Advanced Technologies Research Center

Carrier Systems Business Unit, 3Com

SIP Open Issues

AT 033198

Abstract

The soon to be fully standardized Session Initiation Protocol is still very much a work in progress. As always, the implementation projects will shed new light on various aspects of the protocol. In this short paper we point out and try to fix some ambiguities still present in the draft.

Revised   :   March 31, 1999
Authors   :   Jacek Grabiec
                 Jerry Mahler
                 Rick Dean

3Com
1800 West Central Rd
Mount Prospect, Illinois 60056

Introduction

The soon to be fully standardized Session Initiation Protocol is still very much a work in progress. As always, the implementation projects will shed new light on various aspects of the protocol. In this short paper we point out and try to fix some ambiguities still present in the draft.

Clarifications

## *Simultaneous ending of the call on both sides of the connection.*

6.17 `"Consecutive requests that differ in request method, headers or body, but have the same Call-ID MUST contain strictly monotonically increasing and contiguous sequence numbers"`

Furthermore, a given host should respond with a 400-error message if it receives a BYE with a Cseq that is not higher. It is unclear whether the term "higher" refers to the Cseq in the BYE request versus Cseq in the INVITE request or to last recorded Cseq during a given call.

The problem may occur if both ends hang-up simultaneously. If the Cseq in last request (e.g. INVITE) was 1234 then both ends would issue a BYE request with a 1235 Cseq and both would expect an OK with a Cseq of 1235 or a request with an incremented Cseq of 1236. Therefore, the reception of a BYE with a Cseq of 1235 should cause a 400 response to be generated from both sides. Although the connection should be terminated finally after several retransmissions, it is not a "graceful" termination**.**

The proposal: a server should respond with a 400 message to any request with Cseq<last Cseq but treat any BYE with Cseq=lastCSeq as correct request and respond with 200 OK. Furthermore a host should ignore any new request (with the same Call-ID) while waiting for OK for already issued BYE.

Cancel received after final 200 OK has been issued
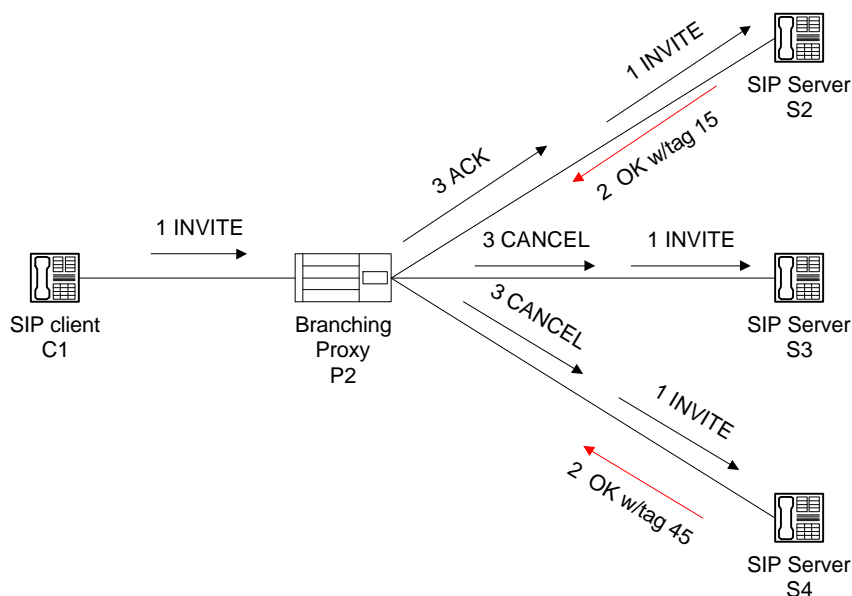
Figure 1: Cancel received after final response

4.2.5 `The CANCEL request cancels a pending request with the same Call-ID, To, From and CSeq (sequence number only) header field values, but does`


```
not affect a completed request. (A request is considered completed if
the server has returned a final status response.)
```

The problem: according to the RFC, CANCEL received after the final response has been sent should not affect the call. Such a CANCEL could be silently discarded. In the above scenario a proxy P2 is configured to accept the first response and cancel all outstanding transactions. If so, it will generate a CANCEL to S3 and S4 as soon as it receives 200 OK from S2. But the OK from e.g. S3 is already on its way to P2. On one hand, the P2 proxy MUST forward all final responses to the client, but on the other hand it is supposed to 'connect' to only one end host and cancel all other connections. The proxy may (?) choose to lose all state information about the pending INVITE as soon as it sent CANCEL, therefore the received 200 OK would not match any outstanding transaction.

The proposal: the CANCEL should have the preference and the behavior of the hosts should be as follows:

- The proxy SHOULD NOT forward 200 OK to the client after the CANCEL for that transaction has been issued (the response should be silently discarded).

- The server SHOULD accept CANCEL, respond with 200 OK to the CANCEL request and 'hang up' on the current call (behave the way it would had it never issued final response). If other INVITE requests for the same Call-Leg are in process, the call is not 'hung up' until (and if) the last INVITE is cancelled (see also the redefinition of isomorphic requests in the section "Different Via list in isomorphic requests").

- The client SHOULD NOT process a 200 OK final response for a transaction it has already issued a CANCEL request for and should silently discard the response to the INVITE.

INVITE and CANCEL reordering
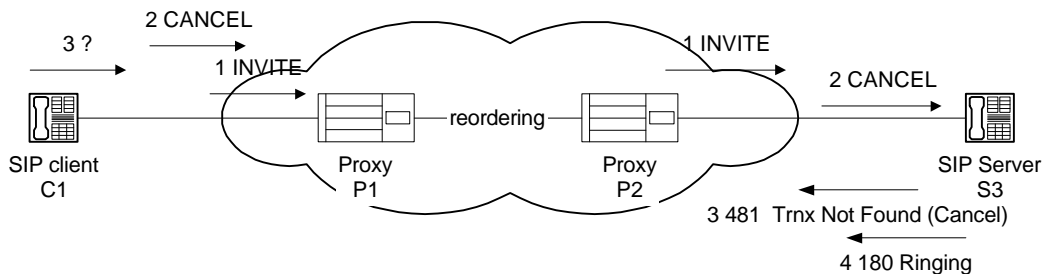
Figure 2: INVITE and CANCEL request reordering.

Both ends of the call may wish to renegotiate new SDP parameters at any time. In such a case a host issues a new INVITE with a higher Cseq. Furthermore, such a host may also wish to cancel a re-INVITE even before it receives any response from the other end.

The problems: There are some potential problems when packet reordering occurs. In the above scenario, the CANCEL reaches the server before the re-INVITE, therefore no matching transaction is found and a 481 error response is returned. The later arriving INVITE creates a new transaction at the server, generating a 1xx or final response. The client may, in turn, return an ACK. The result is that both ends renegotiated the call parameters, but that was not the intention. Alternatively, the client may not process the OK response, resulting in the server resending the OK response to the re-INVITE multiple times before giving up. The result is an unclean end to a re-INVITE.

The proposal: the client SHOULD NOT generate a CANCEL for re-INVITE's before any response from the server is heard (however BYE is still permitted). Then follow with CANCEL if 1xx response is received or yet another INVITE with original call's parameters if final response was received.

Problems Stemming from Assigning Call-Legs based on Call-ID, To, From and Associated Tags Only
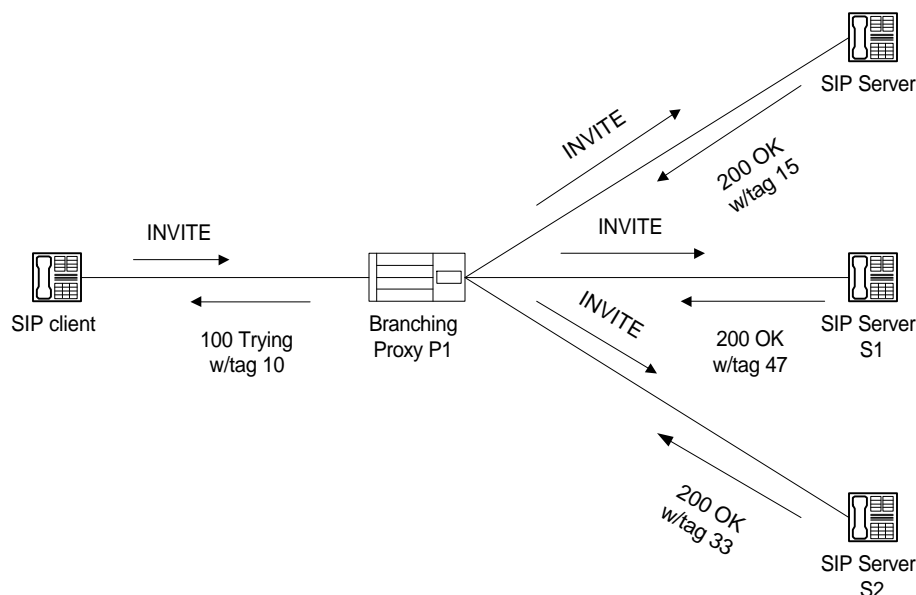
Unnecessary Call-Leg creation

Figure 3: 'Bogus' Call-Leg creation.

12.3.3 `A stateless proxy MUST NOT generate its own provisional responses.`

12.3.4 `A stateful proxy server MAY generate its own provisional (1xx) responses.`

12.4 `The server MUST respond to the request immediately with a 100 (Trying) response.`

Furthermore, a 'To' Tag MAY be added to a provisional response generated by a stateful proxy (6.37).

The problem: This 'freedom' in behavior permitted within the specification is likely to cause problems.

1. The forking proxy is configured to wait for the first 200 OK response from any of the contacted servers and cancel all outstanding requests upon receiving such a response. Alternatively, the proxy is configured to try only for a finite amount of time to contact the desired party. In either case, it adds a 'To' tag to its own provisional response sent upstream.

In that case, the branching proxy P1 generates a 100 Trying message back to the end client that creates a new Call-Leg with To tag of 10 at the end client. Similarly, every server is required to fill the 'To' tag in the generated response. Since there is no way for the client to distinguish between the response from the proxy and the server, four Call-Legs will be created, consisting of To tags of 10, 15, 47 and 33. Moreover, the client will expect the final response for all four Call-Legs but it will never get one from the proxy. The outstanding Call-Leg may exist at the client indefinitely. There's no clear way to cancel this Call-Leg, since it is really only between client and the proxy.

2. The forking proxy is configured to try indefinitely for a final response from all of the contacted servers. At the same time it does not add a 'To' tag to its own provisional response sent upstream. The proxy is not required to retransmit the request while waiting for the final response if it already received a provisional response from the server. Moreover, the server (and/or another proxy) is allowed to try to contact the user indefinitely and is not required to retransmit provisional response.

In that case, if only two of three contacted servers, say S1 and S2, answer with the final response, two call legs will be created at the client. There's no way for the client to stop the proxy from trying to contact the third server: the call leg with a To tag of 47 and another call leg with a To tag of 33.

The solution: the proxy configured according to (1) MUST NOT add a 'To' tag, while a proxy configured according to (2) MUST add a 'To' tag. In addition, if (2), it should generate a 300-like class response "done proxying" when it receives final responses from all tried servers to bring down an outstanding Call-Leg at the client. Also, an EXPIRE header field in the INVITE request should be mandatory. Not only would it limit the time the potential user is notified but would also solve potential problems with stateful proxies indefinitely waiting for the final responses

Cancel Terminating All Pending INVITE Requests.

1.3 A call leg is identified by the combination of Call-ID, To and From (and associated Tags). The end host is not required to check the Via fields.

The problem: In the situations like the one illustrated below it will cause improper behavior. The original INVITE is branched and then merged in such a way that SIP Server 1 receives two copies of the INVITE request that differ only in VIA fields. Therefore only one Call-Leg is created. Say the request from P1 was received first and S1 responds with 180 Ringing message. Subsequent INVITE from P2 is treated as retransmission and the previous response is retransmitted, with the 'Via' list adjusted to be identical to the second INVITE request. If SIP Server 2 responds with 200 OK (while S1 is still ringing), then Proxy P2 may immediately send a CANCEL to S1. That will terminate the pending request and S1 will stop ringing. However P1 is under the assumption that SIP Server S1 is still ringing. Furthermore, it does not have to check the status of S1 by resending the INVITE request periodically, because it has already received a provisional response. If P1 is configured to periodically check the status of the server by resending the INVITE request, one of two things may happen. SIP Server S1 may begin ringing again, resulting in a annoying on-off-on-off modulation of the ringing at that phone. Alternatively, SIP Server S1 may receive the subsequent INVITE from P1 and assume it is a very delayed request which has been cancelled by the CANCEL sent by P2, resulting in the phone not ringing, and perhaps an error response returned to P1.
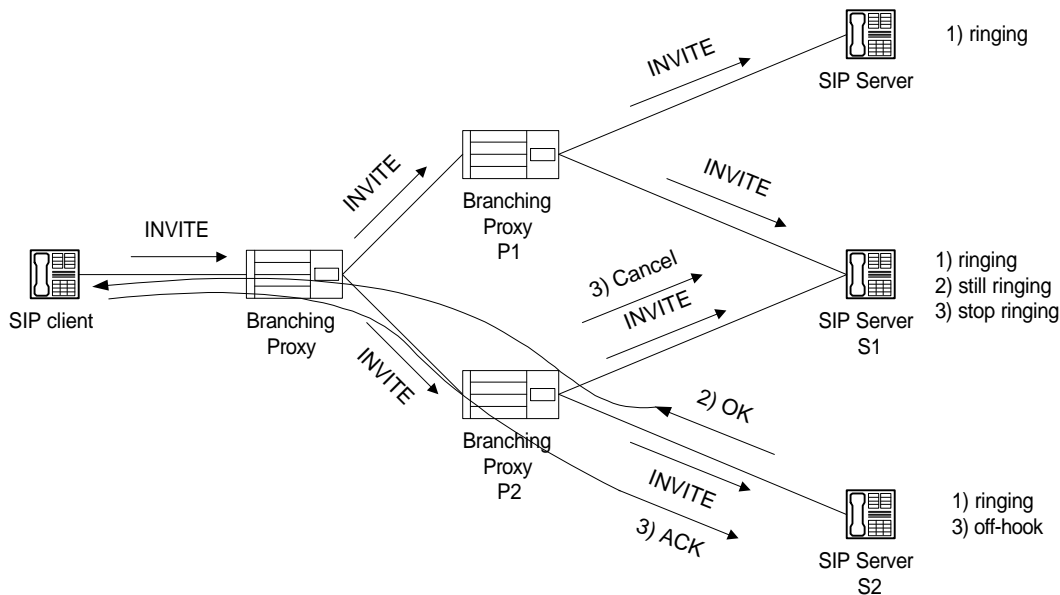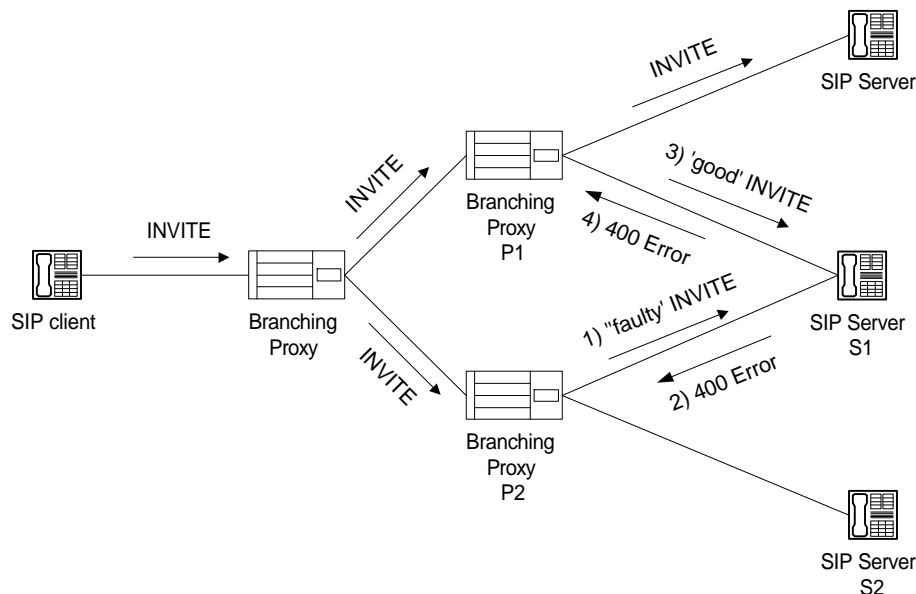

Figure 4: One Cancel terminating all pending request.

Isomorphic Requests

`1.3 Isomorphic request or response: Two requests or responses are defined to be isomorphic for the purposes of this document if they have the same values for the Call-ID, To, From and CSeq header fields. In addition, requests have to have the same Request-URI.`

## The Problems

'Good' / 'Bad' Invites

Figure 5: Good / Bad requests processing.

Two INVITE requests coming to S1 from P1 and P2 are isomorphic according to the specification, therefore one is treated as retransmission of the other. If the first one (1 'faulty' INVITE) is corrupted by the faulty proxy P2 (e.g. bad syntax) it will result in a 400 Error response. Subsequent and perfectly correct requests from P1 will trigger the same response sent to P1 as 'Via' fields are copied, since the S1 is supposed to retransmit last message in response to isomorphic request.

Different Via list in isomorphic requests.

If the request is determined to be 'isomorphic' the end host should treat it as the retransmission of the original request, it should retransmit (once?) the last response and discard the message. On the other hand the response 'The response MUST copy the To, From, Call-ID, CSeq and Via fields from the request.' Since two requests may be isomorphic according to the definition and still have different 'Via' fields the server cannot fullfill both requirements. If the 'Via' fields do not have to be the same for the requests to be isomorphic several problems may occur

It is rather clear that copying the 'Via' fields to the responses should be observed rather than retransmitting the last response. However it implies that the server has to create and keep states for ALL transactions, including 'semi-isomorphic', in case a retransmission is needed.
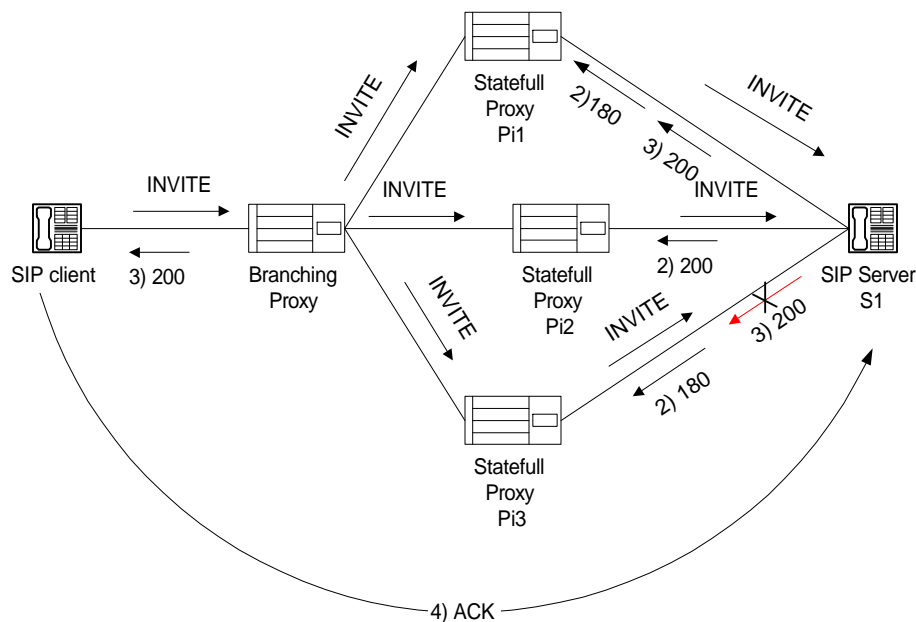
Figure 6: Isomorphic requests w/different 'Via' list.

In the situation pictured above, the server S1 receives three INVITE messages due to the forking proxies. It responds to each with 180 Ringing and later with 200 OK, sending each to the appropriate path according to the 'Via' fields from original request. The response to Pi3 was lost. However since the remaining two OKs safely reached proxy P the client will be forwarded the OK and will respond with an ACK. When this ACK reaches the server the connection is up (the server WILL NOT try to retransmit OK to Pi3). Neither P nor Pi3 is obligated to retransmit the original INVITE that would trigger retransmission of the previous response from the server. The Pi3 may be forever stuck waiting for final response from S1.

The solution: include 'Via' to the list of the fields that must be checked to determine whether the request is isomorphic. Specifically, define two requests to be isomorphic if both have the same Call-ID, To, From, and list of vias with branch ID's which are all identical. That will assure that the responses are sent for each received request since the server will be able to distinguish among different requests.

For example, the following requests are isomorphic:

INVITE user@company.com SIP/2.0 Via: 10.0.0.1 Via: 10.0.0.2; branch = 12a4 Via: 10.0.0.3 Via: 10.0.0.4; branch = 12b6	INVITE user@company.com SIP/2.0 Via: 10.0.0.19 Via: 10.0.0.2; branch = 12a4 Via: 10.0.0.26 Via: 10.0.0.4; branch = 12b6

The following requests are NOT isomorphic by our revised definition:

INVITE user@company.com SIP/2.0 Via: 10.0.0.1 Via: 10.0.0.2; branch = 96fe Via: 10.0.0.3 Via: 10.0.0.4; branch = 12b6	INVITE user@company.com SIP/2.0 Via: 10.0.0.19 Via: 10.0.0.2; branch = 12a4 Via: 10.0.0.26 Via: 10.0.0.4; branch = 12b6

Require the proxy to check the 'Via' fields in the request for branch-IDs. If any branch-ID is found, indicating the request was forked and may merge at the same server, require the proxy to retransmit request until final response is returned, even if a provisional response is provided. That will prevent the proxy from being stuck with pending transaction possible forever (normally such a case would be prevented since the server would have never received an ACK if OK was lost and would have retransmit OK.)

The solution proposed above requires further specification of proxies behavior. If a proxy creates list of pending, isomorphic requests based on Via list in the request (in addition to Call-ID, Cseq, From and To fields) it has two options for processing them.
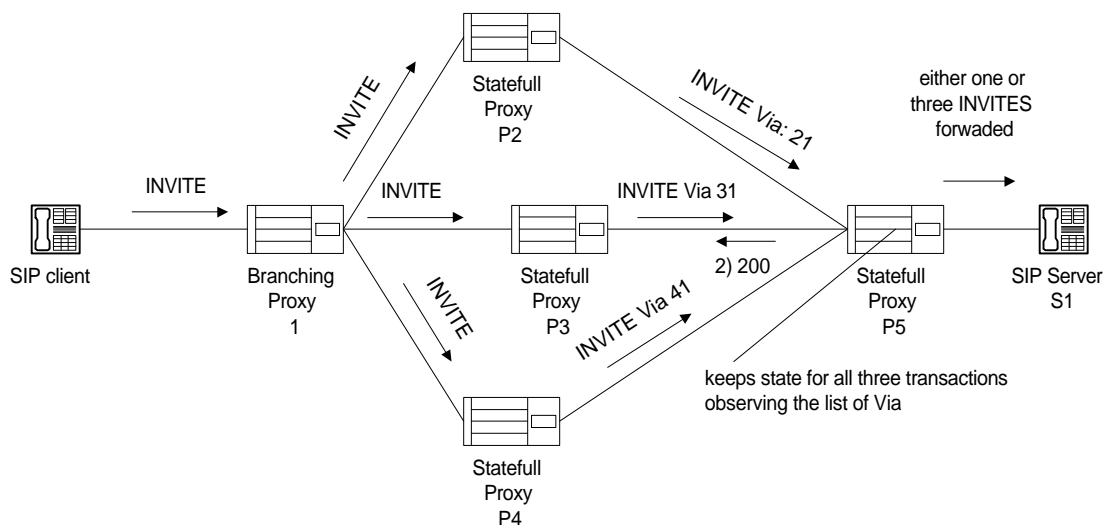


Figure 7: Processing semi-isomorphic requests.

In one implementation, the proxy may forward each received request to the server and create as many pending transactions as requests. The server would likewise detect three distinct transactions and would response to each with identical responses, with vias modified accordingly. Upon reception of the final response from the server, the proxies need only forward each of the three responses upstream after removing itself from the Via list. Therefore, each stateful proxy along the path will receive a final response to an INVITE it transmitted.

In another implementation, the proxy may forward only one request but keep state for ALL pending transactions. Upon reception of final response from the server the proxy SHOULD forward it upstream along ALL recorded in paths after modifying the via list for all but the very first transaction, since the response to the INVITE issued by the proxy will contain the via list identical to the first request.

The proxy SHOULD NOT forward CANCEL requests to the server until it received such a request for ALL pending transactions (created when original INVITEs were received). Otherwise one CANCEL would terminate all transactions as the server has only one pending transaction. However the proxy SHOULD respond to the received CANCEL with 200 final response. Likewise, the UAS should follow these guidelines as well for handling the CANCEL request.

Call-Leg Determination
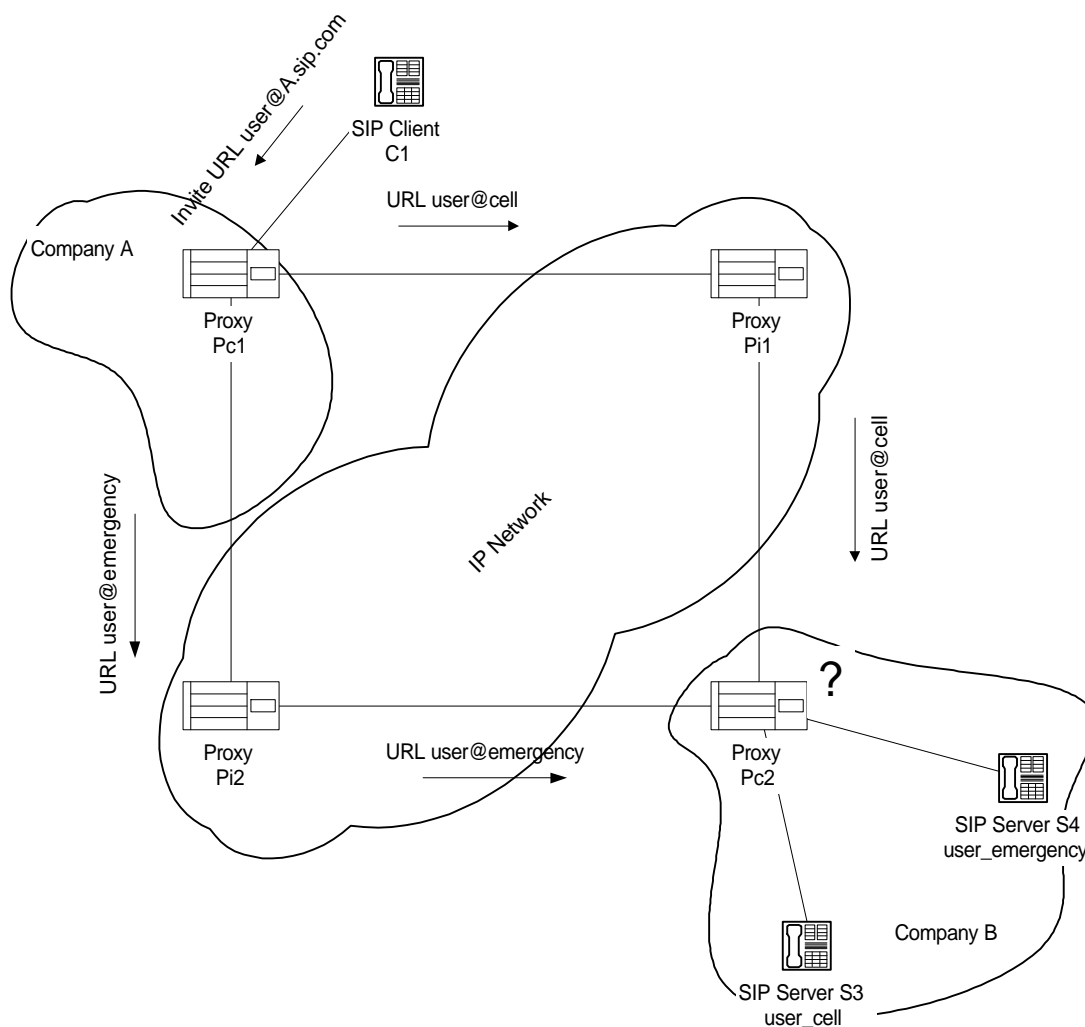
Figure 8: Improper Call-Leg determination.

The isomorphic requests are defined: 'Two requests or responses are defined to be isomorphic for the purposes of this document if they have the same values for the Call-ID, To, From and CSeq header fields. In addition, requests have to have the same Request-URI'. The isomorphic requests should trigger the retransmission of the last response. However when the behavior of the server is defined the Request-URL is never checked, therefore two requests with different Request-URLs and all other fields identical would trigger the retransmission. Although it is not a problem in most cases, the call setup may break down. In the scenario outlined above C1 makes the call to user@A.sip.com. The user is currently registered as being available at either user@emergency.com or user@cell.com. The proxy Pc1 tries both locations in parallel and both requests subsequently merge at Pc2. Here, the user is registered as:

- If user@cell.com contact abc@B.voicemail.com

- If user@emergency.com contact xyz@B.secretary.com

It is clear that depending on the SIP URL different locations should be contacted. However in the outlined scanario only the first INVITE request that reached Pc2 will be served properly and the second will be treated as retransmission since Call-ID, Cseq, To and From fields are identical.

The solution: Call-Leg should be identified by SIP URL as well.

DNS with built-in load balancing.

Special considerations should be applied when dealing with load balancing DNS servers. In particular, the response from such a server may yield different results for the same Request-URI. There's a potential for retransmissions and/or responses to go via distinct physical proxy resulting in malfunctioning due to ufinished transactions and/or Calls.

ACK

Clarification only: after succesful exchange of INVITE and OK the ACK from the client has been lost. Shortly the client decides to send new INVITE with different SDP (and of course higher Cseq). The server should not retransmit the original OK (the one for which an ACK was lost) but rather respond with an OK for the new INVITE. We call this an "implicit ACK".

Registering at a different domain

16.1 Any user wishing to receive his calls while in different domain needs to register with the default server *bell-tel.com* and with the'visiting' server *example.com*

```
C->S: REGISTER sip:bell-tel.com SIP/2.0
          Via: SIP/2.0/UDP saturn.bell-tel.com
          From: sip:watson@bell-tel.com
          To: sip:watson@bell-tel.com
          Call-ID: 70710@saturn.bell-tel.com
          CSeq: 3 REGISTER
          Contact: sip:tawatson@example.com
```

The problem: the registration message sent to *example.com* would have *tawatson@example.com* in the 'To' field. It's possible that such a record is already being used by someone else. The SIP URL *Watson* owns is distinct only if both user and hostport parts are preserved, combination of a user part with different hostport does not guarantee to be globally unique. There's no simple solution to that problem.

The improvement (not complete solution): when in doubt (which is pretty much always) the 'To' field in the registration request with the different server could be: *userinfo_hostport@example.com* where *userinfo* and *hostport* are identical to the *Watson's* SIP URL. Obviously, the same URL would be included in the Contact field of the registration sent to the default registrar. However, there's still no guarantee that new SIP URL is globally unique.

The difference between the Client and the Server

The states of the calls are different depending on whether the state resides on the server or the client. The distinction is necessary since e.g. BYE issued by the server brings down the Call-LEG only when the BYE issued by the client brings down the whole Call. Therefore the Call is up until the Client hangs up.

When to start the RTP Stream

Open issues