Mon" / "Tue" / "Wed"
                    / "Thu" / "Fri" / "Sat" / "Sun"
   month         =  "Jan" / "Feb" / "Mar" / "Apr"
                    / "May" / "Jun" / "Jul" / "Aug"
                    / "Sep" / "Oct" / "Nov" / "Dec"
```

```
   Delivery-Status =  "Delivery-Status" HCOLON msgstat
                      *(SEMI delivery-params)
   msgstat         =  "ok" / "stored" / "failure" / "delay" / token
   delivery-params =  delivery-range / deliver-err /
                      delivery-retry / generic-param
   delivery-range  =  "range" EQUAL
                      ("*" / ( begin-range "-" end-range ))
   begin-range     =  1*DIGIT
   end-range       =  1*DIGIT
   delivery-err    =  "error" EQUAL ( token / quoted-string )
   delivery-retry  =  "retry-after" EQUAL delta-seconds
   delta-seconds   =  1*DIGIT

   Error-Info      =  "Error-Info" HCOLON info *(COMMA info)
   info            =  LAQUOT anyURI RAQUOT *( SEMI generic-param)

   Expires         =  "Expires" HCOLON delta-seconds

   Max-Forwards    =  "Max-Forwards" HCOLON 1*DIGIT

   Message-ID      =  "Message-ID" HCOLON msgid

   MIME-Version  =  "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

   Min-Expires  =  "Min-Expires" HCOLON delta-seconds

   Priority        =  "Priority" HCOLON priority-value
   priority-value  =  "emergency" / "urgent" / "normal"
                      / "non-urgent" / other-priority
   other-priority  =  token

   Require      =  "Require" HCOLON option-tag *(COMMA option-tag)
   option-tag   =  token

   Retry-After  =  "Retry-After" HCOLON delta-seconds
                   *( SEMI retry-param )
   retry-param  =  ("duration" EQUAL delta-seconds)
                   / generic-param

   Route        =  "Route" HCOLON route-param *(COMMA route-param)
   route-param  =  LAQUOT SIMS-URI RAQUOT

   Server            =  "Server" HCOLON server-val *(LWS server-val)
   server-val        =  product / comment
   product           =  token ["/" product-version]
   product-version  =  token
   comment           =  LPAREN *(ctext / quoted-pair / comment) RPAREN
   ctext             =  %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
```

```
                       / LWS

   Supported   =  "Supported" HCOLON option-tag *(COMMA option-tag)

   Thread-ID   =  "Thread-ID" HCOLON msgid

   Unsupported =  "Unsupported" HCOLON option-tag *(COMMA option-tag)

   User-Agent  =  "User-Agent" HCOLON server-val *(LWS server-val)

   Via               =  "Via" HCOLON via-parm *(COMMA via-parm)
   via-parm          =  sent-protocol LWS sent-by *( SEMI via-params )
   via-params        =    via-received / via-branch
                        / via-extension
   via-received      =  "received" EQUAL connection-handle
   connection-handle =  token / hostport / quoted-string
   via-branch        =  "branch" EQUAL token
   via-extension     =  generic-param
   sent-protocol     =  protocol-name "/" protocol-version
                        "/" transport
   protocol-name     =  "SIMS" / token
   protocol-version  =  token
   transport         =  "TCP" / "TLS+TCP" / other-transport
   sent-by           =  host [ ":" port ]

   Warning        =  "Warning" HCOLON warning-value
                      *(COMMA warning-value)
   warning-value  =  warn-code SP warn-agent SP warn-text
   warn-code      =  3DIGIT
   warn-agent     =  hostport / pseudonym
                     ;  the name or pseudonym of the server adding
                     ;  the Warning header, for use in debugging
   warn-text      =  quoted-string
   pseudonym      =  token

   WWW-Authenticate  =  "WWW-Authenticate" HCOLON challenge
   challenge           =  ("Digest" LWS digest-cln *(COMMA digest-cln))
                          / other-challenge
   other-challenge     =  auth-scheme LWS auth-param
                          *(COMMA auth-param)
   digest-cln          =  realm / domain / nonce
                           / opaque / stale / algorithm
                           / qop-options / auth-param
   realm               =  "realm" EQUAL realm-value
   realm-value         =  quoted-string
   domain              =  "domain" EQUAL LDQUOT URI
                          *( 1*SP URI ) RDQUOT
   URI                 =  SIMS-URI / anyURI
```

```
   nonce                 =  "nonce" EQUAL nonce-value
   nonce-value           =  quoted-string
   opaque                =  "opaque" EQUAL quoted-string
   stale                 =  "stale" EQUAL ( "true" / "false" )
   algorithm             =  "algorithm" EQUAL ( "MD5" / "MD5-sess"
                            / token )
   qop-options           =  "qop" EQUAL LDQUOT qop-value
                            *("," qop-value) RDQUOT
   qop-value             =  "auth" / token

   extension-header    =  header-name HCOLON header-value
   header-name         =  token
   header-value        =  *(TEXT-UTF8char / UTF8-CONT / LWS)
   parcel-body         =  *OCTET
```

8. Finding SIMS Servers

When sending a response, the response is always forwarded over an existing connection using the connection handle set in the receiver parameter in the topmost Via header field value and the sent-by transport in that Via header field value to determine the correct connection.

When resolving a URI (for example from a Route header field, or from the Request-URI), examine the hostport portion of the URI and the transport URI parameter to decide how to proceed.

If the hostport is an IPv4 address or an IPv6 reference, send the request to that address using the port and transport specified in the URI. If no transport is provided, use the default (tls+tcp). If no port number is provided, use the default for the selected protocol (port 8999 for tcp, and port 9000 for tls over tcp).

If the hostport is a domain name and an explicit port number is provided, attempt to lookup a valid address record (A, AAAA, or A6) for the domain name. Connect using the specified protocol (or the default of tls+tcp if none is specified) and port number.

If a domain name is provided, but no port number, perform a DNS SRV [8] lookup for all transports supported by the client and select the entry with the highest weight. If no SRV records are found, try an address lookup using the default port number procedures described in the previous paragraph. Note that AUTH requests MUST only be sent over a TLS-protected channel. An SRV lookup in the example.com domain might return:

```
;; in example.com.      Pri Wght Port Target
_sims+tls._tcp   IN SRV 0   1    9000 server1.example.com.
_sims+tls._tcp   IN SRV 0   2    9000 server2.example.com.
_sims._tcp       IN SRV 1   1    8999 server1.example.com.
_sims._tcp       IN SRV 1   2    8999 server2.example.com.
```

If implementing a relay farm, it is RECOMMENDED that each member of the relay farm have an SRV entry. If any members of the farm have multiple IP addresses (for example an IPv4 and an IPv6 address), each of these addresses SHOULD be registered in DNS as separate A, AAAA, or A6 records corresponding to a single target.

9. Security Considerations

This section first describes the security mechanisms available for use in SIMS. Then the threat model is presented. Finally we list implementation requirements related to security.

9.1 Using HTTP Authentication

AUTH requests SHOULD be authenticated using HTTP authentication. HTTP authentication is done as described in [RFC 2617], with the following exceptions. Basic authentication MUST NOT be used. A qop value of auth-int MUST NOT be used as the AUTH requests are integrity protected by TLS and there is no body to protect. Note that unlike in some usages of HTTP Authentication (for example, SIP), the uri parameter in the Authorize header is the same as the Request-URI in the request line of the SIMS parcel of the AUTH request. Note the BNF in RFC-2617 has an error--the value of the uri parameter MUST be in quotes. The BNF in this document is correct, as are the examples in RFC 2617.

9.2 Using TLS

TLS is used to authenticate relays to senders and to provide integrity and confidentiality for the headers being transported. SIMS client and relays MUST support TLS. Clients and relays MUST support the TLS ClientExtendedHello extended hello information for server name indication as described in RFC 3546 [9]. A TLS cipher-suite of TLS_RSA_WITH_AES_128_CBC_SHA [10] MUST be supported (other cipher-suites MAY also be suported). Relays must act as TLS servers and present a certificate with their identity in the SubjectAltName using the choice type of dnsName. Relay to relay connections MUST use TLS and client to relay communications MUST use TLS for AUTH requests and responses.

9.3 S/MIME

Since SIMS carries arbitrary MIME content, it can trivially carry S/MIME protected messages are well. Note that all SIMS implementations MUST support the multipart/signed MIME type even if they do not support S/MIME. Since SIP can carry a session key, S/MIME messages in the context of a session can be protected using a key-wrapped shared secret provided in the session setup.

9.4 Threat Model

This section discuses the threat model and the broad mechanism that must come into place to secure the protocol. The next section describes the details of how the protocol mechanism meet the broad requirements.

SIMS allows two peer to peer clients to exchange messages. Each peer can select a set of relays to perform certain policy operation for them. This combined set of relays is referred to as the route set. There often exists a channel outside of SIMS, such as out-of-band provisioning or an explicit rendezvous protocol such as SIP, that can securely negotiate setting up the SIMS session and communicate the route set to both clients. A client may trust a relay with certain types of routing and policy decisions but it might or might not trust the relay with all the contents of the session. For example, a relay being trusted to look for viruses would probably need to be allowed to see all the contents of the session. A relay that helped deal with firewall traversal of the ISPs firewall would likely not be trusted with the contents of the session but would be trusted to correctly forward information.

Clients need to be able to authenticate that the relay they are communicating with is the one they trust. Likewise, relays need to be able to authenticate the client is the authorized client for them to forward information to. Clients need the option of ensuring information between the relay and the client is integrity protected and confidential to elements other than the relays and clients. To simplify the number of options, traffic between relays must always be integrity protected and encrypted regardless of if the client request it or not. There is no way for the clients to tell the relays what strength of crypto to use between relays other than the clients to choose to use relays that are operated by people requiring an adequate level of security.

The system also need to stop the messages from being directed to relays that are not supposed to see them. To keep the relays from being used in DDoS attacks, the relays must not forward messages unless they have a trust relationship with either the client sending or receiving the message and that they only forward that message if it is coming from or going to the client they have the trust

relationship with. If a relay has a trust relationship with the client that is the destination of the message, it should not send the message anywhere except the client that is the destination.

Some terminology used in this discussion is SClient is the client sending a message and RClient is the client receiving a message. SRelay is a relay the sender trusts and RRelay is a relay the receiver trusts. The message will go from SClient to SRelay1 to SRelay2 to RRelay2 to RRelay1 to RClient.

9.5 Security Mechanism

Confidentiality and Privacy from elements not in the route set is provided by using TLS on all the transports. If a client decided to not use TLS that is it's choice but relays must use TLS. Clients must implement TLS.

The relays authenticate to the clients using TLS (but don't have to do mutual TLS). The clients authenticate to the relays using HTTP Digest inside of TLS. Relays authenticate to each other using mutual TLS.

The clients can protect the contents so that the relays can not see them by using S/MIME encryption. End to end signing is also possible with S/MIME.

The complex part is making sure that relays do not send messages place where they should not. This is done by having the client authenticate to the relay and having the relay return a token. Messages that contain this token can be relayed if they come from the client that got the token or if they are being forwarded towards the client that got the token. The tokens must only ever be seen by things in the route set or other elements that at least one of the parties trusts. If some 3rd party discovers the token that RRelay2 uses to forward messages to RClient, then that 3rd party can send as many messages as they want to RRelay2 and it will forward them to RClient. The 3rd party can not cause them to be forwarded anywhere except to RClient eliminating the open relay problems. SRelay1 will not forward the message unless it contains a valid token.

When SClient goes to get a token from SRelay2, this request is relayed through SRelay1. SRelay authenticates that it really is SClient requesting the token but it generates a token that is only valid for forwarding messages to or from SRelay1. SRelay two knows it is connected to SRelay1 because of the mutual TLS.

The tokens are carried in the user portion of the SIMS URLs.

Issues: How to tokens expire - rekeying. Will probably use Expire header on AUTH response. Token MAY be valid for between 10 minutes and 24 hours with 1 hour recommended. Both sides need to do a SIP re-invite to set up new tokens before the old one expires.

Issues: Token good for single session or for all session

Note: tokens are only required for relays, not clients or note takers.

TODO talk about example from client to client and from Client A, then to a relay that A uses, RA, then on to client B.

9.6 Preventing Spam and Denial of Service Attacks

While this specification already implements a number of significant improvements to prevent unsolicited messaging and Denial of Service, additional mechanisms are envisioned being useful in the future. The 402 Payment Required and 409 Puzzle Required response codes are reserved for future use and may be useful to further discourage unsolicited messages.

10. IANA Considerations

10.1 Port number registrations

SIMS uses port XXX for SIMS over TCP and port YYY for TLS over TCP. These port numbers should be determined by allocation from IANA.

10.2 URI scheme registration

This document defines the sims: URI scheme.

Scheme:	sims
Syntax:	Defined in Section 7 of this document
Character-Encoding:	UTF-8
Intended Usage:	Real-time delivery of MIME content, especially instant messages
Protocol:	SIMPLE Instant Messaging Sessions (SIMS)
Security Considerations:	Section 9 of this document
Relevant Publications:	This document

10.3 Message-Context

This document registers the message-context: "instant-message". The contact person is Rohan Mahy, rohan@cisco.com.

10.4 SDP Parameters

This document registers the following SDP parameters:

[TODO] accept and hop attributes

11. Using SIMS with SIP and SDP

In order for two SIMS clients to communicate with each other, they need to negotiate the characteristics of the SIMS session. These include the addresses where messages can be sent, the path that the SIMS requests/responses should take, and the content type that is acceptable to both ends.

This information MAY be exchanged and agreed upon between two SIMS clients using a session setup protocol like SIP, and the negotiation of the session characteristics MAY be done using the offer-answer approach with SDP contained in the SIP messages.

The Call-ID of the SIP session SHOULD be used as the Call-ID in the SIMS messages, so that the correlation between the media and the control signaling can be achieved.

11.1 SDP Extensions

There will be an m-line in the SDP for the SIMS session. The m-line has the form:

```
m = <media> <port> <protocol> <format-list>
```

The media type for a SIMS session SHOULD be "message". The port is not used. The protocol should be sims/tcp or sims/tcp+tls. And the format list is not used. It should be set to "*".

The m-line used to define a SIMS session has two attributes: the hop attribute and the accept-type attribute.

CHUNK requests can carry any MIME encoded payload. Endpoints specify MIME content types that they are willing to receive in the accept types "a"-line attribute. This attribute has the following syntax:

```
accept-types       = accept-types-label ":" format-list
accept-types-label = "accept-types"
format-list        = format-entry *( SP format-entry)
format-entry       = (type "/" subtype)
type               = token
```

```
subtype            = token
```

SDP offers for SIMS sessions MUST include an accept-types attribute. SDP answers MUST also include the attribute, which MUST contain either the same list as in the offer or a subset of that list.

If no format-entry is specified in the accept-types attribute, it indicates that the sender may attempt to send messages with media types that have not been explicitly listed. If the receiver is able to process the media type, it does so. If not, it will respond with a 415. Note that all explicit entries SHOULD be considered preferred over any non-listed types. This feature is needed as, otherwise, the list of formats for rich IM devices may be prohibitively large.

The accept-types attribute may include container types, that is, mime formats that contain other types internally. If compound types are used, the types listed in the accept-types attribute may be used both as the root payload, or may be wrapped in a listed container type. (Note that the container type MUST also be listed in the accept-types attribute.)

Clients specify the relays they wish to use in an "a=hop" attribute line in the SDP. A SIP answer only contains the relays that that side wishes to use, it does not include the relays that the client that made the offer wishes to use. This attribute line has the following syntax:

```
  hop-attribute = hop-label ":" sims-url
  hop-label = "hop"
```

There can be several hop labels in the SDP and they are associated with the m line that proceed them. The top hop one corresponds to the relay closest to the client that is sending the SDP and the next hop corresponds to the next relay out and so on.

A sample SDP offer for a SIMS session could look like:

```
c=IN IP4 invalid.none
 m=message 1234 sims/tcp+tls alice@alice.example.com
 a=accept: message/cpim text/plain text/html
 a=hop:sims:magic456@a.example.com:1234;transport=tcp+tls
```

In this offer Alice wishes to receive SIMS messages at alice@alice.example.com. She wants to use tcp+tls as the transport for the SIMS session. She can accept message/cpim, text/plain and text/html message boldies in CHUNK requests. She wishes to use the relay sims:magic456@a.example.com for the SIMS session.

To this offer, Bob's answer could look like:

```
c=IN IP4 invalid.none
 m=message 1234 sims/tcp+tls bob@bob.example.com
 a=accept: message/cpim text/plain
 a=hop:sims:magic789@b1.example.com:1234;transport=tcp+tls
 a=hop:sims:magic012@b2.example.com:1234;transport=tcp+tls
```

Here Bob has agreed to use tcp+tls as the transport, and wishes to receive the SIMS messages at bob@bob.example.com. He can accept only message/cpim and text/plain message bodies in CHUNK requests and has rejected text/html offer made by Alice. He wishes to use two relays for the SIMS session - sims:magic789@b1.example.com and sims:magic012@b2.example.com.

12. Comparison with requirements and with MSRP

TODO - Topics to compare: TCP fan out, HOL blocking, next hop congestion at a relay, congestion back pressure, robust sending of a message even as host temporarily disconnects and reconnects. scale, relay farms, multiple relays, and congestion.

13. Examples

13.1 Client to Client with SIP

In this example, Alice and Bob setup a SIMS session with the help of SIP. To keep the example simple and easy to understand, there are no SIP proxies shown. There are no SIMS relays which need to be traversed between Alice and Bob. It also shows the session tear-down using a SIP BYE.

```
        Alice                        Bob
          |                            |
          |                            |
          |---------INVITE (1)------->|
          |                            |
          |<------200 OK (2)----------|
          |                            |
          |----------ACK (3)--------->|
          |                            |
          |--------CHUNK (4)--------->|
          |                            |
          |<-------200 OK (5)---------|
          |                            |
          |<--------INFORM (6)--------|
          |                            |
```

```
             |---------200 OK (7)------->|
             |                            |
             |-----------BYE (8)-------->|
             |                            |
             |                            |
```

1 INVITE Alice -> Bob (SIP) : Alice sends an INVITE to Bob to start an IM session, with an SDP offer for the session.

```
INVITE sip:bob@pc1.example.com SIP/2.0
Via: SIP/2.0/UDP pc2.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@pc1.example.com>
From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
Content-Type: application/ sdp
Content-Length: 120

c=IN IP4 invalid.none
m=message 1234 sims/tcp+tls alice@pc2.example.com
a=accept-types:text/plain message/cpim
```

2 200 OK Bob -> Alice (SIP): Bob responds with a 200 OK and an answer SDP.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc2.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@pc1.example.com>;tag=a6c85cf
From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
Content-Type: application/sdp
Content-Length: 131

c=IN IP4 invalid.none
m=message 1234 sims/tcp+tls bob@pc1.example.com
a=accept-types:text/plain
```

3 ACK Alice -> Bob (SIP): Alice sends an ACK to Bob and the session is successfully set up. Alice and Bob can now start sending messages to each other.

```
ACK sip:bob@pc1.example.com SIP/2.0
Via: SIP/2.0/UDP pc2.example.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@pc1.example.com>;tag=a6c85cf
From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 ACK
```

4 CHUNK Alice -> Bob (SIMS): Alice sends a CHUNK to Bob. This is a complete message.

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bKkjshdyff
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: text/plain;boundary=-----bound123456

-------bound123456
Hi Bob, How are you?
-------bound123456
```

5 200 OK Bob -> Alice (SIMS): Bob responds with a 200 OK to indicate successful delivery of the CHUNK.

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Call-ID: 987asjd97y7atg
```

6 INFORM Bob -> Alice (SIMS): Bob INFORMs Alice of the successful end-to-end delivery of the entire message.

```
INFORM sims:alice@pc2.example.com SIMS/1.0
```

```
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
   Delivery-Status:ok
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

7 200 OK Alice -> Bob (SIMS): Alice responds with a 200 OK to indicate that it has received the INFORM.

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

8 BYE Alice -> Bob (SIP): Alice sends a BYE to Bob to tear down the SIP session.

```
   BYE sip:alice@pc2.example.com SIP/2.0
   Via: SIP/2.0/UDP pc2.example.com;branch=z9hG4bKkjshdyff
   Max-Forwards: 70
   To: Bob <sip:bob@pc1.example.com>;tag=a6c85cf
   From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
   Call-ID: 987asjd97y7atg
   CSeq: 231 BYE
   Content-Length: 0
```

13.2 3 relays with SIP

In this example, Alice has been configured to use two relays (r1.example.com and r2.example.com) for SIMS, and Bob has been configured with one relay (r3.example.com). Alice and Bob establish a TLS session with the relays and authenticate themselves, getting back the URIs for the relays that they should use in the Route headers of the SIMS messages.

```
    Alice     r1.example.com    r2.example.com    r3.example.com   Bob
     |                |                 |              |               |
     |                |                 |              |               |
     |---AUTH (1)---->|                 |              |<--AUTH (5)----|
     |                |                 |              |               |
     |<------401 (2)--|                 |              |------401 (6)->|
```

```
    |                |                |              |               |
    |---AUTH (3)---->|                |              |<--AUTH (7)----|
    |                |                |              |               |
    |<--200 OK (4)---|                |              |--200 OK (8)-->|
    |                |                |              |               |
    |--AUTH (9)----->|                |              |               |
    |                |                |              |               |
    |                |---AUTH (10)--->|              |               |
    |                |                |              |               |
    |                |<-401  (11)-----|              |               |
    |                |                |              |               |
    |<-401 (12)------|                |              |               |
    |                |                |              |               |
    |--AUTH (13)---->|                |              |               |
    |                |                |              |               |
    |                |---AUTH (14)--->|              |               |
    |                |                |              |               |
    |                |<--200 OK (15)--|              |               |
    |                |                |              |               |
    |<--200 OK (16)--|                |              |               |
    |                                                                |
    |                                                                |
    |----------------------------INVITE (17)------------------------>|
    |                                                                |
    |<----------------------------200 OK (18)------------------------|
    |                                                                |
    |-------------------------------ACK (19)------------------------>|
    |                                                                |
    |                |                |              |               |
    |--CHUNK (20)--->|                |              |               |
    |                |                |              |               |
    |<--200 OK (21)--|                |              |               |
    |                |                |              |               |
    |                |--CHUNK (22)--->|              |               |
    |                |                |              |               |
    |                |<--200 OK (23)--|              |               |
    |                |                |              |               |
    |                |                |--CHUNK (24)->|               |
    |                |                |              |               |
    |                |                |<-200 OK (25)-|               |
    |                |                |              |               |
    |                |                |              |--CHUNK (26)-->|
    |                |                |              |               |
    |                |                |              |<--200 OK (27)-|
    |                |                |              |               |
    |                |                |              |<--INFORM (28)-|
    |                |                |              |               |
    |                |                |<-INFORM (29)-|               |
```

```
    |                |                |              |              |
    |                |<--INFORM (30)--|              |              |
    |                |                |              |              |
    |<--INFORM (31)--|                |              |              |
    |                |                |              |              |
    |--200 OK (32)-->|                |              |              |
    |                |                |              |              |
    |                |--200 OK (33)-->|              |              |
    |                |                |              |              |
    |                |                |-200 OK (34)->|              |
    |                |                |              |              |
    |                |                |              |--200 OK (35)->|
    |                |                |              |              |
    |                |                |              |              |
```

1 AUTH Alice -> r1.example.com (SIMS) - Alice wants to authenticate itself with the first relay

```
AUTH sims:r1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
Expires: 3600
```

2 401 Unauthorized r1.example.com -> Alice (SIMS) - Relay challenges Alice

```
SIMS/1.0 401 Unauthorized
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Expires: 3600
WWW-Authenticate: Digest
     realm="testrealm@host.com",
     qop="auth",
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

3 AUTH Alice -> r1.example.com (SIMS) - Alice responds to the challenge

```
AUTH sims:r1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
```

```
   Expires: 3600
   Authorization: Digest username="Alice",
        realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="sims:r1.example.com",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

4 200 OK r1.example.com -> Alice (SIMS) - Relay responds to Alice with its authentication info

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
    ;received=192.0.2.3
   Expires: 3600
   Authentication-info: rspauth="sims:saiulfywifucbscb@r1.example.com"
```

5 AUTH Bob -> r3.example.com (SIMS) - Bob wants to authenticate with its relay

```
   AUTH sims:r3.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bK4b43c2ff8.1
   Expires: 3600
```

6 401 AUTH r3.example.com -> Bob (SIMS) - Relay challenges Bob

```
   SIMS/1.0 401 Unauthorized
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bK4b43c2ff8.1
    ;received=192.0.2.3
   Expires: 3600
   WWW-Authenticate: Digest
        realm="testrealm@host.com",
        qop="auth",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

7 AUTH Bob -> r3.example.com (SIMS) - Bob responds to the challenge

```
AUTH sims:r3.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bK4b43c2ff8.1
Expires: 3600
Authorization: Digest username="Bob",
     realm="testrealm@host.com",
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     uri="sims:r3.example.com",
     qop=auth,
     nc=00000001,
     cnonce="0a4f113b",
     response="6629fae49393a05397450978507c4ef1",
     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

8 200 OK r3.example.com -> Bob (SIMS) - Relay responds to Bob with its authentication information

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Expires: 3600
Authentication-Info: rspauth="sims:skusblfygwuhrwuh@r3.example.com"
```

9 AUTH Alice -> r1.example.com (SIMS) - Alice wants to authenticate itself with its second relay now

```
AUTH sims:r2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
Route:sims:saiulfywifucbscb@r1.example.com
Expires: 3600
```

10 AUTH r1.example.com -> r2.example.com (SIMS) - This authenicate request is routed through the first relay, to which Alice has already authenticated itself

```
AUTH sims:r2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=sldhgsdhgqfwaf
```

```
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Expires: 3600
```

11 401 AUTH r2.example.com -> r1.example.com (SIMS) - Relay 2 challenges Alice

```
SIMS/1.0 401 Unauthorized
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=sldhgsdhgqfwaf
;received=192.0.2.4
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Expires: 3600
WWW-Authenticate: Digest
     realm="testrealm@host.com",
     qop="auth",
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

12 401 AUTH r1.example.com -> Alice (SIMS) - Relay 1 passes on the challenge to Alice

```
SIMS/1.0 401 Unauthorized
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Expires: 3600
WWW-Authenticate: Digest
     realm="testrealm@host.com",
     qop="auth",
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

13 AUTH Alice -> r1.example.com (SIMS) - Alice responds to the challenge

```
AUTH sims:r2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
Route: sims:saiulfywifucbscb@r1.example.com
```

```
   Expires: 3600
   Authorization: Digest username="Alice",
        realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="sims:r2.example.com",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

14 AUTH r1.example.com -> r2.example.com (SIMS) - Relay 1 passes on Alice's response to Relay 2

```
   AUTH sims:r2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=sldhgsdhgqfwaf
   Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
    ;received=192.0.2.3
   Expires: 3600
   Authorization: Digest username="Alice",
        realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="sims:r2.example.com",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

15 200 OK r2.example.com -> r1.example.com (SIMS) - Relay 2 accepts Alice's response and sends back its authentication info

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=sldhgsdhgqfwaf
    ;received=192.0.2.4
   Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
    ;received=192.0.2.3
   Expires: 3600
   Authentication-Info: rspauth="sims:eioweoerhgerofef@r2.example.com"
```

16 200 OK r1.example.com -> Alice (SIMS) - Relay 1 forwards Relay2's authentication info to Alice

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Expires: 3600
Authentication-Info: rspauth="sims:eioweoerhgerofef@r2.example.com"
```

17 INVITE Alice -> Bob (SIP) : Alice sends an INVITe to Bob to start an IM session, with an SDP offer for the session.

```
INVITE sip:bob@pc1.example.com SIP/2.0
Via: SIP/2.0/UDP pc2.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@pc1.example.com>
From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
Content-Type: application/ sdp
Content-Length: 120

c=IN IP4 invalid.none
m=message 1234 sims/tcp+tls alice@pc2.example.com
a=accept-types:text/plain message/cpim
a=hop:sims:saiulfywifucbscb@r1.example.com
a=hop:sims:eioweoerhgerofef@r2.example.com
```

18 200 OK Bob -> Alice (SIP): Bob responds with a 200 OK and an answer SDP.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc2.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@pc1.example.com>;tag=a6c85cf
From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
Content-Type: application/sdp
Content-Length: 131
```

```
c=IN IP4 invalid.none
m=message 1234 sims/tcp+tls bob@pc1.example.com
a=accept-types:text/plain
a=hop:sims:skusblfygwuhrwuh@r3.example.com
```

19 ACK Alice -> Bob (SIP): Alice sends an ACK to Bob and the session is successfully set up. Alice and Bob can now start sending messages to each other.

```
ACK sip:bob@pc1.example.com SIP/2.0
Via: SIP/2.0/UDP pc2.example.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@pc1.example.com>;tag=a6c85cf
From: Alice <sip:alice@pc2.example.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 ACK
```

20 CHUNK Alice -> r1.example.com (SIMS) - Alice sends a CHUNK to Bob. This will be routed through the three relays

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bKkjshdyff
Route: sims:saiulfywifucbscb@r1.example.com
Route: sims:eioweoerhgerofef@r2.example.com
Route: sims:skusblfygwuhrwuh@r3.example.com
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type:  text/plain;boundary=-----bound123456

-------bound123456
Hi Bob! How are you?
-------bound123456
```

21 200 OK r1.example.com -> Alice (SIMS) - Relay 1 responds to Alice that the CHUNK has reached it successfully.

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=z9hG4bKkjshdyff
```

```
     ;received=192.0.2.3
    Call-ID: 987asjd97y7atg
```

22 CHUNK r1.example.com -> r2.example.com (SIMS) - Relay 1 forwards the CHUNK as-is to Relay2

```
    CHUNK sims:bob@pc1.example.com SIMS/1.0
    Via: SIMS/1.0/TCP-TLS r1.example.com;branch=sldhgsdhgqfwaf
    Route: sims:eioweoerhgerofef@r2.example.com
    Route: sims:skusblfygwuhrwuh@r3.example.com
    Call-ID: 987asjd97y7atg
    Message-ID: 34561345
    Max-Forwards: 70
    Content-Type:  text/plain;boundary=-----bound123456

    -------bound123456
    Hi Bob! How are you?
    -------bound123456
```

23 200 OK r2.example.com -> r1.example.com (SIMS) - Relay2 responds to Relay1 that the CHUNK has reached it successfully

```
    SIMS/1.0 200 OK
    Via: SIMS/1.0/TCP-TLS r1.example.com;branch=sldhgsdhgqfwaf
     ;received=192.0.2.3
    Call-ID: 987asjd97y7atg
```

24 CHUNK r2.example.com -> r3.example.com (SIMS) - Relay2 forwards the CHUNK as-is to Relay3

```
    CHUNK sims:bob@pc1.example.com SIMS/1.0
    Via: SIMS/1.0/TCP-TLS r2.example.com;branch=wuoshfuetyheiot
    Route: sims:skusblfygwuhrwuh@r3.example.com
    Call-ID: 987asjd97y7atg
    Message-ID: 34561345
    Max-Forwards: 70
    Content-Type:  text/plain;boundary=-----bound123456

    -------bound123456
```

```
Hi Bob! How are you?
-------bound123456
```

25 200 OK r3.example.com -> r2.example.com (SIMS) - Relay3 responds to Relay2 that the CHUNK has reached it successfully

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r2.example.com;branch=wuoshfuetyheiot
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

26 CHUNK r3.example.com -> Bob (SIMS) - Relay3 forwards the CHUNK to Bob

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r3.example.com;branch=hsruoghlweugho
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: text/plain;boundary=-----bound123456

-------bound123456
Hi Bob! How are you?
-------bound123456
```

27 200 OK Bob -> r3.example.com (SIMS) - Bob reports its successful delivery tp relay3

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r3.example.com;branch=hsruoghlweugho
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

28 INFORM Bob -> r3.example.com (SIMS) - Bob now sends an INFORM to Alice to indicate the successful end-to-end delivery of the message

```
   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
   Route: sims:skusblfygwuhrwuh@r3.example.com
   Route: sims:eioweoerhgerofef@r2.example.com
   Route: sims:saiulfywifucbscb@r1.example.com
   Delivery-Status:ok
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

```
   29 INFORM r3.example.com -> r2.example.com (SIMS)
```

```
   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS r3.example.com;branch=wvehrugheurghei
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Route: sims:eioweoerhgerofef@r2.example.com
   Route: sims:saiulfywifucbscb@r1.example.com
   Delivery-Status:ok
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

```
   30 INFORM r2.example.com -> r1.example.com (SIMS)
```

```
   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS r2.example.com;brnach=woifwehfovndjnv
   Via: SIMS/1.0/TCP-TLS r3.example.com;branch=wvehrugheurghei
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Route: sims:saiulfywifucbscb@r1.example.com
   Delivery-Status:ok
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

```
   31 INFORM r1.example.com -> Alice (SIMS)
```

```
   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wkehweiothoqowq
   Via: SIMS/1.0/TCP-TLS r2.example.com;brnach=woifwehfovndjnv
```

```
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS r3.example.com;branch=wvehrugheurghei
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Delivery-Status:ok
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

32 200 OK Alice -> r1.example.com (SIMS)

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wkehweiothoqowq
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS r2.example.com;brnach=woifwehfovndjnv
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS r3.example.com;branch=wvehrugheurghei
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

33 200 OK r1.example.com -> r2.example.com (SIMS)

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS r2.example.com;brnach=woifwehfovndjnv
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS r3.example.com;branch=wvehrugheurghei
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

34 200 OK r2.example.com -> r3.example.com (SIMS)

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS r3.example.com;branch=wvehrugheurghei
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
```

```
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

35 200 OK r3.example.com -> Bob (SIMS)

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

13.3 client fragmentation

In this example, Alice wants to send a message to Bob. Alice decides to fragment this message into two parts.

```
          Alice              r.example.com                Bob
            |                       |                       |
            |----CHUNK (1)--------->|                       |
            |                       |                       |
            |<-----200 OK (2)-------|                       |
            |                       |                       |
            |                       |-------CHUNK (3)----->|
            |                       |                       |
            |                       |<------200 OK (4)-----|
            |                       |                       |
            |                       |<-----INFORM (5)------|
            |                       |                       |
            |<-------INFORM (6)-----|                       |
            |                       |                       |
            |------200 OK (7)------>|                       |
            |                       |                       |
            |                       |------200 OK (8)----->|
            |                       |                       |
            |------CHUNK (9)------->|                       |
            |                       |                       |
            |<-------200 OK (10)----|                       |
            |                       |                       |
            |                       |-----CHUNK (11)------>|
            |                       |                       |
            |                       |<-----200 OK (12)-----|
            |                       |                       |
            |                       |<------INFORM (13)----|
            |                       |                       |
```

```
             |<-----INFORM (14)------|                       |
             |                       |                       |
             |------200 OK (15)----->|                       |
             |                       |                       |
             |                       |-----200 OK (16)----->|
             |                       |                       |
```

1 CHUNK Alice -> r1.example.com (SIMS) - Alice sends the first CHUNK

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=hsruoghlweugho
Route: saiulfywifucbscb@r1.example.com
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: multipart/byteranges; boundary=-----bound123456

-------bound123456
Content-type: text/plain
Content-range: bytes 0-44/96

This is the first part of a two-part message
-------bound123456
```

2 200 OK r1.example.com -> Alice (SIMS) - Relay1 receives the CHUNK successfully

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=hsruoghlweugho
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

3 CHUNK r1.example.com -> Bob (SIMS) - Relay forwards the CHUNK as-is to Bob

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=shoghwogiwhgokb
Call-ID: 987asjd97y7atg
```

```
   Message-ID: 34561345
   Max-Forwards: 70
   Content-Type: multipart/byteranges; boundary=-----bound123456

   -------bound123456
   Content-type: text/plain
   Content-range: bytes 0-44/96

   This is the first part of a two-part message
   -------bound123456
```

4 200 OK Bob -> r1.example.com (SIMS) - CHUNK reaches Bob successfully

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=shoghwogiwhgokb
    ;received=192.0.2.3
   Call-ID: 987asjd97y7atg
```

5 INFORM Bob -> r1.example.com (SIMS) - Bob INFORMs Alice about the successful end-to-end delivery of the first part of the message

```
   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
   Route: sims:saiulfywifucbscb@r1.example.com
   Delivery-Status:ok;range=0-44
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

6 INFORM r1.example.com -> Alice (SIMS) - INFORM gets forwarded by the relay

```
   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wuwfiuhwifuhwif
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Delivery-Status:ok;range=0-44
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg
```

7 200 OK Alice -> r1.example.com (SIMS) - Alice responds to the INFORM

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wuwfiuhwifuhwif
 ;received=192.0.2.3
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Call-ID: 987asjd97y7atg
```

8 200 OK r1.example.com -> Bob (SIMS) - Relay forwards the response to the INFORM to Bob

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Call-ID: 987asjd97y7atg
```

9 CHUNK Alice -> r1.example.com (SIMS) - Alice sends the second CHUNK of the message to Bob

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=hsruoghlweugho
Route: saiulfywifucbscb@r1.example.com
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: multipart/byteranges; boundary=-----bound123456

-------bound123456
Content-type: text/plain
Content-range: bytes 45-96/96

This is the second and the last part of this message
-------bound123456
```

10 200 OK r1.example.com -> Alice (SIMS)

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=hsruoghlweugho
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

11 CHUNK r1.example.com -> Bob (SIMS) - Relay passes on the second CHUNK as-is to Bob

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=shoghwogiwhgokb
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: multipart/byteranges; boundary=-----bound123456

-------bound123456
Content-type: text/plain
Content-range: bytes 45-96/96

This is the second and the last part of this message
-------bound123456
```

12 200 OK Bob -> r1.example.com (SIMS)

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=shoghwogiwhgokb
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

13 INFORM Bob -> r1.example.com (SIMS) - Bob INFORMs Alice of the successful end-to-end delivery of the entire message

```
INFORM sims:alice@pc2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
Route: sims:saiulfywifucbscb@r1.example.com
Delivery-Status:ok
Message-ID: 34561345
Call-ID: 987asjd97y7atg
```

```
   14 INFORM r1.example.com -> Alice (SIMS)


   INFORM sims:alice@pc2.example.com SIMS/1.0
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wuwfiuhwifuhwif
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Delivery-Status:ok
   Message-ID: 34561345
   Call-ID: 987asjd97y7atg



   15 200 OK Alice -> r1.example.com (SIMS)


   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wuwfiuhwifuhwif
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg



   16 200 OK r1.example.com -> Bob (SIMS)


   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

13.4 relay fragmentation

In this example, Alice sends a message to Bob in a single CHUNK request. The relay decides that it needs to fragment the message into two parts.

```
           Alice                r.example.com              Bob
             |                        |                      |
             |-------CHUNK (1)------>|                      |
             |                        |                      |
             |<-----200 OK (2)-------|                      |
             |                        |                      |
             |                        |-------CHUNK (3)----->|
```

```
          |                        |                       |
          |                        |<------200 OK (4)-----|
          |                        |                       |
          |                        |<------INFORM (5)-----|
          |                        |                       |
          |<-------INFORM (6)-----|                       |
          |                        |                       |
          |--------200 OK (7)---->|                       |
          |                        |                       |
          |                        |------200 OK (8)----->|
          |                        |                       |
          |                        |------CHUNK (9)------>|
          |                        |                       |
          |                        |<-----200 OK (10)-----|
          |                        |                       |
          |                        |<-----INFORM (11)-----|
          |                        |                       |
          |<------INFORM (12)-----|                       |
          |                        |                       |
          |-------200 OK (13)---->|                       |
          |                        |                       |
          |                        |-------200 OK (14)--->|
          |                        |                       |
```

1 CHUNK Alice -> r1.example.com (SIMS) - Alice sends a message to
Bob.

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=hsruoghlweugho
Route: saiulfywifucbscb@r1.example.com
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: text/plain;boundary=-----bound123456

-------bound123456
This is the entire message which will be split into two by the relay
-------bound123456
```

2 200 OK r1.example.com -> Alice (SIMS)

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc2.example.com;branch=hsruoghlweugho
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

3 CHUNK r1.example.com -> Bob (SIMS) - Relay1 splits up the message body in the CHUNK message, and sends the first part to Bob.

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=scjbsdjfksbfsdj
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: multipart/byteranges; boundary=-----bound123456

-------bound123456
Content-type: text/plain
Content-range: bytes 0-32/68

This is the entire message which
-------bound123456
```

4 200 OK Bob -> r1.example.com (SIMS)

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=scjbsdjfksbfsdj
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

5 INFORM Bob -> r1.example.com (SIMS) - Bob INFORMs Alice of the successful end-to-end delivery of the first 32 bytes of the message

```
INFORM sims:alice@pc2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
Route: sims:saiulfywifucbscb@r1.example.com
Delivery-Status:ok;range=0-32
Message-ID: 34561345
Call-ID: 987asjd97y7atg
```

6 INFORM r1.example.com -> Alice (SIMS)

```
INFORM sims:alice@pc2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wsuefhwejhfwejfh
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Delivery-Status:ok;range=0-32
Message-ID: 34561345
Call-ID: 987asjd97y7atg
```

7 200 OK Alice -> r1.example.com (SIMS) - Alice waits for the INFORM
for the remaining bytes that it has already sent

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wsuefhwejhfwejfh
 ;received=192.0.2.3
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Call-ID: 987asjd97y7atg
```

8 200 OK r1.example.com -> Bob (SIMS)

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Call-ID: 987asjd97y7atg
```

9 CHUNK r1.example.com -> Bob (SIMS) - Relay1 now sends the remaining
message in a second CHUNK message to Bob

```
CHUNK sims:bob@pc1.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=scjbsdjfksbfsdj
Call-ID: 987asjd97y7atg
Message-ID: 34561345
Max-Forwards: 70
Content-Type: multipart/byteranges; boundary=-----bound123456

-------bound123456
```

```
Content-type: text/plain
Content-range: bytes 33-68/68

 will be split into two by the relay
-------bound123456
```

10 200 OK Bob -> r1.example.com (SIMS)

```
SIMS/1.0 200 OK
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=scjbsdjfksbfsdj
 ;received=192.0.2.3
Call-ID: 987asjd97y7atg
```

11 INFORM Bob -> r1.example.com (SIMS) - Bob INFORMs Alice about the successful end-to-end delivery of the entire message

```
INFORM sims:alice@pc2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
Route: sims:saiulfywifucbscb@r1.example.com
Delivery-Status:ok
Message-ID: 34561345
Call-ID: 987asjd97y7atg
```

12 INFORM r1.example.com -> Alice (SIMS)

```
INFORM sims:alice@pc2.example.com SIMS/1.0
Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wsuefhwejhfwejfh
Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Delivery-Status:ok
Message-ID: 34561345
Call-ID: 987asjd97y7atg
```

13 200 OK Alice -> r1.example.com (SIMS)

```
SIMS/1.0 200 OK
```

```
   Via: SIMS/1.0/TCP-TLS r1.example.com;branch=wsuefhwejhfwejfh
    ;received=192.0.2.3
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

   14 200 OK r1.example.com -> Bob (SIMS)

```
   SIMS/1.0 200 OK
   Via: SIMS/1.0/TCP-TLS pc1.example.com;branch=z9hG4bKnashds8
    ;received=192.0.2.1
   Call-ID: 987asjd97y7atg
```

14. Acknowledgments

Many thanks to the following members of the SIMPLE WG for spirited discussions on session mode: Ben Campbell, Jonathan Rosenberg, Robert Sparks, Paul Kyzivat, Allison Mankin, Jon Peterson, Brian Rosen, Dean Willis, Adam Roach, Aki Niemi, Hisham Khartabil, Pekka Pessi, and Chris Boulton

Normative References

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

[2]   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.

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

[4]   Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[5]   Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[6]   Burger, E., Candell, E., Eliot, C. and G. Klyne, "Message

Context for Internet Mail", RFC 3458, January 2003.

[7] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[8] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[9] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J. and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

[10] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[11] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[12] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[13] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[14] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[15] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

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

[17] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

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

Informative References

[19] Campbell, B., "Instant Message Sessions in SIMPLE", draft-ietf-simple-message-sessions-02 (work in progress), Oct

2003.

[20] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[21] Atkins, D. and G. Klyne, "Common Presence and Instant Messaging: Message Format", draft-ietf-impp-cpim-msgfmt-08 (work in progress), January 2003.

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

[23] Day, M., Aggarwal, S. and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.

[24] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[25] Mahy, R., "Relay Requirements for Session-Mode Instant Messaging", draft-mahy-simple-session-relay-reqs-00.txt (work in progress), February 2004.

[26] Mahy, R., "Benefits of Session-Mode Instant Messaging", draft-mahy-simple-why-session-mode-00.txt (work in progress), February 2004.

URIs

[27] <http://www.softarmor.com/simple/drafts/morgue/draft-sparks-simple-jabber-sessions-00.txt>

[28] <http://www.softarmor.com/simple/drafts/morgue/draft-rosenberg-simple-message-session-00.txt>

[29] <http://www.softarmor.com/simple/drafts/morgue/draft-rosenberg-simple-im-transport-00.txt>

[30] <http://www.softarmor.com/simple/drafts/morgue/draft-mrose-simple-exchange-01.txt>

Authors' Addresses

Cullen Jennings
Cisco Systems, Inc.
170 West Tasman Dr.
MS: SJC-21/3
San Jose, CA 95134
USA

Phone: +1 408 527-9132
EMail: fluffy@cisco.com

Rohan Mahy
Cisco Systems, Inc.
5617 Scotts Valley Drive, Suite 200
Scotts Valley, CA 95066
USA

EMail: rohan@cisco.com

Juhee Garg
Cisco Systems, Inc.
170 West Tasman Dr, MS: SJC21/2/4
San Jose, CA 95134
USA

EMail: juhee@cisco.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

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