=> Tiếp nối phần “Response Processing” của phần 16. Proxy Behavior

## 5. Check response for forwarding

* Until a final response has been sent on the server transaction, the following responses MUST be forwarded immediately:
  + Any provisional response other than 100 (Trying)
  + Any 2xx response 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](https://www.rfc-editor.org/rfc/rfc3261#section-10), and it MUST NOT create any new branches in this context.
* 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.
* After a final response has been sent on the server transaction, the following responses MUST be forwarded immediatel => Any 2xx response to an INVITE request
* 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".
* Any response chosen for immediate forwarding MUST be processed as described in steps "Aggregate Authorization Header Field Values" through "Record-Route".

=> 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.

## 6. Choosing the best response

* A stateful proxy MUST send a final response to a response context's server transaction if no final responses have been immediately forwarded by the above rules and all client transactions in this response context have been terminated.
* The stateful proxy MUST choose the "best" final response among those received and stored in the response context.
* If there are no final responses in the context, the proxy MUST send a 408 (Request Timeout) response to the server transaction.
* Otherwise, 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.
* 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".
* For example, if a proxy forwarded a request to 4 locations, and received 503, 407, 501, and 404 responses, it may choose to forward the 407 (Proxy Authentication Required) response.
* 1xx and 2xx responses may be involved in the establishment of dialogs. When a request does not contain a To tag, the To tag in the response is used by the UAC to distinguish multiple responses to a dialog creating request. A proxy MUST NOT insert a tag into the To header field of a 1xx or 2xx response if the request did not contain one. A proxy MUST NOT modify the tag in the To header field of a 1xx or 2xx response.
* Since a proxy may not insert a tag into the To header field of a 1xx response to a request that did not contain one, it cannot issue non-100 provisional responses on its own. However, it can branch the request to a UAS sharing the same element as the proxy. This UAS can return its own provisional responses, entering into an early dialog with the initiator of the request. The UAS does not have to be a discreet process from the proxy. It could be a virtual UAS implemented in the same code space as the proxy.
* 3-6xx responses are delivered hop-by-hop. When issuing a 3-6xx response, the element is effectively acting as a UAS, issuing its own response, usually based on the responses received from downstream elements. An element SHOULD preserve the To tag when simply forwarding a 3-6xx response to a request that did not contain a To tag.
  + - preserving the original tag may assist with debugging.
* When the proxy is aggregating information from several responses, choosing a To tag from among them is arbitrary, and generating a new To tag may make debugging easier. This happens, for instance, when combining 401 (Unauthorized) and 407 (Proxy Authentication Required) challenges, or combining Contact values from unencrypted and unauthenticated 3xx responses.

## 7. Aggregate Authorization Header Field Values

* If the selected response is a 401 (Unauthorized) or 407 (Proxy Authentication Required), the proxy MUST collect any WWW- Authenticate and Proxy-Authenticate header field values from all other 401 (Unauthorized) and 407 (Proxy Authentication Required) responses received so far in this response context and add them to this response without modification before forwarding.
* The resulting 401 (Unauthorized) or 407 (Proxy Authentication Required) response could have several WWW- Authenticate AND Proxy-Authenticate header field values.

## 8. Record-Route

* If the selected response contains a Record-Route header field value originally provided by this proxy, the proxy MAY choose to rewrite the value before forwarding the response. This allows the proxy to provide different URIs for itself to the next upstream and downstream elements.

(For instance, it is useful for multi-homed hosts)

* If the proxy received the request over TLS, and sent it out over a non-TLS connection, the proxy MUST rewrite the URI in the Record-Route header field to be a SIPS URI.
* If the proxy received the request over a non-TLS connection, and sent it out over TLS, the proxy MUST rewrite the URI in the Record-Route header field to be a SIP URI
* The new URI provided by the proxy MUST satisfy the same constraints on URIs placed in Record-Route header fields in requests (see Step 4 of [Section 16.6](https://www.rfc-editor.org/rfc/rfc3261#section-16.6)) with the following modifications: The URI SHOULD NOT contain the transport parameter unless the proxy has knowledge that the next upstream (as opposed to downstream) element that will be in the path of subsequent requests supports that transport.
* When a proxy does decide to modify the Record-Route header field in the response, one of the operations it performs is locating the Record-Route value that it had inserted. If the request spiraled, and the proxy inserted a Record-Route value in each iteration of the spiral, locating the correct value in the response (which must be the proper iteration in the reverse direction) is tricky.
* When the response arrives, the proxy modifies the first Record-Route whose identifier matches the proxy instance. The modification results in a URI without this piece of data appended to the user portion of the URI.
* Not every response to a request to which a proxy adds a Record-Route header field value will contain a Record-Route header field. If the response does contain a Record-Route header field, it will contain the value the proxy added.

## 9. Forward response

* After performing the processing described in steps "Aggregate Authorization Header Field Values" through "Record-Route", the proxy MAY perform any feature specific manipulations on the selected response.
  + The proxy MUST NOT add to, modify, or remove the message body. Unless otherwise specified
  + The proxy MUST NOT remove any header field values other than the Via header field value discussed in [Section 16.7](https://www.rfc-editor.org/rfc/rfc3261#section-16.7) Item 3. The proxy MUST pass the response to the server transaction associated with the response context. This will result in the response being sent to the location now indicated in the topmost Via header field value.
  + If the server transaction is no longer available to handle the transmission, the element MUST forward the response statelessly by sending it to the server transport. The server transaction might indicate failure to send the response or signal a timeout in its state machine.

=> The proxy MUST maintain the response context until all of its associated transactions have been terminated, even after forwarding a final response.

## 10. Generate CANCELs

* If the forwarded response was a final response, the proxy MUST generate a CANCEL request for all pending client transactions associated with this response context. A proxy SHOULD also generate a CANCEL request for all pending client transactions associated with this response context when it receives a 6xx response. A pending client transaction is one that has received a provisional response, but no final response (it is in the proceeding state) and has not had an associated CANCEL generated for it. Generating CANCEL requests is described in [Section 9.1](https://www.rfc-editor.org/rfc/rfc3261#section-9.1).
* The requirement to CANCEL pending client transactions upon forwarding a final response does not guarantee that an endpoint will not receive multiple 200 (OK) responses to an INVITE. 200 (OK) responses on more than one branch may be generated before the CANCEL requests can be sent and processed. Further, it is reasonable to expect that a future extension may override this requirement to issue CANCEL requests.

[16.8](https://www.rfc-editor.org/rfc/rfc3261#section-16.8) Processing Timer C

* 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

[16.9](https://www.rfc-editor.org/rfc/rfc3261#section-16.9) Handling Transport Errors

* If the transport layer notifies a proxy of an error when it tries to forward a request (see [Section 18.4](https://www.rfc-editor.org/rfc/rfc3261#section-18.4)), 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.
* If a proxy cancels its outstanding client transactions, a single malicious or misbehaving client can cause all transactions to fail through its Via header field.

[16.10](https://www.rfc-editor.org/rfc/rfc3261#section-16.10) CANCEL Processing

* A stateful proxy MAY generate a CANCEL to any other request it has generated at any time (subject to receiving a provisional response to that request as described in [section 9.1](https://www.rfc-editor.org/rfc/rfc3261#section-9.1)). A proxy MUST cancel any pending client transactions associated with a response context when it receives a matching CANCEL request.
* 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.
* While a CANCEL request is handled in a stateful proxy by its own server transaction, a new response context is not created for it.
* Instead, the proxy layer searches its existing response contexts for the server transaction handling the request associated with this CANCEL.
  + If a matching response context is found, the element MUST immediately return a 200 (OK) response to the CANCEL request. In this case, the element is acting as a user agent server as defined in [Section 8.2](https://www.rfc-editor.org/rfc/rfc3261#section-8.2). Furthermore, the element MUST generate CANCEL requests for all pending client transactions in the context as described in [Section 16.7](https://www.rfc-editor.org/rfc/rfc3261#section-16.7) step 10.
  + If a response context is not found, 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).

[16.11](https://www.rfc-editor.org/rfc/rfc3261#section-16.11) Stateless Proxy

* A proxy is a simple message forwarder. Much of the processing performed when acting statelessly is the same as when behaving statefully. The differences are detailed here.
  + A stateless proxy does not have any notion of a transaction, or of the response context used to describe stateful proxy behavior.
  + Stateless proxies do not retransmit messages on their own.
  + They do, however, forward all retransmissions they receive (they do not have the ability to distinguish a retransmission from the original message). Furthermore, when handling a request statelessly, an element MUST NOT generate its own 100 (Trying) or any other provisional response.
  + A stateless proxy MUST validate a request as described in [Section](https://www.rfc-editor.org/rfc/rfc3261#section-16.3) [16.3](https://www.rfc-editor.org/rfc/rfc3261#section-16.3) A stateless proxy MUST follow the request processing steps described in Sections [16.4](https://www.rfc-editor.org/rfc/rfc3261#section-16.4) through [16.5](https://www.rfc-editor.org/rfc/rfc3261#section-16.5) with the following exception:
  + 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.

* A stateless proxy MUST follow the request processing steps described in [Section 16.6](https://www.rfc-editor.org/rfc/rfc3261#section-16.6) with the following exceptions:
  + The requirement for unique branch IDs across space and time applies to stateless proxies as well.
    - However, a stateless proxy cannot simply use a random number generator to compute the first component of the branch ID, as described in [Section](https://www.rfc-editor.org/rfc/rfc3261#section-16.6) [16.6](https://www.rfc-editor.org/rfc/rfc3261#section-16.6) bullet 8. This is because retransmissions of a request need to have the same value, and a stateless proxy cannot tell a retransmission from the original request. The stateless proxy MAY use any technique it likes to guarantee uniqueness of its branch IDs across transactions.
    - 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-URI from the received request. One of these fields will always vary across two different transactions.
  + All other message transformations specified in [Section 16.6](https://www.rfc-editor.org/rfc/rfc3261#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. As for the Via branch parameter, this implies that the transformations MUST be based on time-invariant configuration or retransmission-invariant properties of the request.

* + A stateless proxy determines where to forward the request as described for stateful proxies in [Section 16.6](https://www.rfc-editor.org/rfc/rfc3261#section-16.6) Item 10. The request is sent directly to the transport layer instead of through a client transaction.
* Since a stateless proxy must forward retransmitted requests to the same destination and add identical branch parameters to each of them, it can only use information from the message itself and time-invariant configuration data for those calculations.
* 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. Response processing as described in [Section 16.7](https://www.rfc-editor.org/rfc/rfc3261#section-16.7) does not apply to a proxy behaving statelessly.
* 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. The proxy MUST NOT add to, modify, or remove the message body.
  + Unless specified otherwise, the proxy MUST NOT remove any other header field values. If the address does not match the proxy, the message MUST be silently discarded.

[16.12](https://www.rfc-editor.org/rfc/rfc3261#section-16.12) Summary of Proxy Route Processing

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.

1. 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.
2. 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).
3. 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 [[4](https://www.rfc-editor.org/rfc/rfc3261#ref-4)] 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.

[16.12.1](https://www.rfc-editor.org/rfc/rfc3261#section-16.12.1) Examples

[16.12.1.1](https://www.rfc-editor.org/rfc/rfc3261#section-16.12.1.1) Basic SIP Trapezoid

This scenario is the basic SIP trapezoid, U1 -> P1 -> P2 -> U2, with both proxies record-routing. Here is the flow. U1 sends:

INVITE sip:callee@domain.com SIP/2.0

Contact: <sip:caller@u1.example.com>

to P1. P1 is an outbound proxy. P1 is not responsible for domain.com, so it looks it up in DNS and sends it there. It also adds a Record-Route header field value:

INVITE sip:callee@domain.com SIP/2.0

Contact: <sip:caller@u1.example.com>

Record-Route: <sip:p1.example.com;lr>

P2 gets this. It is responsible for domain.com so it runs a location service and rewrites the Request-URI. It also adds a Record-Route header field value. There is no Route header field, so it resolves the new Request-URI to determine where to send the request:

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: <sip:caller@u1.example.com>

Record-Route: <sip:p2.domain.com;lr>

Record-Route: <sip:p1.example.com;lr>

The callee at u2.domain.com gets this and responds with a

200 OK: SIP/2.0 200 OK

Contact: <sip:callee@u2.domain.com>

Record-Route: <sip:p2.domain.com;lr>

Record-Route: <sip:p1.example.com;lr>

The callee at u2 also sets its dialog state's remote target URI to sip:caller@u1.example.com and its route set to:

(<sip:p2.domain.com;lr>,<sip:p1.example.com;lr>)

This is forwarded by P2 to P1 to U1 as normal. Now, U1 sets its dialog state's remote target URI to sip:callee@u2.domain.com and its route set to:

(<sip:p1.example.com;lr>,<sip:p2.domain.com;lr>)

Since all the route set elements contain the lr parameter, U1 constructs the following BYE request:

BYE sip:callee@u2.domain.com SIP/2.0

Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr

As any other element (including proxies) would do, it resolves the URI in the topmost Route header field value using DNS to determine where to send the request. This goes to P1. P1 notices that it is not responsible for the resource indicated in the Request-URI so it doesn't change it. It does see that it is the first value in the Route header field, so it removes that value, and forwards the request to P2:

BYE sip:callee@u2.domain.com SIP/2.0

Route: <sip:p2.domain.com;lr>

P2 also notices it is not responsible for the resource indicated by the Request-URI (it is responsible for domain.com, not u2.domain.com), so it doesn't change it. It does see itself in the first Route header field value, so it removes it and forwards the following to u2.domain.com based on a DNS lookup against the Request-URI:

BYE sip:callee@u2.domain.com SIP/2.0

### [16.12.1.2](https://www.rfc-editor.org/rfc/rfc3261#section-16.12.1.2) Traversing a Strict-Routing Proxy

* In this scenario, a dialog is established across four proxies, each of which adds Record-Route header field values. The third proxy implements the strict-routing procedures specified in [RFC 2543](https://www.rfc-editor.org/rfc/rfc2543) and many works in progress.

U1->P1->P2->P3->P4->U2 The INVITE arriving at U2 contains:

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: <sip:caller@u1.example.com>

Record-Route: <sip:p4.domain.com;lr>

Record-Route: <sip:p3.middle.com>

Record-Route: <sip:p2.example.com;lr>

Record-Route: <sip:p1.example.com;lr>

* Which U2 responds to with a 200 OK. Later, U2 sends the following BYE request to P4 based on the first Route header field value.

BYE sip:caller@u1.example.com SIP/2.0

Route: <sip:p4.domain.com;lr>

Route: <sip:p3.middle.com>

Route: <sip:p2.example.com;lr>

Route: <sip:p1.example.com;lr>

* P4 is not responsible for the resource indicated in the Request-URI so it will leave it alone. It notices that it is the element in the first Route header field value so it removes it. It then prepares to send the request based on the now first Route header field value of sip:p3.middle.com, but it notices that this URI does not contain the lr parameter, so before sending, it reformats the request to be:

BYE sip:p3.middle.com

SIP/2.0 Route: <sip:p2.example.com;lr>

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

* P3 is a strict router, so it forwards the following to P2:

BYE sip:p2.example.com;lr SIP/2.0

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

* P2 sees the request-URI is a value it placed into a Record-Route header field, so before further processing, it rewrites the request to be:

BYE sip:caller@u1.example.com SIP/2.0

Route: <sip:p1.example.com;lr>

* P2 is not responsible for u1.example.com, so it sends the request to P1 based on the resolution of the Route header field value.
* P1 notices itself in the topmost Route header field value, so it removes it, resulting in:

BYE sip:caller@u1.example.com SIP/2.0

* + - Since P1 is not responsible for u1.example.com and there is no Route header field, P1 will forward the request to u1.example.com based on the Request-URI.

[16.12.1.3](https://www.rfc-editor.org/rfc/rfc3261#section-16.12.1.3) Rewriting Record-Route Header Field Values

* In this scenario, U1 and U2 are in different private namespaces and they enter a dialog through a proxy P1, which acts as a gateway between the namespaces.
* U1->P1->U2 U1 sends:

INVITE sip:callee@gateway.leftprivatespace.com

SIP/2.0 Contact: <sip:caller@u1.leftprivatespace.com>

* P1 uses its location service and sends the following to U2:

INVITE sip:callee@rightprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

* U2 sends this 200 (OK) back to P1:

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

* P1 rewrites its Record-Route header parameter to provide a value that U1 will find useful, and sends the following to U1:

SIP/2.0 200 OK Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.leftprivatespace.com;lr>

* Later, U1 sends the following BYE request to P1:

BYE sip:callee@u2.rightprivatespace.com SIP/2.0

Route: <sip:gateway.leftprivatespace.com;lr>

which P1 forwards to U2 as:

BYE sip:callee@u2.rightprivatespace.com SIP/2.0