PJSIP

Requirements Document

ABOUT PJSIP

PJSIP is small-footprint and high-performance SIP stack written in C.

PJSIP is distributed under dual licensing schemes: GPL and commercial license.

Please visit http://www.pjproject.net for more details.

ABOUT THIS DOCUMENT

DOCUMENT REVISION HISTORY

Version	Date	Author	Description
0.1	10 Sep 2005	bennylp	Initial revision

Table of Contents

Chapter 1:General

1.1 Objectives

The objectives, based on the priority:

Item	Case Description	Status
1.	**Compliant** The library MUST be compliant to the corresponding standards that it implements. This is the priority number one!	√
2.	**High Performance, Scalable, Flexible** Performance is the next highest priority. The performance MUST be scalable to multiple processors (minimum 4). Features that may improve (or compromise) performance SHOULD be able to be used/excluded from the library. The library MUST be able to utilize multithreading.	√
3.	**Small Footprint** The library MUST yield small footprint, sufficiently small for use in embedded appliances. It is expected that the total footprint of the library when all features are implemented will be several kilobytes instead of megabytes. The core should weight around 100KB only. However, the desire to make the library as small as possible MUST NOT sacrifice the design of the library. The library MUST be kept as general purpose SIP library, not library specifically built for small footprint devices.	√
4.	**Extensible, Feature Rich, Modular** The library MUST be extensible. Different kinds of extensions (e.g. ones that work on dialog layer, message layer, or transport layer) SHOULD be able to plug in to the library. Initially the implementation SHOULD provide support for several important extensions. Extensions MUST be implemented in modular fashion. Application developer MUST be able to select (i.e. link) to only selected extensions that he/she wants to use.	√
5.	**Easy to Use with High Level API, Modular** The library SHOULD provide reasonably easy to use high-level API. This high-level API SHOULD be implemented in modular fashion, in higher layer library. The existence of this high-level API MUST NOT compromise the core API.	√

1.2 Code Convention

Item	Case Description	Status
1.	[Coding Style, The Most Important Requirement!] Tab size=8, indentation=4 (vim: "set sts=4")	√

Chapter 2:Core Functions

This chapter describes requirements and function checklists for functionalities in the core library.

2.1 General Core Design Requirements

Item	Case Description	Status
1.	MUST be able to create multiple stacks within a single application. Multiple stacks are useful to apply different preferences (such as different extensions selection).	√
2.	MUST be multithread flexible.	√

2.2 Message Components

2.2.1 General

Item	Case Description	Status
1.	[Escapement] MUST provide consistent escapement rules. In general, the rule is if one component performs un-escaping during parsing, then that component MUST escape the characters during printing. Otherwise, the library provides **no** automatic escaping functionality in both parsing and printing the components. The library MUST escape and un-escape URI components automatically, except for the host part where escapement MUST NOT be used (Ref: RFC 3261 Section 19.1.2).	
2.	[Special Check for SIP URL] Just for checklist, the lexer MUST allow semicolon and plus characters to be put in the user part of an URL.	

2.2.2 URI Design Requirements

Item	Case Description	Status
1.	MUST support SIP (**sip:** and **sips:** scheme) and **tel:** URI scheme.	
2.	MUST have generic URI representation (e.g. for http, mailto, etc.)	
3.	All URI schemes MUST share generic and common interface.	

4.	URI parsing MAY be extensible. If URI parsing is extensible, then it MUST not cause confusion/conflict with above requirements (e.g. if application provides http_url struct, then given a HTTP URL, should the user use http_url or the generic URL).	

2.2.3 URI Comparison Rules

URI comparison rules, according to RFC 3261 Section 19.1.4:

Item	Case Description	Status
1.	A SIP and SIPS URI are never equivalent.	
2.	Comparison of the userinfo of SIP and SIPS URIs is case-sensitive. This includes userinfo containing passwords or formatted as telephone-subscribers. Comparison of all other components of the URI is case-insensitive unless explicitly defined otherwise.	
3.	The ordering of parameters and header fields is not significant in comparing SIP and SIPS URIs.	
4.	Characters other than those in the "reserved" set (see RFC 2396 [5]) are equivalent to their "encoding.	
5.	An IP address that is the result of a DNS lookup of a host name does not match that host name.	
6.	For two URIs to be equal, the user, password, host, and port components must match. A URI omitting the user component will not match a URI that includes one. A URI omitting the password component will not match a URI that includes one. A URI omitting any component with a default value will not match a URI explicitly containing that component with its default value. For instance, a URI omitting the optional port component will not match a URI explicitly declaring port 5060. The same is true for the transport-parameter, ttl-parameter, user-parameter, and method components.	
7.	URI uri-parameter components are compared as follows: o Any uri-parameter appearing in both URIs must match. o A user, ttl, or method uri-parameter appearing in only one URI never matches, even if it contains the default value. o A URI that includes an maddr parameter will not match a URI that contains no maddr parameter. o All other uri-parameters appearing in only one URI are ignored when comparing the URIs. o URI header components are never ignored. Any	

	present header component MUST be present in both URIs and match for the URIs to match. The matching rules are defined for each header field in RFC 3261 Section 20.	

2.2.4 Methods, Design Requirements

Item	Case Description	Status
1.	MUST provide generic representation of any methods. Support of new methods MUST NOT require changes in the library.	

2.2.5 Header Fields

Item	Case Description	Status
1.	MUST adhere the following header fields comparison rules: o Field names (i.e. header names) are case insensitive o Unless otherwise stated in the definition of a particular header field, field values, parameter names, and parameter values are case-insensitive. o Tokens are always case-insensitive. o Unless specified otherwise, values expressed as quoted strings are case sensitive. [RFC 3261, Section 7.3.1]	
2.	MUST be able to parse header field names in compact form.	
3.	SHOULD provide user with option to write header field names in compact form.	
4.	[Record-Route] MUST support this header field. The URI placed in the Record-Route header field value MUST be a SIP or SIPS URI.	
5.	[Contact Header] MUST support this header field. When the header field value contains a display name, the URI including all URI parameters is enclosed in "<" and ">". Even if the display-name is empty, the name-addr form MUST be used if the addr-spec contains a comma, semicolon, or question mark.	
6.	[Content-Length] This header field MUST be implemented.	
7.	[Content-Type] This header field MUST be implemented.	

8.	[Date Header] The library SHOULD provide implementation for Date header representation and parsing. However this implementation MUST be able to be excluded from compilation (since not all endpoints need this). This implementation, if present, MUST comply with RFC 1123.	
9.	[From and To Header] When the URI in From/To header is not enclosed with "<" and ">", all parameters belong to the header, not the URI.	
10.	[Max-Forwards] The library MUST provide Max-Forwards header field.	
11.	[Proxy-Authenticate, Proxy-Authorization] These header fields MUST be implemented.	
12.	[Proxy-Require] This header field MUST be implemented.	
13.	[Record-Route] This header field MUST be implemented.	
14.	[Require] This header field MUST be implemented.	
15.	[Route] This header field MUST be implemented.	
16.	[Supported] This header field MUST be implemented.	
17.	[Timestamp] This header field SHOULD be implemented, since UAS MUST put a Timestamp header field in the response if that header is present in the request, with recommendation to add delay value. However the implementation should be able to be excluded from compilation.	
18.	[Unsupported] This header field MUST be implemented.	
19.	[Via] This header field MUST be implemented. The implementation MUST allow LWS to be present on either side of "/" character for the SIP version.	
20.	[WARNING] This header SHOULD be implemented.	

	However the implementation should be able to be excluded from compilation.	
21.	[WWW-Authenticate] This header field MUST be implemented.	

2.2.6 Parser

Item	Case Description	Status
1.	Implementations processing SIP messages over stream-oriented transports MUST ignore any CRLF appearing before the start-line [RFC 3261, Section 7.5].	

2.2.7 Message Bodies

This section provides core requirements from RFC 3261 for handling SIP message bodies. Further chapters in this document deals with the specific requirements in PJSIP for advanced message bodies handling such as compression, encryption, etc.

Item	Case Description	Status
1.	The Internet media type of the message body MUST be given by the **Content-Type** header field. [RFC 3261 Section 7.4.1]	
2.	If the body has undergone any encoding such as compression, then this MUST be indicated by the **Content-Encoding** header field; otherwise, **Content-Encoding** MUST be omitted. [RFC 3261 Section 7.4.1]	

2.2.8 UAC Generating Requests

Item	Case Description	Status
1.	[Creating Request from URI [RFC 3261 section 19.1.5]] An implementation MUST include any provided transport, maddr, ttl, or user parameter in the Request-URI of the formed request. If the URI contains a method parameter, its value MUST be used as the method of the request. The method parameter MUST NOT be placed in the Request-URI. Unknown URI parameters MUST be placed in the message's Request-URI. An implementation SHOULD treat the presence of any headers or body parts in the URI as a desire to include them in the message, and choose to honor the request on a per-component basis. An implementation SHOULD NOT honor these obviously dangerous header fields: From, Call-ID, CSeq, Via, Record-Route, and Route, and headers that can falsely advertise its	

	location or capabilities, such as Accept, Accept-Encoding, Accept-Language, Allow, Contact (in its dialog usage), Organization, Supported, and User-Agent. An implementation SHOULD verify the accuracy of any requested descriptive header fields, including: Content-Disposition, Content-Encoding, Content-Language, Content-Length, Content-Type, Date, Mime-Version, and Timestamp.	
2.	A valid SIP request formulated by a UAC MUST, at a minimum, contain the following header fields: To, From, CSeq, Call-ID, Max-Forwards, and Via.	
3.	From header field MUST always have tag.	
4.	Via header MUST contain **branch** parameter.	
5.	Application (i.e. TU) SHOULD be able to provide its own customized **branch** parameter (e.g. for proxy detection).	
6.	The **Contact** header MUST be present for requests that establish dialog.	
7.	If the **Request-URI** or top **Route** header field value contains a SIPS URI, the **Contact** header field MUST contain a SIPS URI as well.	

[Ref: RFC 3261 Section 8.1.1]

2.2.9 UAS Generating Responses

Item	Case Description	Status
1.	These headers must be copied as is to the response: From, To, Call-ID, CSeq, Via (maintaining the order).	
2.	When a 100 (Trying) response is generated, any Timestamp header field present in the request MUST be copied into this 100 (Trying) response, adding delay value if necessary.	

2.3 Endpoint

Endpoint is a term that is defined in PJSIP, for entity that manages transactions, transports, etc.

Item	Case Description	Status
1.	For incoming requests, the stack MUST allow application to choose whether to handle the request in statefull or stateless manner. This also applies for requests that belong to a dialog.	
2.	UAS SHOULD process the requests in the order of the steps that follow in section 8.2 of RFC 3261 (that is, starting with authentication, then inspecting the method, the header fields, and so on throughout the remainder of that section).	

3.	The stack MUST know what SIP methods it supports. If the UAS recognizes but does not support the method of a request, it MUST generate a 405 (Method Not Allowed) response, along with Allow header.	
4.	[Merged Requests] If the request has no tag in the To header field, the UAS core MUST check the request against ongoing transactions. If the From tag, Call-ID, and CSeq exactly match those associated with an ongoing transaction, but the request does not match that transaction (based on the matching rules in Section 17.2.3), the UAS core SHOULD generate a 482 (Loop Detected) response and pass it to the server transaction.	
5.	[Stray Response] If the response doesn't match any transactions, the response MUST be passed to the core (whether it be stateless proxy, stateful proxy, or UA) for further processing. Handling of these "stray" responses is dependent on the core (a proxy will forward them, while a UA will discard, for example). [Section 18.1.2]	

2.4 Transactions

2.4.1 Design Requirements

Item	Case Description	Status
1.	Transaction API MUST be generic so it can be used by both user agents and proxies.	

2.4.2 Client Transactions

Ref: RFC 3261 Section 17.1

Item	Case Description	Status
1.	UA starts the transaction by passing up the SIP request message and IP address, port, and transport to send the message.	
2.	MUST have mechanism to insert own generated branch parameter (example: for proxy loop detection).	
3.	MUST have configurable timer values.	
4.	MUST NOT retransmit request for reliable transports.	
5.	[Non INVITE] After sending request, the client transaction SHOULD set timeout timer to fire in 64*T1 seconds.	
6.	[For INVITE] The retransmission timer doubles every transmission, and doesn't cap off. [Non INVITE] Requests are retransmitted at an interval which	

	starts at T1 and doubles until it hits T2.	
7.	[Matching Response to Client Transaction] Response is matched to a client transaction if: ○ The response has the same value of the branch parameter in the top Via header field as the branch parameter in the top Via header field of the request that created the transaction. ○ The method parameter in the CSeq header field matches the method of the request that created the transaction. The method is needed since a CANCEL request constitutes a different transaction, but shares the same value of the branch parameter.	
8.	[For INVITE] The provisional response MUST be passed to the TU. Any further provisional responses MUST be passed up to the TU while in the "Proceeding" state. [Non INVITE] If a *provisional response* is received, retransmissions continue for unreliable transports, but at an interval of T2 (The server transaction retransmits the last response it sent, which can be a provisional or final response, only when a retransmission of the request is received).	
9.	[For INVITE] Upon receiving 300-699 response, MUST pass the received response up to the TU, and MUST generate an ACK request. The ACK MUST be sent to the same address, port, and transport to which the original request was sent.	
10.	[For INVITE] The client transaction SHOULD start timer D when it enters the "Completed" state, with a value of at least 32 seconds for unreliable transports, and a value of zero seconds for reliable transports. [Non INVITE] Once the client transaction enters the "Completed" state, it MUST set Timer K to fire in 5 seconds (T4) for unreliable transports, and zero seconds for reliable transports.	
11.	[For INVITE] Any retransmissions of the final response that are received while in the "Completed" state MUST cause the ACK to be re-passed to the transport layer for retransmission.	
12.	[For INVITE] Reception of a 2xx response MUST cause the client transaction to enter the "Terminated" state, and the response MUST be passed up to the TU.	
13.	[Transport Error] When there is an error in sending the request, the client transaction SHOULD inform the TU that a transport failure has occurred, and the client transaction SHOULD transition directly to the "Terminated" state. The TU will handle the failover mechanisms described in [4].	

2.4.3 Generating ACK (for 3xx-6xx Responses)

Ref: RFC 3261 Section 17.1.1

Item	Case Description	Status
1.	Header fields Call-ID, From, and Request-URI MUST be equal to the original INVITE request.	
2.	The To header field in the ACK MUST equal the To header field in the response being acknowledged (including tag).	
3.	The ACK MUST contain a single Via header field, and this MUST be equal to the top Via header field of the original request.	
4.	The CSeq header field in the ACK MUST contain the same value for the sequence number as was present in the original request, but the method parameter MUST be equal to "ACK".	
5.	If the INVITE request whose response is being acknowledged had Route header fields, those header fields MUST appear in the ACK.	
6.	[Message Body] Although any request MAY contain a body, a body in an ACK is special since the request cannot be rejected if the body is not understood. Therefore, placement of bodies in ACK for non-2xx is NOT RECOMMENDED , but if done, the body types are restricted to any that appeared in the INVITE, assuming that the response to the INVITE was not 415. If it was, the body in the ACK MAY be any type listed in the Accept header field in the 415.	

2.4.4 Server Transactions

Ref: RFC 3261 Section 17.2

Item	Case Description	Status
1.	[Matching Requests for Server Transactions] If it is present and begins with the magic cookie "z9hG4bK", the request matches a transaction if: o The branch parameter in the request is equal to the one in the top Via header field of the request that created the transaction, and o The sent-by value in the top Via of the request is equal to the one in the request that created the transaction, and o The method of the request matches the one that	

	created the transaction, except for ACK, where the method of the request that created the transaction is INVITE. If the magic cookie is not present, the following rules are used. ○ The INVITE request matches a transaction if the Request-URI, To tag, From tag, Call-ID, CSeq, and top Via header field match those of the INVITE request which created the transaction. ○ The ACK request matches a transaction if the Request-URI, From tag, Call-ID, CSeq number (not the method), and top Via header field match those of the INVITE request which created the transaction, and the To tag of the ACK matches the To tag of the response sent by the server transaction. ○ An ACK request that matches an INVITE transaction matched by a previous ACK is considered a retransmission of that previous ACK. ○ For all other request methods, a request is matched to a transaction if the Request-URI, To tag, From tag, Call-ID, CSeq (including the method), and top Via header field match those of the request that created the transaction.	
2.	[For INVITE] MUST NOT retransmit 2xx response for reliable transports.	
3.	[Provisional Response] [For INVITE] The server transaction MUST generate a 100 (Trying) response unless it knows that the TU will generate a provisional or final response within 200 ms, in which case it MAY generate a 100 (Trying) response. [Non INVITE] A SIP element MUST NOT send any provisional response with a Status-Code other than 100 to a non-INVITE request. A SIP element MUST NOT respond to a non-INVITE request with a Status-Code of 100 over any unreliable transport, such as UDP, before the amount of time it takes a client transaction's Timer E to be reset to T2. A SIP element MAY respond to a non-INVITE request with a Status-Code of 100 over a reliable transport at any time. Without regard to transport, a SIP element MUST respond to a non-INVITE request with a Status-Code of 100 if it has not otherwise responded after the amount of time it takes a client transaction's Timer E to be reset to T2. A transaction-stateful SIP element MUST NOT send a response with Status-Code of 408 to a non-INVITE request.	

	As a consequence, an element that can not respond before the transaction expires will not send a final response at all. [draft-sparks-sip-nit-actions-03]	
4.	If a request retransmission is received while in the "Proceeding" state, the most recent provisional response that was received from the TU MUST be passed to the transport layer for retransmission.	
5.	[2xx Response for INVITE] If, while in the "Proceeding" state, the TU passes a 2xx response to the server transaction, the server transaction MUST pass this response to the transport layer for transmission. It is not retransmitted by the server transaction; retransmissions of 2xx responses are handled by the TU. The server transaction MUST then transition to the "Terminated" state.	
6.	[2xx Response to INVITE] After 2xx response is passed to UAS transaction, the transaction MUST NOT be destroyed immediately, but instead must be kept in "lingering" state where it would pass INVITE request retransmission to UA layer. (If the UAS transaction is destroyed immediately, incoming INVITE request retransmission will cause the core to recreate the UAS transaction).	
7.	[300-699 Response for INVITE] While in the "Proceeding" state, if the TU passes a response with status code from 300 to 699 to the server transaction, the response MUST be passed to the transport layer for transmission, and the state machine MUST enter the "Completed" state. For unreliable transports, retransmission timer (timer G) is set to fire in T1 seconds, and is not set to fire for reliable transports. Timer G caps on T2 seconds. [All Final Responses for Non INVITE] If the TU passes a final response (status codes 200-699) to the server while in the "Proceeding" state, the transaction MUST enter the "Completed" state, and the response MUST be passed to the transport layer for transmission.	
8.	[For INVITE] When the "Completed" state is entered, timer H (timeout timer) MUST be set to fire in 64*T1 seconds for all transports. [Non INVITE] When the server transaction enters the "Completed" state, it MUST set Timer J to fire in 64*T1 seconds for unreliable transports, and zero seconds for reliable transports. Any other final responses passed by the TU to the server transaction MUST be discarded while in the "Completed" state.	

9.	[For INVITE] If an ACK is received while the server transaction is in the "Completed" state, the server transaction MUST transition to the "Confirmed" state.	
10.	[For INVITE] If the ACK was never received, the server transaction MUST transition to the "Terminated" state, and MUST indicate to the TU that a transaction failure has occurred.	
11.	[Transport Error] The procedures in [RFC 3263] are followed, which attempt to deliver the response to a backup. If those should all fail, based on the definition of failure in [RFC 3263], the server transaction SHOULD inform the TU that a failure has occurred, and SHOULD transition to the terminated state.	

2.5 Transports

Reference:

- RFC 3261 (SIP)
- RFC 3581 (rport extension)

2.5.1 Design Requirements

Item	Case Description	Status
1.	MUST provide mechanism to add new types of transports.	
2.	MUST provide hook to plug certain type of process for each outgoing/incoming packets/messages. This hook mechanism is used for extensions that work on message level, such as SigComp.	
3.	MUST be able to support **Ipv6**, and this support MUST be configurable (e.g. can be excluded from compilation).	
4.	MUST be able to support TLS (configurable).	

2.5.2 Computing Destination of the Request

Item	Case Description	Status
1.	MUST follow the rules in Section 8.1.2 of RFC 3261.	

2.5.3 Behavior

Item	Case Description	Status
1.	[Large Request]	

	If a request is within 200 bytes of the path MTU, or if it is larger than 1300 bytes and the path MTU is unknown, the request MUST be sent using an RFC 2914 [36] congestion controlled transport protocol, such as TCP. If this causes a change in the transport protocol from the one indicated in the top Via, the value in the top Via MUST be changed. If **this** attempt (and only for this cause) to establish the connection generates either an ICMP Protocol Not Supported, or results in a TCP reset, the element SHOULD retry the request, using UDP (for backward compatibility).	
2.	[Multicast] A client that sends a request to a multicast address MUST add the maddr parameter to its Via header field value containing the destination multicast address, and for IPv4, SHOULD add the ttl parameter with a value of 1.	
3.	**[UDP Response Address]** Server chooses the address to send response: o From the host part in Via *sent-by* if **received** parameter is not present. o From source address of the request if **received** parameter is present.	
4.	**[UDP Response Port]** Server chooses the port number to send response: o From the port in Via *sent-by* if **rport** parameter is not present. o From source port of the request if **rport** parameter is present.	
5.	Transport layer adds **received** and **rport** parameter for each outbound request [ref:3261,3581].	
6.	Transport layer MUST fill the **received** parameter for each incoming requests.	
7.	Transport layer MUST fill in the **rport** parameter for each outgoing requests.	
8.	[Receiving Response] If the value of the sent-by parameter in that header field value does not correspond to a value that the client transport is configured to insert into requests, the response MUST be silently discarded. [Section 18.1.2]	
9.	For backward compatibility, transport MUST still be prepared to receive responses from the address as advertised in Via *sent-by*. [ref:3581]	

10.	[Stray Response] If the response doesn't match any transactions, the response MUST be passed to the core (whether it be stateless proxy, stateful proxy, or UA) for further processing. Handling of these "stray" responses is dependent on the core (a proxy will forward them, while a UA will discard, for example). [Section 18.1.2]	
11.	**[TCP]** Server must send response using existing connection.	
12.	**[TCP]** Section 18.2.2: For server, when existing connection has closed, it SHOULD open a new connection to the address in **received** parameter, using the port in **sent-by** if present or default port.	
13.	**[TCP]** Section 18.2.2: If above procedure fails, server SHOULD use procedure in RFC 3263 to open new connection to send response to.	
14.	**[TCP]** Section 17.2.4: If above procedure fails, transaction should inform TU that a failure has occurred, and progress the state to terminated.	
15.	**[Invalid Message]** If the message is response, it MUST be discarded. If the message is request, the element SHOULD generate 400 (Bad Request) response.	
16.	**[ICMP Errors for Unreliable Transports]** Host, network, port or protocol unreachable errors, or parameter problem errors SHOULD cause the transport layer to inform the transport user of a failure in sending. Source quench and TTL exceeded ICMP errors SHOULD be ignored.	

2.6 Resolver

Resolver is the element who resolves the destination host/port based on the rule set in RFC 3263. The table below outlines the requirements for this element.

Item	Case Description	Status
1.	The resolver MUST provide mechanism to be used by any elements in the stack (e.g. transactions, user agents, proxies, etc.)	

2.7 Digest Authentication

2.7.1 Basic Case

Item	Case Description	Status
1.	When no credential is supplied, forward all authorization failures to application.	
2.	Responding to digest authentication with **qop=none**	
3.	**[PJSIP_AUTH_QOP_SUPPORT==1]** Responding to digest authentication with **qop="auth,auth-int"**	
4.	**[PJSIP_AUTH_QOP_SUPPORT==0]** Report authentication failure to application when **qop="auth,auth-int"** is requested.	
5.	Report authentication failure when: - scheme is not supported - digest algorithm is not supported	

2.7.2 Next Requests

Item	Case Description	Status
1.	**[PJSIP_AUTH_AUTO_SEND_NEXT==0]** New requests MUST NOT have any Authorization headers.	
2.	**[PJSIP_AUTH_AUTO_SEND_NEXT==1]** When **qop=none**, if next request has different method, send brand new Authorization header.	
3.	**[PJSIP_AUTH_AUTO_SEND_NEXT==1]** When **qop=auth**, always send fresh Authorization header. No header should be put in the cache.	
4.	**[PJSIP_AUTH_HEADER_CACHING==1]** When **qop=none**, if endpoint has sent request with the same method, resent cached authorization header.	
5.	When received 401/407 with **stale=1**, recalculate new Authorization header with the new nonce.	
6.	When received 401/407 with **stale=0**, inform user about authorization failure.	

2.7.3 Multiple Challenges

Item	Case Description	Status
1.	When multiple WWW-Authenticate/Proxy-Authenticate headers are present (with different realm), client should respond to each of them automatically (as long as they have the credential)	
2.	When authorization failed with new challenge from downstream proxy and client retries new request, the stack should include all Authorization headers sent in the previous request.	

2.7.4 Open Issues

Item	Case Description	Status
1.	When one server sends multiple WWW-Authenticate with different algorithm, client should select one with the highest level of encryption that it supports.	N/A

Chapter 3:User Agent/Dialog Layer

References:

- RFC 3261
- RFC 3515 (REFER)
- RFC 3265 (events)

3.1 Design Requirements

Item	Case Description	Status
1.	Must be able to handle forking.	
2.	Dialog can be created by several methods (e.g. INVITE, SUBSCRIBE, REFER).	
3.	Extensible; future methods may create dialog.	
4.	Feature extensible; extensions can be plugged in to the dialog framework (e.g. PRACK) along with their Supported/Require header flag indication.	
5.	Multiple API level: basic, low level API which common for all kind of dialog usage, and higher level API for manipulating feature specific extension.	
6.	Different API should provide different callbacks too, which are specific to the features.	
7.	Extensions can work together on a single dialog (e.g. event subscription inside a dialog created by INVITE).	

3.2 UAC Behavior

Item	Case Description	Status
1.	MUST provide a SIP or SIPS URI with global scope in the Contact header field of the request.	
2.	If the request has a Request-URI or a topmost Route header field value with a SIPS URI, the Contact header field MUST contain a SIPS URI.	
3.	If the request was sent over TLS, and the Request-URI contained a SIPS URI, the "secure" flag is set to TRUE.	
4.	The route set MUST be set to the list of URIs in the Record-Route header field from the response, taken in reverse order and preserving all URI parameters. If no Record-Route header field is present in the response, the route set MUST be set to the empty set. This route set, even if empty, overrides any pre-existing route set for future requests in this dialog.	

5.	The remote target MUST be set to the URI from the Contact header field of the response.	
6.	The local sequence number MUST be set to the value of the sequence number in the CSeq header field of the request. The remote sequence number MUST be empty (it is established when the remote UA sends a request within the dialog).	
7.	The call identifier component of the dialog ID MUST be set to the value of the Call-ID in the request.	
8.	The local tag component of the dialog ID MUST be set to the tag in the From field in the request, and the remote tag component of the dialog ID MUST be set to the tag in the To field of the response. A UAC MUST be prepared to receive a response without a tag in the To field, in which case the tag is considered to have a value of null.	
9.	The remote URI MUST be set to the URI in the To field, and the local URI MUST be set to the URI in the From field.	
10.	Requests within a dialog MAY contain Record-Route and Contact header fields. However, these requests do not cause the dialog's route set to be modified, although they may modify the remote target URI. Specifically, requests that are not target refresh requests do not modify the dialog's remote target URI, and requests that are target refresh requests do. Target refresh requests only update the dialog's remote target URI, and not the route set formed from the Record-Route.	
11.	If the route set is not empty, and the first URI in the route set contains the lr parameter (see Section 19.1.1), the UAC MUST place the remote target URI into the Request-URI and MUST include a Route header field containing the route set values in order, including all parameters.	
12.	If the route set is not empty, and its first URI does not contain the lr parameter, the UAC MUST place the first URI from the route set into the Request-URI, stripping any parameters that are not allowed in a Request-URI. The UAC MUST add a Route header field containing the remainder of the route set values in order, including all parameters. The UAC MUST then place the remote target URI into the Route header field as the last value.	
13.	A UAC SHOULD include a Contact header field in any target refresh requests within a dialog, and unless there is a need to change it, the URI SHOULD be the same as used in previous requests within the dialog. If the "secure" flag is true, that URI MUST be a SIPS URI.	
14.	When a UAC receives a 2xx response to a target refresh request, it MUST replace the dialog's remote target URI with the URI from the Contact header field in that response, if	

	present.	
15.	MUST NOT send CANCEL before receiving provisional response. The transmission of CANCEL request MAY be delayed until provisional response is received.	
16.	UAC canceling a request cannot rely on receiving a 487 (Request Terminated) response for the original request, as an RFC 2543-compliant UAS will not generate such a response. If there is no final response for the original request in 64*T1 seconds (T1 is defined in Section 17.1.1), the client SHOULD then consider the original transaction cancelled and SHOULD destroy the client transaction handling the original request.	
17.	If the response for a request within a dialog is a 481 (Call/Transaction Does Not Exist) or a 408 (Request Timeout), the UAC SHOULD terminate the dialog. A UAC SHOULD also terminate a dialog if no response at all is received for the request (the client transaction would inform the TU about the timeout.)	
18.	Independent of the method, if a request outside of a dialog generates a non-2xx final response, any early dialogs created through provisional responses to that request are terminated.	
19.	When Expires header is present, UAC core SHOULD generate CANCEL after the timeout expires [Ref: RFC 3261 Section 13.2.1].	
20.	ACK MUST be sent to the new target URI (the one in Contact header in 2xx response to INVITE). RFC 3261 says that the request is passed to the transport layer directly for transmission, rather than a client transaction.	
21.	UAC MUST NOT initiate a new INVITE transaction within a dialog while another INVITE transaction is in progress in either direction.	
22.	UA MAY initiate a regular transaction while an INVITE transaction is in progress. A UA MAY also initiate an INVITE transaction while a regular transaction is in progress.	
23.	If a UAC receives a 491 (Request Pending, or glare) response to a re-INVITE, it SHOULD start a timer with a value T chosen as follows: ○ If the UAC is the owner of the Call-ID of the dialog ID (meaning it generated the value), T has a randomly chosen value between 2.1 and 4 seconds in units of 10 ms. ○ If the UAC is not the owner of the Call-ID of the dialog ID, T has a randomly chosen value of between 0 and 2 seconds in units of 10 ms.	
24.	The UAC MUST consider the session terminated (and therefore stop sending or listening for media) as soon as the BYE request is passed to the client transaction. If the	

	response for the BYE is a 481 (Call/Transaction Does Not Exist) or a 408 (Request Timeout) or no response at all is received for the BYE (that is, a timeout is returned by the client transaction), the UAC MUST consider the session and the dialog terminated [Section 15.1.1].	
25.	Endpoints MUST NOT use a URI obtained from a Record-Route header field outside the dialog in which it was provided.	

[Ref: RFC 3261 Chapter 12]

3.3 UAS Behavior

Item	Case Description	Status
1.	When responding to a request with a response that establishes a dialog (such as a 2xx to INVITE), the UAS MUST copy exactly all **Record-Route** header field values from the request into the response and MUST maintain the order of those values. The UAS MUST add a **Contact** header field to the response. The URI provided in the Contact header field MUST be a SIP or SIPS URI. If the request that initiated the dialog contained a SIPS URI in the Request-URI or in the top Record-Route header field value, if there was any, or the Contact header field if there was no Record-Route header field, the Contact header field in the response MUST be a SIPS URI. The URI SHOULD have global scope (that is, the same URI can be used in messages outside this dialog). [Ref: RFC 3261 Section 21.1.1]	
2.	If the request arrived over TLS, and the Request-URI contained a SIPS URI, the "secure" flag is set to TRUE.	
3.	The route set MUST be set to the list of URIs in the Record-Route header field from the request, taken in order and preserving all URI parameters. If no Record-Route header field is present in the request, the route set MUST be set to the empty set. This route set, even if empty, overrides any pre-existing route set for future requests in this dialog.	
4.	The remote target MUST be set to the URI from the Contact header field of the request.	
5.	The remote URI MUST be set to the URI in the From field, and the local URI MUST be set to the URI in the To field.	
6.	The remote sequence number MUST be set to the value of the sequence number in the CSeq header field of the request. The local sequence number MUST be empty.	
7.	The call identifier component of the dialog ID MUST be set to	

	the value of the Call-ID in the request.	
8.	The local tag component of the dialog ID MUST be set to the tag in the To field in the response to the request (which always includes a tag), and the remote tag component of the dialog ID MUST be set to the tag from the From field in the request. A UAS MUST be prepared to receive a request without a tag in the From field, in which case the tag is considered to have a value of null.	
9.	UAS MUST adhere to **Expire** header field. If the invitation expires before the UAS has generated a final response, a 487 (Request Terminated) response SHOULD be generated.	
10.	Dialog must retransmit provisional response at least every **1 minute**, to ask for an "extension" in order to prevent proxies from canceling the transaction. A proxy has the option of canceling a transaction when there is a gap of 3 minutes between responses in a transaction.	
11.	Contact header field in a target refresh request updates the remote target URI.	
12.	If the remote sequence number was not empty, but the sequence number of the request is lower than the remote sequence number, the request is out of order and MUST be rejected with a 500 (Server Internal Error) response.	
13.	It is possible for the CSeq sequence number to be higher than the remote sequence number by more than one. This is not an error condition, and a UAS SHOULD be prepared to receive and process requests with CSeq values more than one higher than the previous received request. The UAS MUST then set the remote sequence number to the value of the sequence number in the CSeq header field value in the request (e.g. proxy challenged previous request).	
14.	When a UAS receives a target refresh request, it MUST replace the dialog's remote target URI with the URI from the Contact header field in that request, if present.	
15.	A UAS rejecting an offer contained in an INVITE SHOULD return a 488 (Not Acceptable Here) response. Such a response SHOULD include a Warning header field value explaining why the offer was rejected.	
16.	Redirect response SHOULD contain a Contact header field containing one or more URIs of new addresses to be tried.	
17.	The INVITE server transaction will be destroyed as soon as it receives 2xx final response and passes it to the transport. Therefore, it is necessary to periodically pass the response directly to the transport until the ACK arrives. This is done even with reliable transport.	
18.	When no ACK is received after 61*T1 seconds, the session SHOULD be terminated by sending BYE.	

19.	A UAS that receives a second INVITE before it sends the final response to a first INVITE with a lower CSeq sequence number on the same dialog MUST return a 500 (Server Internal Error) response to the second INVITE and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds [Section 14.2].	
20.	A UAS that receives an INVITE on a dialog while an INVITE it had sent on that dialog is in progress MUST return a 491 (Request Pending) response to the received INVITE [Section 14.2].	
21.	If a UA receives a re-INVITE for an existing dialog, it MUST check any version identifiers in the session description or, if there are no version identifiers, the content of the session description to see if it has changed. If the session description has changed, the UAS MUST adjust the session parameters accordingly, possibly after asking the user for confirmation [Section 14.2].	
22.	For Re-INVITE, if a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a BYE to terminate the dialog [Section 14.2].	
23.	If the BYE does not match an existing dialog, the UAS core SHOULD generate a 481 (Call/Transaction Does Not Exist) response and pass that to the server transaction [Section 15.1.2].	
24.	After receiving BYE, UAS MUST still respond to any pending requests received for that dialog. It is RECOMMENDED that a 487 (Request Terminated) response be generated to those pending requests [Section 15.1.2].	

[Ref: RFC 3261 Chapter 12]

3.4 Application Consideration

Item	Case Description	Status
1.	[Receiving INVITE] Application SHOULD check incoming INVITE for the same Call-Id with existing dialogs. It should be able to detect forking INVITE requests.	

Chapter 4:Redirection

The core stack provides auxiliary API for processing redirection (3xx response). Below are the requirements for such API, following the rules in RFC 3261 Section 8.1.3.

Item	Case Description	Status
1.	Use the URI(s) in the Contact header field to formulate one or more new requests based on the redirected request.	
2.	MUST NOT add any given URI to the target set more than once.	
3.	If the original request had a SIPS URI in the Request-URI, the client MAY choose to recurse to a non-SIPS URI, but SHOULD inform the user of the redirection to an insecure URI.	
4.	MAY reorder the Contacts by the q value.	

Chapter 5:Proxy Behavior

5.1 Request Validation

Item	Case Description	Status
1.	Any components that are NOT required by proxy processing logic, well-formed or not, SHOULD be ignored and remain unchanged when the message is forwarded. For instance, an element would not reject a request because of a malformed Date header field. Likewise, a proxy would not remove a malformed Date header field before forwarding a request.	
2.	If the **Request-URI** has a URI whose scheme is not understood by the proxy, the proxy SHOULD reject the request with a 416 (Unsupported URI Scheme) response.	
3.	If the request does not contain a **Max-Forwards** header field, this check is passed. If the request contains a **Max-Forwards** header field with a field value greater than zero, the check is passed. If the request contains a **Max-Forwards** header field with a field value of zero (0), the element MUST NOT forward the request. If the request was for OPTIONS, the element MAY act as the final recipient and respond the request. Otherwise, the element MUST return a 483 (Too many hops) response.	
4.	An element MAY check for forwarding loops before forwarding a request.	
5.	If the request contains a **Proxy-Require** header field with one or more option-tags this element does not understand, the element MUST return a 420 (Bad Extension) response. The response MUST include an Unsupported header field listing those option-tags the element did not understand.	
6.	If an element requires credentials before forwarding a request, the request MUST be inspected.	

[Ref: RFC 3261 Section 16.3]

5.2 Route Processing

Item	Case Description	Status
1.	The proxy MUST inspect the Request-URI of the request. If the Request-URI of the request contains a value this proxy previously placed into a Record-Route header field (see Section 16.6 item 4), the proxy MUST replace the Request-URI in the request with the last value from the Route header field, and remove that value from the Route header field. The proxy MUST then proceed as if it received this modified request.	

2.	If the Request-URI contains a maddr parameter, the proxy MUST check to see if its value is in the set of addresses or domains the proxy is configured to be responsible for. If the Request-URI has a maddr parameter with a value the proxy is responsible for, and the request was received using the port and transport indicated (explicitly or by default) in the Request-URI, the proxy MUST strip the maddr and any non-default port or transport parameter and continue processing as if those values had not been present in the request. A request may arrive with a maddr matching the proxy, but on a port or transport different from that indicated in the URI. Such a request needs to be forwarded to the proxy using the indicated port and transport.	
3.	If the first value in the Route header field indicates this proxy, the proxy MUST remove that value from the request.	

[Ref: RFC 3261 Section 16.4]

5.3 Calculating Request Target

Item	Case Description	Status
1.	If the Request-URI of the request contains an maddr parameter, the Request-URI MUST be placed into the target set as the only target URI, and the proxy MUST begin forwarding the request.	
2.	If the domain of the Request-URI indicates a domain this element is not responsible for, the Request-URI MUST be placed into the target set as the only target, and the element MUST proceed to the task of Request Forwarding	
3.	If the Request-URI does not provide sufficient information for the proxy to determine the target set, it SHOULD return a 485 (Ambiguous) response. This response SHOULD contain a Contact header field containing URIs of new addresses to be tried.	
4.	A proxy MUST NOT add additional targets to the target set if the Request-URI of the original request does not indicate a resource this proxy is responsible for.	
5.	If the Request-URI indicates a resource at this proxy that does not exist, the proxy MUST return a 404 (Not Found) response.	
6.	If the target set remains empty after applying all of the above, the proxy MUST return an error response, which SHOULD be the 480 (Temporarily Unavailable) response.	

[Ref: RFC 3261 Section 16.5]

5.4 Request Forwarding

Item	Case Description	Status
1.	A stateful proxy must have a mechanism to maintain the target set as responses are received and associate the responses to each forwarded request with the original request. For the purposes of this model, this mechanism is a "response context" created by the proxy layer before forwarding the first request.	
2.	For each target, the proxy forwards the request following these steps: 1. Make a copy of the received request 2. Update the Request-URI 3. Update the Max-Forwards header field (if the copy does not contain a Max-Forwards header field, the proxy MUST add one with a field value, which SHOULD be 70). 4. Optionally add a Record-route header field value 5. Optionally add additional header fields 6. Post-process routing information 7. Determine the next-hop address, port, and transport 8. Add a Via header field value 9. Add a Content-Length header field if necessary 10. Forward the new request 11. Set timer C	
3.	[Copying the Request] The copy MUST initially contain all of the header fields from the received request. Fields not detailed in the processing described below MUST NOT be removed. The copy SHOULD maintain the ordering of the header fields as in the received request. The proxy MUST NOT reorder field values with a common field name.	
4.	[Request-URI] The Request-URI in the copy's start line MUST be replaced with the URI for this target. If the URI contains any parameters not allowed in a Request-URI, they MUST be removed.	
5.	[Max-Forwards] If the copy contains a Max-Forwards header field, the proxy MUST decrement its value by one (1). If the copy does not contain a Max-Forwards header field, the proxy MUST add one with a field value, which SHOULD be 70.	

6.	[Record-Route] If this proxy wishes to remain on the path of future requests in a dialog created by this request (assuming the request creates a dialog), it MUST insert a Record-Route header field value into the copy before any existing Record-Route header field values, even if a Route header field is already present.	
7.	[Record-Route] If this request is already part of a dialog, the proxy SHOULD insert a Record-Route header field value if it wishes to remain on the path of future requests in the dialog. In normal endpoint operation as described in Section 12, these Record-Route header field values will not have any effect on the route sets used by the endpoints. A proxy MAY insert a Record-Route header field value into any request. The URI placed in the Record-Route header field value MUST be a SIP or SIPS URI. This URI MUST contain an lr parameter. The URI SHOULD NOT contain the transport parameter unless the proxy has knowledge (such as in a private network) that the next downstream element that will be in the path of subsequent requests supports that transport. If the Request-URI contains a SIPS URI, or the topmost Route header field value (after the post processing of Route headers) contains a SIPS URI, the URI placed into the Record-Route header field MUST be a SIPS URI. Furthermore, if the request was not received over TLS, the proxy MUST insert a Record-Route header field. In a similar fashion, a proxy that receives a request over TLS, but generates a request without a SIPS URI in the Request-URI or topmost Route header field value (after the post processing of Route header), MUST insert a Record-Route header field that is not a SIPS URI. A proxy at a security perimeter must remain on the perimeter throughout the dialog. If the URI placed in the Record-Route header field needs to be rewritten when it passes back through in a response, the URI MUST be distinct enough to locate at that time. (The request may spiral through this proxy, resulting in more than one Record-Route header field value being added). The proxy MAY include parameters in the Record-Route header field value. These will be echoed in some responses to the request such as the 200 (OK) responses to INVITE. Such parameters may be useful for keeping state in the message rather than the proxy. If a proxy needs to be in the path of any type of dialog (such as one straddling a firewall), it SHOULD add a Record-Route	

	header field value to every request with a method it does not understand since that method may have dialog semantics.	
8.	[Route] A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. This set MUST be pushed into the Route header field of the copy ahead of any existing values, if present. If the Route header field is absent, it MUST be added, containing that list of URIs. A proxy MUST ensure that all such proxies are loose routers. Furthermore, if the Request-URI contains a SIPS URI, TLS MUST be used to communicate with that proxy.	
9.	[Route] If the copy contains a Route header field, the proxy MUST inspect the URI in its first value. If that URI does not contain an lr parameter, the proxy MUST modify the copy as follows: ○ The proxy MUST place the Request-URI into the Route header field as the last value. ○ The proxy MUST then place the first Route header field value into the Request-URI and remove that value from the Route header field.	
10.	[Determine Next-Hop Address, Port, and Transport] The proxy applies the procedures listed in [RFC 3263: Locating SIP Servers] as follows to determine where to send the request.	
11.	[Add a Via header field value] MAY add customized branch parameter if the proxy wants to perform loop detection. A proxy choosing to detect loops SHOULD create a branch parameter separable into two parts by the implementation.	
12.	[Add a Content-Length header field if necessary] If the request will be sent to the next hop using a stream-based transport and the copy contains no Content-Length header field, the proxy MUST insert one with the correct value for the body of the request	
13.	Forward request to each target. For each target, a new client transaction is created (for stateful proxy).	
14.	[Start Timer C] In order to handle the case where an INVITE request never generates a final response, the TU uses a timer which is called timer C. Timer C MUST be set for each client transaction when an INVITE request is proxied. The timer MUST be larger than 3 minutes.	

[Ref: RFC 3261 Section 16.6]

5.5 Processing Response

Item	Case Description	Status
1.	When a response is received by an element, it first tries to locate a client transaction matching the response. If none is found, the element MUST process the response (even if it is an informational response) as a stateless proxy.	
2.	A transaction-stateful SIP proxy MUST NOT send any response to a non-INVITE request unless it has a matching server transaction that is not in the Terminated state. As a consequence, this proxy will not forward any "late" non-INVITE response [draft-sparks-sip-nit-actions-03.txt].	
3.	[Processing by Client Transaction] As client transactions pass responses to the proxy layer, the following processing MUST take place: o Find the appropriate response context o Update timer C for provisional responses o Remove the topmost Via o Add the response to the response context o Check to see if this response should be forwarded immediately o When necessary, choose the best final response from the response context. If no final response has been forwarded after every client transaction associated with the response context has been terminated, the proxy must choose and forward the “best” response from those it has seen so far. The following processing MUST be performed on each response that is forwarded. It is likely that more than one response to each request will be forwarded: at least each provisional and one final response: o Aggregate authorization header field values if necessary o Optionally rewrite Record-Route header field values o Forward the response o Generate any necessary CANCEL requests	
4.	[Removing Via Header] If no Via header field values remain in the response, the response was meant for this element and MUST NOT be forwarded.	

	This will happen, for instance, when the element generates CANCEL requests	
5.	If the proxy chooses to recurse on any contacts in a 3xx response by adding them to the target set, it MUST remove them from the response before adding the response to the response context. However, a proxy SHOULD NOT recurse to a non-SIPS URI if the Request-URI of the original request was a SIPS URI. If the proxy recurses on all of the contacts in a 3xx response, the proxy SHOULD NOT add the resulting contactless response to the response context.	
6.	Removing the contact before adding the response to the response context prevents the next element upstream from retrying a location this proxy has already attempted.	
7.	If a proxy receives a 416 (Unsupported URI Scheme) response to a request whose Request-URI scheme was not SIP, but the scheme in the original received request was SIP or SIPS (that is, the proxy changed the scheme from SIP or SIPS to something else when it proxied a request), the proxy SHOULD add a new URI to the target set. This URI SHOULD be a SIP URI version of the non-SIP URI that was just tried. As with a 3xx response, if a proxy "recurses" on the 416 by trying a SIP or SIPS URI instead, the 416 response SHOULD NOT be added to the response context.	
8.	Until a final response has been sent on the server transaction, the following responses MUST be forwarded immediately: o Any provisional response other than 100 (Trying) o Any 2xx response	
9.	If a 6xx response is received, it is not immediately forwarded, but the stateful proxy SHOULD cancel all client pending transactions as described in Section 10, and it MUST NOT create any new branches in this context.	
10.	Under the new rules, upon receiving a 6xx, a proxy will issue a CANCEL request, which will generally result in 487 responses from all outstanding client transactions, and then at that point the 6xx is forwarded upstream.	
11.	After a final response has been sent on the server transaction, the following responses MUST be forwarded immediately: o Any 2xx response to an INVITE request	
12.	A stateful proxy MUST NOT immediately forward any other responses. In particular, a stateful proxy MUST NOT forward any 100 (Trying) response. Those responses that are candidates for forwarding later as the "best" response have been gathered as described in step "Add Response to Context".	

13.	Any response chosen for immediate forwarding MUST be processed as described in steps "Aggregate Authorization Header Field Values" through "Record-Route".	
14.	This step, combined with the next, ensures that a stateful proxy will forward exactly one final response to a non-INVITE request, and either exactly one non-2xx response or one or more 2xx responses to an INVITE request.	
15.	[Choosing Best Response] The proxy MUST forward a response from the responses stored in the response context. It MUST choose from the 6xx class responses if any exist in the context. If no 6xx class responses are present, the proxy SHOULD choose from the lowest response class stored in the response context. The proxy MAY select any response within that chosen class. The proxy SHOULD give preference to responses that provide information affecting resubmission of this request, such as 401, 407, 415, 420, and 484 if the 4xx class is chosen.	
16.	[503 Response] A proxy which receives a 503 (Service Unavailable) response SHOULD NOT forward it upstream unless it can determine that any subsequent requests it might proxy will also generate a 503. In other words, forwarding a 503 means that the proxy knows it cannot service any requests, not just the one for the Request-URI in the request which generated the 503. If the only response that was received is a 503, the proxy SHOULD generate a 500 response and forward that upstream. The forwarded response MUST be processed as described in steps "Aggregate Authorization Header Field Values" through "Record-Route".	

[Ref: RFC 3261 Section 16.7]

5.6 Handling Timer C

Item	Case Description	Status
1.	If timer C should fire, the proxy MUST either reset the timer with any value it chooses, or terminate the client transaction. If the client transaction has received a provisional response, the proxy MUST generate a CANCEL request matching that transaction. If the client transaction has not received a provisional response, the proxy MUST behave as if the transaction received a 408 (Request Timeout) response.	

5.7 Transport Error

Item	Case Description	Status
1.	If the transport layer notifies a proxy of an error when it tries to forward a request , the proxy MUST behave as if the forwarded request received a 503 (Service Unavailable) response. If the proxy is notified of an error when forwarding a response, it drops the response. The proxy SHOULD NOT cancel any outstanding client transactions associated with this response context due to this notification.	

[Ref: RFC 3261 Section 16.9]

5.8 Processing CANCEL

Item	Case Description	Status
1.	[Cancel in General] A stateful proxy MAY generate CANCEL requests for pending INVITE client transactions based on the period specified in the INVITE's Expires header field elapsing. However, this is generally unnecessary since the endpoints involved will take care of signaling the end of the transaction.	
2.	[Receiving CANCEL with no Context] If the proxy couldn't find response context for an incoming CANCEL request, the element does not have any knowledge of the request to apply the CANCEL to. It MUST statelessly forward the CANCEL request (it may have statelessly forwarded the associated request previously).	

[Ref: RFC 3261 Section 16.10]

5.9 Stateless Proxy

Item	Case Description	Status
1.	MUST NOT generate its own 100 (Trying) or any other provisional response.	
2.	A stateless proxy MUST validate a request.	
3.	[Choosing Target] A stateless proxy MUST choose one and only one target from the target set. This choice MUST only rely on fields in the message and time-invariant properties of the server. In particular, a retransmitted request MUST be forwarded to the same destination each time it is processed. Furthermore, CANCEL and non-Routed ACK requests MUST generate the same choice as their associated INVITE.	
4.	[Branch parameter] The requirement for unique branch IDs across space and time applies to stateless proxies as well. For a stateless proxy, the	

	branch parameter MUST be computed as a combinatory function of message parameters which are invariant on retransmission.	
5.	[Calculating Branch Parameter] The following procedure is RECOMMENDED .The proxy examines the branch ID in the topmost Via header field of the received request. If it begins with the magic cookie, the first component of the branch ID of the outgoing request is computed as a hash of the received branch ID. Otherwise, the first component of the branch ID is computed as a hash of the topmost Via, the tag in the To header field, the tag in the From header field, the Call-ID header field, the CSeq number (but not method), and the Request-URIfrom the received request.	
6.	[Other Transformations] All other message transformations specified in Section 16.6 MUST result in the same transformation of a retransmitted request. In particular, if the proxy inserts a Record-Route value or pushes URIs into the Route header field, it MUST place the same values in retransmissions of the request.	
7.	[Forwarding Request] A stateless proxy determines where to forward the request. The request is sent directly to the transport layer instead of through a client transaction.	
8.	[CANCEL Request] Stateless proxies MUST NOT perform special processing for CANCEL requests. They are processed by the above rules as any other requests. In particular, a stateless proxy applies the same Route header field processing to CANCEL requests that it applies to any other request.	
9.	[Response Processing] When a response arrives at a stateless proxy, the proxy MUST inspect the sent-by value in the first (topmost) Via header field value. If that address matches the proxy, (it equals a value this proxy has inserted into previous requests) the proxy MUST remove that header field value from the response and forward the result to the location indicated in the next Via header field value. If the address does not match the proxy, the message MUST be silently discarded.	

[Ref: RFC 3261 Section 16.11]

5.10Summary of Proxy Route Processing

Item	Case Description	Status

1.	In the absence of local policy to the contrary, the processing a proxy performs on a request containing a Route header field can be summarized in the following steps. ○ The proxy will inspect the Request-URI. If it indicates a resource owned by this proxy, the proxy will replace it with the results of running a location service. Otherwise, the proxy will not change the Request-URI. ○ The proxy will inspect the URI in the topmost Route header field value. If it indicates this proxy, the proxy removes it from the Route header field (this route node has been reached). ○ The proxy will forward the request to the resource indicated by the URI in the topmost Route header field value or in the Request-URI if no Route header field is present. The proxy determines the address, port and transport to use when forwarding the request by applying the procedures in [RFC 3263] to that URI. ○ If no strict-routing elements are encountered on the path of the request, the Request-URI will always indicate the target of the request.	

References:

- RFC 3261
- draft-ietf-sip-guidelines-05

6.1 Modeling SIP Extensions

Characteristics of SIP extensions:

- Extensions may define new methods. Extensions that define new methods do not need to use **Require** header, because the support of such extensions can be indicated by observing Allow header field in OPTIONS, INVITE (and its responses), 405 (Method Not Allowed) response, etc.
- Extensions may require proxy to insert header fields or header field parameters in the request while it traverses across network, or to require UAS to insert header fields or header field parameters in the response to the request. This capability is indicated by **Supported**, **Require**, or **Proxy-Require** header field.
- Extensions may define new body types. Message bodies MUST be understood by user agent in order to process a request. **Content-Disposition** is used to indicate that the message body is optional. User agents communicate what types of message bodies they support in the **Accept** header field, which is RECOMMENDED to be placed in requests that establish dialog.

For the purpose of design, we classify extensions into two types based of their interaction type:

- Extensions that work inside the context of a dialog (most of extensions),
- Extensions that work outside the context of a dialog (e.g. message signing, message compression).

Note that the same extension may specify that it supports both types of interaction.

In PJSIP, a SIP extension is characterized by:

- A unique name in PJSIP extension namespace. This unique name will be specified in Supported, Require, and/or Proxy-Require header field value if the extension specifies that it needs to be listed there.
- At which layer does it operate (core, or UA, or both)
- Dependencies to other extensions.
- What new methods it defines,
- For each individual new method, whether the method is a target refresh for a dialog.
- What new body types (MIME type) it defines,

- Whether it wants to be listed in Supported header fields.
- Whether the proxy needs to understand this extension (thus a token will be listed in Proxy-Require header if this extension is used).

6.2 Generic Design Requirements for SIP Extensions

Item	Case Description	Status
1.	Implement extension characterization as described above.	
2.	Core MUST put appropriate header fields to indicate the extensions it supports during transmission of appropriate requests and responses.	
3.	If a UAS does not understand an option-tag listed in a Require header field, it MUST respond by generating a response with status code 420 (Bad Extension). The UAS MUST add an Unsupported header field, and list in it those options it does not understand amongst those in the Require header field of the request.	
4.	Require and Proxy-Require MUST NOT be used in a SIP CANCEL request, or in an ACK request sent for a non-2xx response. These header fields MUST be ignored if they are present in these requests.	
5.	[Message Processing Content] UAS MUST check if it understands the message body type (**Content-Type**), the language (**Content-Language**), and encoding (**Content-Encoding**). If it doesn't understand any of them, and the handling of the body is not optional (as indicated by **Content-Disposition**), the UAS MUST reject the request with 415 (Unsupported Media Type), listing the appropriate header fields.	
6.	[Selecting extensions for dialog] User/application MUST be able to select which extensions it wants to use for both incoming and outgoing dialogs. This selection MAY be represented as a profile. For incoming requests, dialog determines what extensions are to be used.	
7.	[UAS forcing extension to be used] UAS MUST be able to indicate that processing of certain requests require some extensions to be used, or otherwise a 421 (Extension Required) response will be sent.	
8.	UAS MUST list all the extensions being applied to the request in the response message.	
9.	Extensions must be able to query that other extensions are being used (particularly for the same dialog).	

10.	Extensions MAY modify dialog's behavior: o It may reject incoming message o It may retransmit message (e.g. response in 100rel). o It may suspend transmission of certain messages (e.g. suspend 200 when there is outstanding PRACK in 100rel)	
11.	For extensions that work on dialogs, the design SHOULD allow extensions to receive all request and response messages for the dialog. Extensions that don't understand a particular message should ignore it.	
12.	User agent should also indicate the language that the user prefers to use by listing the languages in **Accept-Language** header.	
13.	Extension framework MUST allow extension to work on or modify the whole message (e.g. digest auth-int, message signing, compression, encryption, etc.).	

6.3 100rel Extension (RFC 3262)

Reference:

- RFC 3262

6.3.1 UAS Behavior

Item	Case Description	Status
1.	MUST NOT send 100 response reliably.	
2.	MUST NOT send any reliable provisional response if UAC doesn't specify **100rel** in **Supported** header field.	
3.	MUST add **Require** header field in the provisional response containing **100rel** option tag, and add **RSeq** header.	
4.	Retransmits the reliable provisional response, until a matching PRACK is received.	
5.	Keeps list of unacknowledged responses.	
6.	Answers PRACK with 481 if it doesn't match any outstanding provisional response.	
7.	MUST NOT send second reliable provisional response when the first one has not completed. It is RECOMMENDED that the UAS doesn't send reliable provisional response when there is outstanding one.	
8.	The UAS MAY send a final response to the initial request	

	before having received PRACKs for all unacknowledged reliable provisional responses, unless the final response is 2xx and any of the unacknowledged reliable provisional responses contained a session description. In that case, it MUST NOT send a final response until those provisional responses are acknowledged.	
9.	If the UAS does send a final response when reliable responses are still unacknowledged, it SHOULD NOT continue to retransmit the unacknowledged reliable provisional responses, but it MUST be prepared to process PRACK requests for those outstanding responses. A UAS MUST NOT send new reliable provisional responses (as opposed to retransmissions of unacknowledged ones) after sending a final response to a request.	

6.3.2 UAC Behavior

Item	Case Description	Status
1.	The extension framework controls whether the dialog should support or require reliable provisional response extension to be used.	
2.	The receipt of provisional MUST establish the dialog if one is not yet created.	
3.	MUST send PRACK request (within dialog) when reliable provisional response is received.	
4.	SHOULD NOT retransmit PRACK when the retransmission of reliable provisional response is received.	
5.	MUST drop reliable provisional response that is received out of order (i.e. out of sequence **RSeq** header value).	
6.	MAY acknowledge reliable provisional response that is received after the final response.	

6.3.3 Offer/Answer Model

Item	Case Description	Status
1.	The reliable provisional response extension CAN be used to exchange session's offer/answer description.	
2.	User agents MUST take care the offer/answer exchange so that there is no unacknowledged offer when the session is confirmed.	

6.4 Alternative Network Address Types Extension (sip-anat-usage, RFC 4092)

Abstract: *Alternative Network Address Types (ANAT) extension provides mechanism to offer callee with either Ipv4 or Ipv6 in the SDP. User agents specify the support of this extension by putting option tag **sdp-anat** in **Supported** or **Require** header field.*

This extension will not change the processing of SIP messages, thus no specific design requirement or consideration is required for this extension.

6.5 P-Asserted-Identity Private Extension (sip-asserted-identity, RFC 3325)

Abstract: *This RFC describes private extensions to the Session Initiation Protocol (SIP) that enable a network of trusted SIP servers to assert the identity of authenticated users, and the application of existing privacy mechanisms to the identity problem.*

This extension mostly changes the behavior of SIP proxies.

TODO: set requirements for processing by proxies.

6.6 Media Authorization Private Extension (sip-call-auth, RFC 3313)

Abstract: *This RFC describes the need for Quality of Service (QoS) and media authorization and defines a Session Initiation Protocol (SIP) extension that can be used to integrate QoS admission control with call signaling and help guard against denial of service attacks.*

The processing of this extension will not change the behavior of SIP protocol, thus does NOT need special requirements or considerations.

6.7 User Agent Capabilities Extension (sip-callee-caps, "pref", RFC 3840)

Abstract: *RFC 3840 defines mechanisms by which a Session Initiation Protocol (SIP) user agent can convey its capabilities and characteristics to other user agents and to the registrar for its domain. This information is conveyed as parameters of the Contact header field.*

The implementation of this extension will be used by application and will not change the behavior of the SIP stack.

6.8 Caller Preferences Extension (caller-prefs, RFC 3841)

Abstract: *This document describes a set of extensions to the Session Initiation Protocol (SIP) which allow a caller to express preferences about request handling in servers. This extension works for the caller side of sip-callee-caps/RFC 3840, and it will be treated similarly.*

This extension changes the way a request is processed by proxies.

TODO: requirements for proxies.

6.9 INFO Method Extension (RFC 2976)

Abstract: *This extension adds the INFO method to the SIP protocol. The intent of the INFO method is to allow for the carrying of session related control information that is generated during a session. One example of such session control information is ISUP and ISDN signaling messages used to control telephony call services.*

This does not even need an extension in PJSIP. Application can directly use any method without needed a special extension.

Chapter 7:Message Body Handling

Multipart/mixed

Message/sipfrag

Chapter 8:Message Integrity and Encryption

References:

- RFC 3261 (SIP), Section 23: S/MIME
- RFC 3893: Authenticated Identity Body (AIB) Format

Chapter 9:Message Compression

References:

- RFC 3320: Signaling Compression (SigComp)
- RFC 3486: Compressing the Session Initiation Protocol (SIP)