## 13.3.1.1 Progress

* If the UAS is not able to answer the invitation immediately=> VD: Ringing (quy trình khác).This is accomplished with a provisional response between 101 and 199. These provisional responses establish early dialogs and therefore follow the procedures of Section 12.1.1 in addition to those of Section 8.2.6. A UAS MAY send as many provisional responses as it likes. Each of these MUST indicate the same dialog ID. However, these will not be delivered reliably .
* If the UAS desires an extended period of time to answer the INVITE, it will need 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. To prevent cancellation, the UAS MUST send a non-100 provisional response at every minute, to handle the possibility of lost provisional responses .

=> Happen when user is placed on hold, or when interworking with PSTN systems

## 13.3.1.2 The INVITE is Redirected

* If the UAS decides to redirect the call, a 3xx response is sent. The response is passed to the INVITE server transaction, which will deal with its retransmissions .

13.3.1.3 The INVITE is Rejected

* A common scenario occurs when the callee is currently not willing or able to take additional calls at this end system. A 486 (Busy Here) SHOULD be returned in such a scenario. If the UAS knows that no other end system will be able to accept this call, a 600 (Busy Everywhere) response SHOULD be sent instead( not usually be used). The response is passed to the INVITE server transaction, which will deal with its retransmissions .
* A UAS rejecting an offer contained in an INVITE SHOULD return a 488 (Not Acceptable Here) response.

13.3.1.4 The INVITE is Accepted

* The UAS core generates a 2xx response. This response establishes a dialog, and therefore follows the procedures of Section 12.1.1 in addition to those of Section 8.2.6 .
* A 2xx response to an INVITE SHOULD contain the Allow header field and the Supported header field, and MAY contain the Accept header field .
* If the INVITE request contained an offer, and the UAS had not yet sent an answer, the 2xx MUST contain an answer. If the INVITE did not contain an offer, the 2xx MUST contain an offer if the UAS had not yet sent an offer .
* Once the response has been constructed, it is passed to the INVITE server transaction. Note, however, that the INVITE server transaction will be destroyed as soon as it receives this 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. The 2xx response is passed to the transport with an interval that starts at T1 seconds and doubles for each retransmission until it reaches T2 seconds (T1 and T2 are defined in Section 17). Response retransmissions cease when an ACK request for the response is received. This is independent of whatever transport protocols are used to send the response .
* Since 2xx is retransmitted end-to-end, there may be hops between UAS and UAC that are UDP. To ensure reliable delivery across these hops, the response is retransmitted periodically even if the transport at the UAS is reliable
* If the server retransmits the 2xx response for 64\*T1 seconds without receiving an ACK, the dialog is confirmed, but the session SHOULD be terminated. This is accomplished with a BYE, as described in Section 15 .

14 Modifying an Existing Session

* In Section 12 explains how to modify an existing dialog using a target refresh request. But, this section describes how to modify the actual session. This modification can involve changing addresses or ports, adding a media stream, deleting a media stream, and so on.

14.1 UAC Behavior

* If the session description format has the capability for version numbers, the offerer SHOULD indicate that the version of the session description has changed .
* The To, From, Call-ID, CSeq, and Request-URI of a re-INVITE are set following the same rules as for regular requests within an existing dialog, described in Section 12 .
* A UAC MAY choose not to add an Alert-Info header field or a body with Content-Disposition "alert" to re-INVITEs because UASs do not typically alert the user upon reception of a re-INVITE .
* Unlike an INVITE, which can fork, a re-INVITE will never fork, and therefore, only ever generate a single final response. The reason a re-INVITE will never fork is that the Request-URI identifies the target as the UA instance it established the dialog with, rather than identifying an address-of-record for the user .
* Note that a UAC MUST NOT initiate a new INVITE transaction within a dialog while another INVITE transaction is in progress in either direction .

1. If there is an ongoing INVITE client transaction, the TU MUST wait until the transaction reaches the completed or terminated state before initiating the new INVITE .
2. If there is an ongoing INVITE server transaction, the TU MUST wait until the transaction reaches the confirmed or terminated state before initiating the new INVITE .

* However, a 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 .

If a UA receives a non-2xx final response to a re-INVITE, the session parameters MUST remain unchanged, as if no re-INVITE had been issued .

* Note that, as stated in Section 12.2.1.2, if the non-2xx final response is a 481 (Call/Transaction Does Not Exist), or a 408 (Request Timeout), or no response at all is received for the re- INVITE, the UAC will terminate the dialog .

1. 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 .
2. 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 .

* When the timer fires, the UAC SHOULD attempt the re-INVITE once more, if it still desires for that session modification to take place.
* The rules for transmitting a re-INVITE and for generating an ACK for a 2xx response to re-INVITE are the same as for the initial INVITE (Section 13.2.1) .

14.2 UAS Behavior

* Section 13.3.1 describes the procedure for distinguishing incoming re-INVITEs from incoming initial INVITEs and handling a re-INVITE for an existing dialog .
* 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 .
* 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 .
* 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 .
* Versioning of the session description can be used to accommodate the capabilities of new arrivals to a conference, add or delete media, or change from a unicast to a multicast conference .
* If the new session description is not acceptable, the UAS can reject it by returning a 488 (Not Acceptable Here) response for the re- INVITE. This response SHOULD include a Warning header field .
* If a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a BYE to terminate the dialog .
* A UAS MAY choose not to generate 180 (Ringing) responses for a re- INVITE because UACs do not typically render this information to the user. For the same reason, UASs MAY choose not to use an Alert-Info header field or a body with Content-Disposition "alert" in responses to a re-INVITE .
* A UAS providing an offer in a 2xx (because the INVITE did not contain an offer) SHOULD construct the offer as if the UAS were making a brand new call, subject to the constraints of sending an offer that updates an existing session, as described in [13] in the case of SDP .

15 Terminating a Session

# 15.1 Terminating a Session with a BYE Request

15.1.1 UAC

* 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 .

## 15.1.2 UAS Behavior

* A UAS first processes the BYE request according to the general UAS processing described in Section 8.2. A UAS core receiving a BYE request checks if it matches an existing dialog. 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 .
* This rule means that a BYE sent without tags by a UAC will be rejected.
* A UAS core receiving a BYE request for an existing dialog MUST follow the procedures of Section 12.2.2 to process the request. Once done, the UAS SHOULD terminate the session (and therefore stop sending and listening for media). The only case where it can select not to are multicast sessions, where participation is possible even if the other participant in the dialog has terminated its involvement in the session. Whether or not it ends its participation on the session, the UAS core MUST generate a 2xx response to the BYE, and MUST pass that to the server transaction for transmission .
* The 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 .

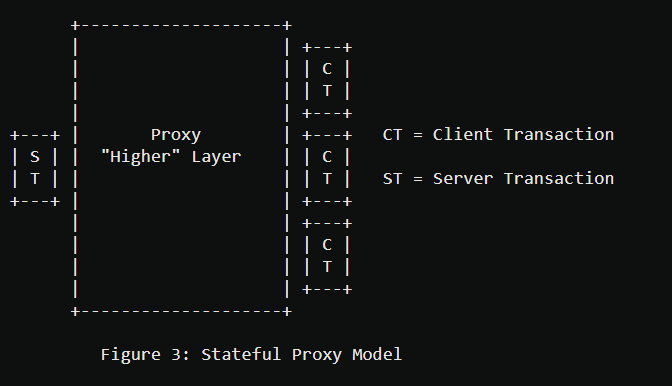
16 Proxy Behavior

16.1 Overview

* SIP proxies are elements that route SIP requests to user agent servers and SIP responses to user agent clients. A request may traverse several proxies on its way to a UAS. Each will make routing decisions, modifying the request before forwarding it to the next element. Responses will route through the same set of proxies traversed by the request in the reverse order .
* Being a proxy is a logical role for a SIP element. When a request arrives, an element that can play the role of a proxy first decides if it needs to respond to the request on its own.
* A proxy can operate in either a stateful or stateless mode for each new request.
  + When stateless, a proxy acts as a simple forwarding element. It forwards each request downstream to a single element determined by making a targeting and routing decision based on the request. It simply forwards every response it receives upstream. A stateless proxy discards information about a message once the message has been forwarded.
  + A stateful proxy remembers information (specifically, transaction state) about each incoming request and any requests it sends as a result of processing the incoming request. It uses this information to affect the processing of future messages associated with that request. A stateful proxy MAY choose to "fork" a request, routing it to multiple destinations. Any request that is forwarded to more than one location MUST be handled statefully .
* A stateful proxy MAY transition to stateless operation at any time during the processing of a request, so long as it did not do anything that would otherwise prevent it from being stateless initially (forking, for example, or generation of a 100 response). When performing such a transition, all state is simply discarded. The proxy SHOULD NOT initiate a CANCEL request .

16.2 Stateful Proxy

* A stateful proxy has a server transaction associated with one or more client transactions by a higher layer proxy processing component (see figure 3), known as a proxy core. An incoming request is processed by a server transaction. Requests from the server transaction are passed to a proxy core. The proxy core determines where to route the request, choosing one or more next-hop locations. An outgoing request for each next-hop location is processed by its own associated client transaction. The proxy core collects the responses from the client transactions and uses them to send responses to the server transaction .
* A stateful proxy creates a new server transaction for each new request received. Any retransmissions of the request will then be handled by that server transaction per Section 17. The proxy core MUST behave as a UAS with respect to sending an immediate provisional on that server transaction (such as 100 Trying) as described in Section 8.2.6. Thus, a stateful proxy SHOULD NOT generate 100 (Trying) responses to non-INVITE requests .
* This is a model of proxy behavior, not of software. An implementation is free to take any approach that replicates the external behavior this model defines .
* For all new requests, including any with unknown methods, an element intending to proxy the request MUST:
  + Validate the request (Section 16.3)
  + Preprocess routing information (Section 16.4)
  + Determine target(s) for the request (Section 16.5)
  + Forward the request to each target (Section 16.6)
  + Process all responses (Section 16.7)



16.3 Request Validation

Before an element can proxy a request, it MUST verify the message's validity. A valid message must pass the following checks:

1. Reasonable Syntax

2. URI scheme

3. Max-Forwards

4. (Optional) Loop Detection

5. Proxy-Require

6. Proxy-Authorization

=> If any of these checks fail, the element MUST behave as a user agent server (see Section 8.2) and respond with an error code .Notice that a proxy is not required to detect merged requests and MUST NOT treat merged requests as an error condition.

1. ***Reasonable syntax check***

* The request MUST be well-formed enough to be handled with a server transaction. Any components involved in the remainder of these Request Validation steps or the Request Forwarding section MUST be well-formed. Any other components, well-formed or not, SHOULD be ignored and remain unchanged when the message is forwarded
* This protocol is designed to be extended. Future extensions may define new methods and header fields at any time. An element MUST NOT refuse to proxy a request because it contains a method or header field it does not know about .

1. ***URI scheme check***

* 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 .

1. ***Max-Forwards check***

* 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 per Section 11. Otherwise, the element MUST return a 483 (Too many hops) response .

1. ***Optional Loop Detection check***

* To determine if the request has looped, the element MAY perform the branch parameter calculation described in Step 8 of Section 16.6 on this message and compare it to the parameter received in that Via header field. If the parameters match, the request has looped. If they differ, the request is spiraling, and processing continues. If a loop is detected, the element MAY return a 482 (Loop Detected) response .

1. ***Proxy-Require check***

* Future extensions to this protocol may introduce features that require special handling by proxies. Endpoints will include a Proxy-Require header field in requests that use these features, telling the proxy not to process the request unless the feature is understood .
* If the request contains a Proxy-Require header field (Section 20.29) 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 (Section 20.40) header field listing those option-tags the element did not understand .

1. ***Proxy-Authorization***

* Check If an element requires credentials before forwarding a request, the request MUST be inspected as described in Section 22.3. That section also defines what the element must do if the inspection fails .

16.4 Route Information Preprocessing

* 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 .
* This will only happen when the element sending the request to the proxy (which may have been an endpoint) is a strict router.

Instead, a proxy need only place enough information in those URIs to recognize them as values it provided when they later appear .

* 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 .
* If the first value in the Route header field indicates this proxy, the proxy MUST remove that value from the request .

## 16.5 Determining Request Targets

* Each target in the set is represented as a URI .
* 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 proceed to Section 16.6 .
* 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 (Section 16.6) .
* If the target set for the request has not been predetermined as described above, this implies that the element is responsible for the domain in the Request-URI, and the element MAY use whatever mechanism it desires to determine where to send the request. Any of these mechanisms can be modeled as accessing an abstract Location Service .
* When accessing the location service constructed by a registrar, the Request-URI MUST first be canonicalized as described in Section 10.3 before being used as an index. The output of these mechanisms is used to construct the target set .
* 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..
* Any information in or about the request or the current environment of the element MAY be used in the construction of the target set.
* As potential targets are located through these services, their URIs are added to the target set. Targets can only be placed in the target set once. If a target URI is already present in the set (based on the definition of equality for the URI type), it MUST NOT be added again .
* 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 .
* A proxy can only change the Request-URI of a request during forwarding if it is responsible for that URI. If the proxy is not responsible for that URI, it will not recurse on 3xx or 416 responses as described below .
* If the Request-URI of the original request indicates a resource this proxy is responsible for, the proxy MAY continue to add targets to the set after beginning Request Forwarding. It MAY use any information obtained during that processing to determine new targets .
* Allowing a URI to be added to the set only once reduces unnecessary network traffic, and in the case of incorporating contacts from redirect requests prevents infinite recursion .
* If the Request-URI indicates a resource at this proxy that does not exist, the proxy MUST return a 404 (Not Found) response .
* 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 .

16.6 Request Forwarding

* A stateful proxy MAY process the set in any order. It MAY process multiple targets serially, allowing each client transaction to complete before starting the next. It MAY start client transactions with every target in parallel. It also MAY arbitrarily divide the set into groups, processing the groups serially and processing the targets in each group in parallel .
* A common ordering mechanism is to use the qvalue parameter of targets obtained from Contact header fields (see Section 20.10). Targets are processed from highest qvalue to lowest. Targets with equal qvalues may be processed in parallel .
* 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
  + 4. Optionally add a Record-route header field value
  + 5. Optionally add additional header fields
  + 6. Postprocess 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
* Each of these steps is detailed below:
  + **1. Copy request** 
    - The proxy starts with a copy of the received 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 (See Section 7.3.1). The proxy MUST NOT add to, modify, or remove the message body .
    - An actual implementation need not perform a copy; the primary requirement is that the processing for each next hop begin with the same request .
  + **2. 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 .
    - This is the essence of a proxy's role. This is the mechanism through which a proxy routes a request toward its destination .
    - In some circumstances, the received Request-URI is placed into the target set without being modified. For that target, the replacement above is effectively a no-op .
  + **3. 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 .
    - Some existing UAs will not provide a Max-Forwards header field in a request .
  + **4. 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 .
    - Requests establishing a dialog may contain a preloaded Route header field .
    - 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.
    - The proxy will remain on the path if it chooses to not insert a Record-Route header field value into requests that are already part of a dialog.
* However, it would be removed from the path when an endpoint that has failed reconstitutes the dialog .

A proxy MAY insert a Record-Route header field value into any request. If the request does not initiate a dialog, the endpoints will ignore the value.

* Each proxy in the path of a request chooses whether to add a Record-Route header field value independently - the presence of a Record-Route header field in a request does not obligate this proxy to add a value .
* lr parameter need to be considered.
  + - The URI this proxy provides will be used by some other element to make a routing decision. This proxy, in general, has no way of knowing the capabilities of that element, so it must restrict itself to the mandatory elements of a SIP implementation: SIP URIs and either the TCP or UDP transports .
    - If the Request-URI contains a SIPS URI, or the topmost Route header field value 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..
    - A proxy at a security perimeter must remain on the perimeter throughout the dialog .
    - The proxy MAY include parameters in the Record-Route header field value. => 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 .
    - The URI a proxy places into a Record-Route header field is only valid for the lifetime of any dialog created by the transaction in which it occurs.
    - Record-routing may be required by certain services where the proxy needs to observe all messages in a dialog. However, it slows down processing and impairs scalability and thus proxies should only record-route if required for a particular service .
  + **5. Add Additional Header Fields** 
    - The proxy MAY add any other appropriate header fields to the copy at this point .
  + **6. Postprocess routing information**
    - A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, this can only be known with certainty if the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). 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 .
    - If the proxy has a local policy that mandates that the request visit one specific proxy, an alternative to pushing a Route value into the Route header field is to bypass the forwarding logic of item 10 below, and instead just send the request to the address, port, and transport for that specific proxy. If the request has a Route header field, this alternative MUST NOT be used unless it is known that next hop proxy is a loose router.
    - Furthermore, if the Request-URI contains a SIPS URI, TLS MUST be used to communicate with that proxy .
* 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 .
* **7. Determine Next-Hop Address, Port, and Transport**
* In the absence of such an overriding mechanism, the proxy applies the procedures listed in [4] as follows to determine where to send the request. If the proxy has reformatted the request to send to a strict-routing element as described in step 6 above, the proxy MUST apply those procedures to the Request-URI of the request. Otherwise, the proxy MUST apply the procedures to the first value in the Route header field, if present, else the Request-URI. The procedures will produce an ordered set of (address, port, transport) tuples .
* Independently of which URI is being used as input to the procedures of [4], if the Request-URI specifies a SIPS resource, the proxy MUST follow the procedures of [4] as if the input URI were a SIPS URI .
* For each tuple attempted, the proxy MUST format the message as appropriate for the tuple and send the request using a new client transaction as detailed in steps 8 through 10 .
* Since each attempt uses a new client transaction, it represents a new branch. Thus, the branch parameter provided with the Via header field inserted in step 8 MUST be different for each attempt .
* If the client transaction reports failure to send the request or a timeout from its state machine, the proxy continues to the next address in that ordered set. If the ordered set is exhausted, the request cannot be forwarded to this element in the target set. The proxy does not need to place anything in the response context, but otherwise acts as if this element of the target set returned a 408 (Request Timeout) final response .
* **8. Add a Via header field value**
* The proxy MUST insert a Via header field value into the copy before the existing Via header field values. The construction of this value follows the same guidelines of Section 8.1.1.7 .This implies that the proxy will compute its own branch parameter, which will be globally unique for that branch, and contain the requisite magic cookie. Note that this implies that the branch parameter will be different for different instances of a spiraled or looped request through a proxy .
  + - Proxies choosing to detect loops have an additional constraint in the value they use for construction of the branch parameter .A proxy choosing to detect loops SHOULD create a branch parameter separable into two parts by the implementation. The first part MUST satisfy the constraints of Section 8.1.1.7 as described above. The second is used to perform loop detection and distinguish loops from spirals .
    - The request method MUST NOT be included in the calculation of the branch parameter. In particular, CANCEL and ACK requests (for non-2xx responses) MUST have the same branch value as the corresponding request they cancel or acknowledge. The branch parameter is used in correlating those requests at the server handling them (see Sections 17.2.3 and 9.2) .
* **9. 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 (see Section 20.14) .
* **10. Forward Request**
  + - A stateful proxy MUST create a new client transaction for this request as described in Section 17.1 and instructs the transaction to send the request using the address, port and transport determined in step 7 .
* **11. Set timer**
  + - 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. Section 16.7 bullet 2 discusses how this timer is updated with provisional responses, and Section 16.8 discusses processing when it fires .

16.7 Response Processing

When a response is received by an element, it first tries to locate a client transaction (Section 17.1.3) matching the response. If none is found, the element MUST process the response (even if it is an informational response) as a stateless proxy (described below). If a match is found, the response is handed to the client transaction .

Forwarding responses for which a client transaction (or more generally any knowledge of having sent an associated request) is not found improves robustness. In particular, it ensures that "late" 2xx responses to INVITE requests are forwarded properly .

As client transactions pass responses to the proxy layer, the following processing MUST take place:

1. Find the appropriate response context
2. Update timer C for provisional responses
3. Remove the topmost Via
4. Add the response to the response context
5. Check to see if this response should be forwarded immediately
6. 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 .
7. Aggregate authorization header field values if necessary 8. Optionally rewrite Record-Route header field values 9. Forward the response 10. Generate any necessary CANCEL requests Session Initiation Protocol June 2002

* **Each of the above steps are detailed below:**

1. Find Context

* The proxy locates the "response context" it created before forwarding the original request using the key described in Section 16.6. The remaining processing steps take place in this context .

1. Update timer C for provisional responses

* For an INVITE transaction, if the response is a provisional response with status codes 101 to 199 inclusive (i.e., anything but 100), the proxy MUST reset timer C for that client transaction. The timer MAY be reset to a different value, but this value MUST be greater than 3 minutes .

1. Via

* The proxy removes the topmost Via header field value from the response .If no Via header field values remain in the response, the response was meant for this element and MUST NOT be forwarded .The remainder of the processing described in this section is not performed on this message, the UAC processing rules described in Section 8.1.3.

1. Add response to context

* Final responses received are stored in the response context until a final response is generated on the server transaction associated with this context. The response may be a candidate for the best final response to be returned on that server transaction. Information from this response may be needed in forming the best response, even if this response is not chosen .
* 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