SIP 协议深入介绍

网络事业部软交换开发部 王笑蓉

1．SIP 简介

SIP（Session Initiation Protocol）是应用层控制协议，可以创建，修改，以及终止一个多媒体会话。它具有以下几个主要功能：

User location： 确定通信中的终端位置
User availability： 确定被叫方是否愿意进行通信
User capabilities： 确定用于通信的媒体类型及参数
Session setup： 建立会话各方的会话参数
Session management： 终止会话，修改会话参数

SIP 协议需要和其他 IETF 协议一起来构成一个完整的多媒体通信构架。这些协议有：

RTP(Real Time Transport)：传输实时数据，提供 QoS 反馈信息
RTSP(Real Time Streaming protocol)：控制流媒体的传送
MEGACO(Media Gateway Control Protocol)：控制媒体网关
SDP（Session Description Protocol）：描述多媒体会话

1．1 SIP 协议结构

SIP 协议的行为模型可以用几个分层的相对独立处理阶段来描述：

1．语法及编码层

2．传输层

定义了客户端如何通过网络发送请求及接收响应，以及服务器端如何接收请求并发送响应。所有 SIP 逻辑实体都包含此层。

3．事务层

事务层处理应用层请求或响应消息的重发，响应与请求的匹配以及应用层的超时。一个 SIP 事务由一个请求和对该请求的所有响应构成，这些响应分临时响应（provisional response）和最终响应(final response)。对于 INVITE 事务，对应于非 2xx 响应的 ACK 确认消息也属于该事物，而对应于 2xx 响应的 ACK 确认消息则不属于该 INVITE 事物。

UA 以及 stateful proxy 均包含事务层，而 stateless proxy 不包含事务层

一个事物根据逻辑功能分为客户事务(client transaction)和服务器事务（server transaction）。客户事务和服务器事务是存在于 UA 及有状态代理服务器(stateful

proxy server)的逻辑功能，stateless proxy 不存在客户机及服务器事务，即不包含事务层

客户机事务从事物用户 TU（UA 或 stateful proxy）接收请求，并把此请求可靠传输到服务器事务。客户机事务还负责从传输层接收来自服务器的响应，滤除响应的重发及不允许的响应（如对 ACK 请求的响应）。此外，对于 INVITE 请求，还负责对除 2xx 外的 final response（3xx-6xx）产生 ACK 确认。

服务器事务负责从传输层接收来自客户机的请求消息，并传递给 TU，服务器事务滤除请求的重发。服务器事务还负责从 TU 接收响应，并传递给传输层发出。此外，对于 INVITE 事务，还负责吸收对除 2xx 外的 final response（3xx-6xx）产生 ACK 确认。

2xx 响应及与之对应的 ACK 确认采用特殊的端到端处理方式。2xx 响应的重发由 UAS 负责，与 2xx 响应对应的 ACK 确认仅由 UAC 产生。这种端到端的处理方式使得呼叫方能知道所有接受此呼叫的用户。因此，2xx 响应的重发及对应 2xx 的确认 ACK 分别由 UAS core 和 UAC core 而不由事务层处理。UAC 及 UAS 间的 Proxy 仅转发 2xx 响应及其对应的 ACK 确认。

客户事务收到传输层来的 2xx 响应后传递给 TU，TU 如何处理取决于 TU 是 Proxy core 还是 UAC Core，UAC Core 对 INVITE 事务产生 ACK(non-INVITE 事务则无需 ACK 确认)，而 Proxy 仅 Forward 响应。

4．事务用户层

所有 SIP 逻辑元素，如 UAC，UAS，stateful proxy，stateless proxy，registrar 均包含一个 core 以示区别。除 stateless proxy 外的所有 SIP 逻辑实体 core 均是事务用户（TU）。

1．2 SIP 消息

SIP 消息分为从客户机到服务器的请求消息以及从服务器到客户机的响应消息。两种类型的消息结构相似，虽然语义不尽相同。两种消息都有一个起始行（start line）；一个或多个头字段（ head fields）;一个空行（empty line）表示头部的结束；还包括一个可选择的消息体（message-body）。

```
generic-message   =   start-line
                      *message-header
                      CRLF
                      [ message-body ]
start-line            =   Request-Line / Status-Line
```

请求消息 Request 中请求行：

Request-Line = Method SP Request-URI SP SIP-Version CRLF

请求方法（Method）有六种：

REGISTER 用来登记；INVITE,ACK,CANCEL 用来建立对话；BYE 用来结束对话；OPTIONS 用来咨询服务器的能力。

Request-URI: 是 SIP 或 SIPS URI，也支持“ tel ”URI，代表 request 所请求的用户或服务器。它的初始值可以被设成与 to field 中 URI 一致；一个比较特殊的情况是 register，它是被设成所要登记地址的 location service 的域。在呼叫寻址的过程中可以改写,在寻址过程中有详细介绍。

SIP-Version：表示 request 和 response 所使用的 SIP 版本。

响应消息 Response 中状态行：

Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF

状态码（Status-Code）有：

1xx: Provisional -- request received, continuing to process the request;

2xx: Success -- the action was successfully received, understood, and accepted;

3xx: Redirection -- further action needs to be taken in order to complete the request;

4xx: Client Error -- the request contains bad syntax or cannot be fulfilled at this server;

5xx: Server Error -- the server failed to fulfill an apparently valid request;

6xx: Global Failure -- the request cannot be fulfilled at any server.

消息头字段（head fields）：

header = "header-name" HCOLON header-value *(COMMA header-value)

（字段值，参数名，参数值大小写无关）

(请求消息必须包括 Via，To，From，CSeq，Call-ID，Max-Forwards 六个字段)

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 ; received=192.0.2.1

Via head field Via 头字段保存所经过 SIP 网元的主机名或网络地址（可能还有端口号），消息中的所有 Via 头字段对请求消息而言，从下至上依次表示到当前所在 SIP 网元为止，请求消息所经过的路径；对响应消息而言，从上至下依次表示从当前网元开始，响应所应遵循的路径。

UAC 在发出请求时，他在其请求中插入 Via 字段，该字段包含 UAC 的主机名或网络地址，在消息的传递过程中经过的每个 SIP 网元均在其转发的请求中插入 Via 字段。同时当 SIP 网元收到上一跳 SIP 网元的请求消息时，在上一跳 SIP 网元所插入的 Via 字段中加入 **received** 参数，代表所收到来自上一跳 SIP 网元数据包的实际源地址（对于上一跳 SIP 网元具有多主机地址即 multi-homed host 时，可能会混淆所用的网络接口）。见 RFC3261 P180。

在 SIP 网元（UAC 或 Proxy）发出或转发请求消息时，在其插入的 Via 字段中必须包含 **branch** 参数，**该参数用于标识此请求消息所创建的事务**。UA 所发送的所有 request 的 branch 参数必须在时间和空间上都唯一；这里也有一个例外，CANCEL 以及非 2XX response 的 ACK。对于 CANCEL 来讲，与它所 cancel 的 request 的 **branch** 参数一致；此外，当 INVITE 请求其 final response 是 non-2xx 响应时，对应该 response 的 ACK 与此 INVITE 请求的 **branch** 参数一致。因而，CANCEL 以及非 2XX response 的 ACK 与其所 cancel 的 request 或对应的 INVITE 属于同一个事务。见 RFC 3261 P39 P122

branch 参数可以用做 loop detection，这时参数必须被分成两部分：第一部分符合一般的原则（对于 RFC3261，z9hG4bK），第二部分被用来实现 loop detection 以用来区分 **loop** 和 **spiral**。loop 和 spiral 均指 Proxy 收到一个请求后转发，然后此转发的请求又重新到达该 Proxy，区别是 loop 中请求的 Request-URI 以及其他影响 Proxy 处理的头字段均不变，而 Spiral 请求中这些部分必需有某个发生改变，spiral 发生的典型情况是 Request-URI 发生改变。Proxy 在插入 Via 字段前，其 **branch** 参数的 loop

detection 部分依据以下元素编码：To Tag，From Tag，Call-ID 字段，Request-URI，Topmost Via 字段，Cseq 的序号部分（即与 request method 无关），以及 proxy-require 字段，proxy authorization 字段。**注意**：request method 不能用于计算 branch 参数，比如 CANCEL 以及非 2XX response 的 ACK 与其所 cancel 的 request 或对应的 INVITE 属于同一个事务，即其 branch 参数相同。见 RFC3261 P22 P25 P39 P95 P105

此外 Via 字段还包含 SIP 协议版本以及消息传输所用的传输协议。

Max-Forwards: 70

Max-Forwards head field 表示 request 到达 UAS 的跳数的限制。是一个整数，经过每一跳时减去一。如果 Max-Forwards 已经是零，可是 request 还没有到达目的地，则就会产生一个 483(too many hops)响应。

To: Bob <sip:bob@biloxi.com>

To head field 预置 request 所希望的的"logical" recipient（REGISTER 事务除外），它可能是，也可能不是 request 的最终 recipient。To head field 在 REGISTER 事务中表示要在 Location Server 中登记谁的地址，即地址绑定中的 address-of-record（vs. contact address）。

To head field 可以有一个 tag 参数，to tag 代表 dialog 的对等参与者（peer）。在 UAC 发出一个初始 Dialog 的请求（如 INVITE）时，即发出 out-of-dialog 请求时，由于 dialog 还没有建立，不含 to tag 参数。当 UAS 收到 INVITE 请求时，在其发出的 2xx 或 101-199 响应中设置 to tag 参数，与 UAC 设置的 From Tag 参数以及 Call-ID（呼叫唯一标识）一起作为一个 Dialog ID（对话唯一标识，包含 To tag，From Tag，Call-ID）的一个部分。RFC3261 规定只有 INVITE 请求与 2xx 或 101-199 响应可以建立 Dialog（由 101-199 响应创建的 Dialog 称为 early dialog）。见 RFC3261 P70

见 RFC3261 P36，P159，P178

From: Alice <sip:alice@atlanta.com>;tag=1928301774

From head field 是 request 发起者的 logical 标识（REGISTER 事务除外）。from head field 在 REGISTER 事务中表示谁负责这次登记，如果由发起者负责该字段也就是地址绑定中的 address-of-record（vs. contact address）。From head field 与 To head field 类似，包含一个 URI 以及可选的 display name，但 From head field 不能是 UA 的主机 IP 地址或主机名，因为 From 字段只能是逻辑名（From 字段用于 SIP 元素决定对该请求采用何种处理规则，例如对 From 字段代表的用户进行自动拒绝）。

From 字段必须包含 tag 参数，在 UAC 发出一个 out-of-dialog 请求（对话建立请求）时，必须设置一个唯一的 tag 参数，作为 Dialog ID 的一个部分。

见 RFC3261 P37，P159，P172

Call-ID: a84b4c76e66710

Call-ID head field 是一个邀请(Invitation)或来自同一个 UAC 用户的所有登记请求(Registeration，包括更新登记，取消登记)以及由此产生的一组响应的唯一标识。一个邀请可以建立多个 Dialog（当被叫用户有多个联系方式时），这成为 Forking，因而 Call-ID 只是一次呼叫邀请的唯一标识，Call-ID 与 UAC 在发出请求中设置的 From Tag 字段以及 UAS 在其相映中设置的 To Tag 字段三者一起作为一个 Dialog-ID。

在一个 Dialog 中，所有的 requests 和 responses 的 Call-ID 必须一致

同一 UA 的每一个 register 的 Call-ID 必须一致。

见 RFC3261 P37，P159，P166

CSeq: 314159 INVITE

CSeq head field 用于在同一个 Dialog 中标识及排序事务（transaction）以及区分新的请求与请求的重发。

（请求的重发-retransmission 与 re-REQUEST 不同，前者消息完全相同，后者则与原请求不同。例如 re-INVITE 能用于改变媒体会话的参数；在原请求产生 4xx 响应时，按照响应内容再 re-REQUEST，见 RFC3261 P45）。

CSeq 包括顺序号和方法（method），方法必须和它所对应的 request 相匹配。对于 out-of-dialog 的非 register request，取值任意。

对于 dialog 内的每一个新的 request（如 BYE,re-INVITE,OPTION），Cseq 的序号加 1。但是对于 CANCEL,ACK 除外。对于 ACK 而言，Cseq 的序号必须与其所对应的 request 相同。对于 CANCEL 而言，Cseq 的序号也必须与其 cancel 掉的 request 相同。

注意：在同一个对话中的 UAC 和 UAS 分别维护自己的 CSeq 序号，他们发出请求的 CSeq 序号是不相关的。见 RFC3261 P218

见 RFC3261 P38，P159，P170，P218

Contact: <sip:alice@pc33.atlanta.com>

contact head field 对于非 Register 事务，Contact header field 主要提供了 UAC 或 UAS 的直接联系 SIP URI，UAC 在发出的对话建立（out-of-dialog）INVITE 请求的 Contact 字段中提供自己的直接联系 SIP URI，在 UAS 收到该请求后在其发出响应的 Contact 字段中提供自己的直接联系 SIP URI，这样在建立对话后，UA 间可以通过对方的直接联系 SIP URI 绕过 Proxy 直接发送请求。

对于 Register 事务，表示地址绑定中的 contact address（vs. address-of-record）

Content-Type: application/sdp

Content-type header field 主要表示发给接收器的消息体的媒体类型。如果消息体不是空的，则 Content-type header field 一定要存在。如果 Content-type header field 存在，而消息体是空的，表明该类型的媒体流长度是 0。

Content-Length: 142

Content-length header field 表示消息体的长度。是八位组的十进制数。

(Alice's SDP not shown)

消息体 SIP Message Body：

编码方式主要是由头部确定，现在为一般为 SDP。

//SIP Request line/Status line ＋ Headers ＋ CRLF

v=0 //版本号为 0

o=UserA 2890844526 2890844526 IN IP4 here.com

//建立者用户名＋会话 ID＋版本＋网络类型＋地址类型＋地址

s=Session SDP //会话名

c=IN IP4 100.101.102.103 //连接信息：网络类型＋地址类型＋地址

t=0 0 //会话活动时间 起始时间＋终止时间

m=audio 49172 RTP/AVP 0

//媒体描述：媒体＋端口＋传送＋格式列表

音频 ＋ 端口 49172 ＋ 传输协议 RTP ＋ 格式 AVT，有效负荷 0（u 率 PCM 编码）

a=rtpmap:0 PCMU/8000

//0 或多个会话属性： 属性 ＋ 有效负荷＋ 编码名称 ＋ 抽样频率。

// rtpmap ＋ 0 型 ＋ PCMU ＋ 8KHz

…… //其他 SDP 描述

SIP 通过 offer/answer 模型来进行 UA 间的 SDP 会话描述交互：

a) UA 发出会话描述，称为 offer，它包含会话描述的提议，代表期望建立的通讯方式（aduio,video,games），通讯方式的参数（如 codex types）以及用于接收 answer 方媒体的地址。
b) 另一方 UA 接到 offer，并用另一个会话描述予以响应，代表此 UA 接受的通讯方式，参数以及用于接收 offer 方媒体的地址。
c) UA 双方会话描述的 offer/answer 交互在一个建立的对话内进行，当一个 INVITE 请求建立多个对话时，每个对话的 UA 间都要有 offer/answer 交互。
d) 会话描述 SDP 只能由 INVITE，response，ACK 的消息体携带，SDP 的 offer/answer 交互有以下几种方式：
    i. INVITE 携带 SDP 消息体发出 offer，2xx response 携带 SDP 消息体 answer
    ii. 2xx response 携带 SDP 消息体发出 offer，ACK 携带 SDP 消息体 answer

2．呼叫过程概述

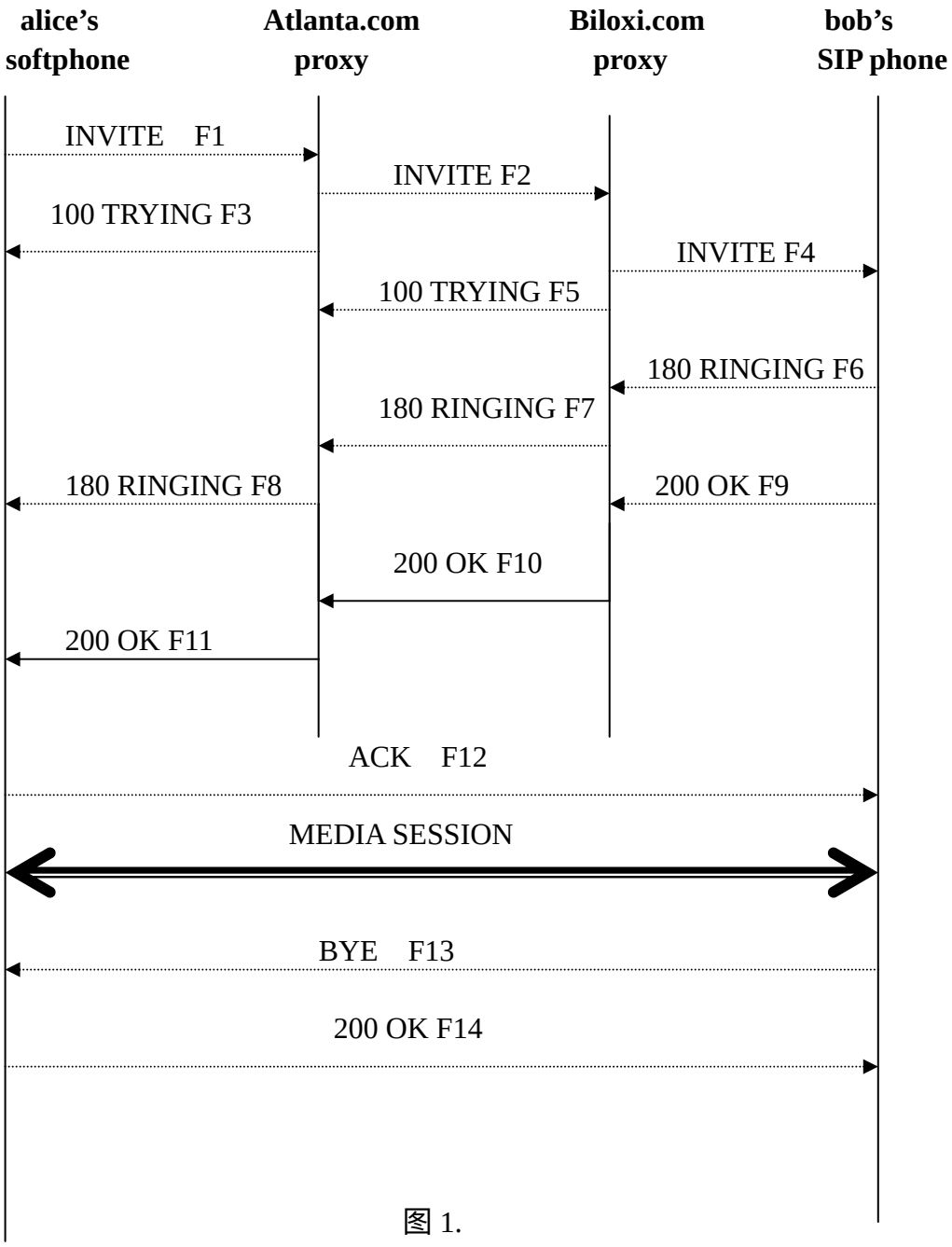


图 1.

上图中，Alice 通过其 PC 上的 SIP 应用程序呼叫 Bob，Bob 使用一个在 Internet 上的 SIP Phone。Alice 通过 Bob 的 SIP 标识，即 SIP URI sip:bob@biloxi.com 来呼叫 Bob，其中 biloxi.com 是 Bob 的 SIP 服务提供者（SIP service provider）所在的域。同样 Alice 也有一个 SIP URI sip:alice@atlanta.com。

1) Alice’s softphone 并不知道 Bob 以及 biloxi.com 域中 SIP 代理服务器的具体位置，因而 Alice 发送 INVITE 请求到 Alice 所在的 atlanta.com 域中 SIP 代理服务器，该代理服务器的地址一般配置在 Alice 的 softphone 中，或者可以通过 DHCP 查找。

Alice 作为 UAC 在发出请求时，他在其请求中插入 Via 字段，该字段包含 UAC 的主机名或网络地址。Via 字段的 branch 参数用于标识此请求消息所创建的事务。

此时，Alice 作为 UAC 发出的是一个 out-of-dialog 请求（对话建立请求），必须在 From 字段中设置 tag 参数，作为 Dialog ID 的一个部分。

Alice 作为 UAC 在 Contact 字段中提供自己的直接联系 SIP URI。

F1 INVITE Alice -> atlanta.com proxy

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)

2) Atlanta.com 代理服务器收到 INVITE 请求后发送 100（Trying）响应给 Alice，该响应包含与 INVITE 同样的 To，From，Call-ID，CSeq 以及 Via 字段的 branch 参数，通过这些字段建立响应与请求间的对应关系。

F2 100 Trying atlanta.com proxy -> Alice

SIP/2.0 100 Trying
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0

3) Alice 在 atlanta.com 域中的 SIP 代理服务器不负责域 biloxi.com，因而通过某种特殊形式的 DNS 定位 biloxi.com 域中的 SIP 代理服务器，从而获得其 IP 地址，并向其转发 INVITE 请求。

在 atlanta.com 代理服务器收到上一跳 Alice UAC 的请求消息时，在 Alice UAC 所插入的 Via 字段中加入 **received** 参数，代表所收到来自上一跳 SIP 网元数据包的实际源地址。

在 atlanta.com 代理服务器向 biloxi.com 代理服务器转发 INVITE 请求前，在 INVITE 请求的 Via 字段中加上自己的地址（INVITE 请求的 Via 字段已包含 Alice 的地址），同时 Max-Forwards 由于经过了一跳而减一。

F3 INVITE atlanta.com proxy -> biloxi.com proxy

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
Max-Forwards: 69
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)

4) 在 biloxi.com 代理服务器收到 atlanta.com 代理服务器转发的 INVITE 请求后，向其返回 100（Trying）响应以通知收到并正在处理该请求。

F4 100 Trying biloxi.com proxy -> atlanta.com proxy

SIP/2.0 100 Trying
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0

5) biloxi.com 域中的 SIP 代理服务器通过查询其 Location Service 数据库获得 Bob 的当前实际地址，并改写 Request-URI。（Bob 通过 Registrar 服务器在 Location Service 中建立了其 SIP URI 与实际地址的映射）。

biloxi.com 代理服务器在转发其收到的 INVITE 请求至 Bob 的 SIP Phone 前，在 Via 字段中加入其地址。

F5 INVITE biloxi.com proxy -> Bob

INVITE sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
Max-Forwards: 68
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)

6) Bob 的 SIP Phone 收到转发来的 INVITE 请求后返回 180（Ringing）响应消息，该响应将沿着 INVITE 请求的传递路径返回 Alice 的 softphone，途中的每个代理通过消息中的 Via 字段决定转发的目的地，并把包含自己的地址的 Via 字段从消息中删除。

Bob 作为 UAS 发出的 180（Ringing）响应消息拷贝其收到的 INVITE 请求的 Via，To，From，Call-ID，CSeq，Record-Route 等头字段，并在 To 头字段中加入 Tag 参数（含有 Tag 参数的 To，From 字段与 Call-ID 一起作为该对话 Dialog 的唯一标识）。此外，响应消息的 Contact 头字段放入 Bob SIP Phone 的直接联系 URI 地址。

F6 180 Ringing Bob -> biloxi.com proxy

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F7 180 Ringing biloxi.com proxy -> atlanta.com proxy

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F8 180 Ringing atlanta.com proxy -> Alice

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

7) 当 Bob 拿起话筒时，其 SIP Phone 将发出 200（OK）响应消息，该响应与 INVITE 请求类似，均包含一个 SDP 媒体描述的消息体，代表 Bob 想建立的会话的媒体描述。上述过程提供了会话各方的媒体协商机制。

如果 Bob 不想应答或占线时将发出错响应而不是 200（OK），这将不会建立会话。

除 DNS 及 Location Service 等功能外，代理服务器也可以具有“柔性路由决策”功能，例如当 Bob 返回 486（Busy Here）响应时，biloxi.com 代理服务器可以把 INVITE 请求代理到 Bob 的其他联系方式，如语音邮件。SIP 代理服务器也可以发送请求消息到一组 Bob 登记的地址。

与 180 响应类似，200（OK）响应将沿着 INVITE 请求的传递路径返回 Alice 的 softphone，途中的每个代理通过消息中的 Via 字段决定转发的目的地，并把包含自己的地址的 Via 字段从消息中删除。

F9 200 OK Bob -> biloxi.com proxy

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1 ;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)

F10 200 OK biloxi.com proxy -> atlanta.com proxy

SIP/2.0 200 OK
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)

F11 200 OK atlanta.com proxy -> Alice

SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)

8) Alice 收到 Bob’s SIP Phone 发出的 200（OK）后将发出 ACK 确认请求消息，因为 Alice 发出的 INVITE 请求和 Bob 发出的 180(Ringing)响应中的 Contact 字段中给出了这两个 UA 间的直接联系地址，因而 ACK 消息可以绕过代理直接发送到 Bob，这样就完成了建立 SIP 会话的 INVITE/200/ACK 三次握手。

该 ACK 请求属于前面已建立的对话（相同对话标示：Call-ID，From| Tag，To|Tag），但与 BYE 或 re-INVITE 不同，ACK 和 CANCEL 请求与其对应的 INVITE 请求的 CSeq 序号相同。

F12 ACK Alice -> Bob

ACK sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 ACK
Content-Length: 0

9) INVITE/200/ACK 三次握手后，Alice 与 Bob 间的媒体会话建立。会话双方按照在 SDP 媒体协商中所确定的格式发送媒体包，一般来说端到端的媒体包路由与 SIP 信令消息路由不同。

Alice 或 Bob 均可以改变媒体会话的参数，这通过 re-INVITE 请求实现，在其 SDP 消息体中包含新的媒体描述，该 re-INVITE 对应前面已存在的一个对话（通过对话标示：Call-ID，From| Tag，To|Tag），这样对方知道是修改一个已存在的会话而不是建立一个新会话。对方发出 200(OK)响应来接受绘画改变，而发起方以 ACK 确认，如对方不同意改变，则发出 488(Not Acceptable Here)，发起方同样以 ACK 确认。re-INVITE 的失败不会导致原会话的失败。

因为 re-INVITE 与原 INVITE 属于同一个 Dialog 对话，其 Cseq 头字段的序号为原序号加一。

10) Bob 挂机，发出 BYE 请求消息，消息绕过代理直接发往 Alice。此外，Alice 作为 UAC，Bob 作为 UAS 分别维护其私有的 CSeq 编号空间，因而其所发出请求的 CSeq 序号是不相关的，假如 BYE 请求由 Alice 发出，则其 CSeq 序号为 314159+1=314160。

因为由 INVITE 的被叫方 Bob 发出 BYE 请求，其 To 和 From 字段互换。

F13 BYE Bob -> Alice

BYE sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
Max-Forwards: 70
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0

11) Alice 收到 BYE 消息后发出 200（OK）响应以确认收到了 BYE 请求，从而终止整个会话以及 BYE 事务。Bob 收到 200（OK）响应后无需发 ACK 确认，基于 INVITE 请求的可靠性考虑，ACK 只用来确认 INVITE 请求产生的响应。

F14 200 OK Alice -> Bob

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0

3．登记过程概述

SIP Proxy 及 Redirect 服务器负责接收发往其负责域中 UAs 的请求消息，并通过查询该域 **Location Service** 中的 **address-of-record vs. contact addresses** 地址绑定（一个 address-of-record 可绑定多个 contact addresses），找到与消息 Request-URI 相匹配的 address-of-record 后，Proxy 或 Redirect 服务器把消息 Forward 到此 address-of-record 对应的 contact addresses（此时消息的 Request-URI 改写为这些 contact addresses）。

登记服务器（Registrar）是一种特殊的 UAS，它负责为所在域中的 location server 创建地址绑定。在很多情况下，某域中的登记服务器和代理服务器作为不同逻辑实体存在于同一个网络设备，如果他们处在不同的网络设备，注册用户也可通过代理服务器访问登记服务器。

Location Service 必须保证域中的登记服务器能够对其进行读写操作，而域中的代理服务器，重定向服务器能进行读操作。

REGISTER 请求消息不创建 Dialog，REGISTER 请求消息头字段必须包含以下部分（除 Contact 外）

Request-URI：表示 location server 所在的域，如 sip:chicago.com。SIP URI 中的“ userinfo”“ @ ”不能出现在 Request-URI 中。

To：表示要登记的 address-of-record，即代表要创建，查询，修改谁的 contact addresses 地址绑定，其形式为 SIP URI。

From：表示负责此登记者的 address-of-record，除由第三方登记的情况外，From 字段与 To 字段相同。

Call-ID：由同一 UAC 向同一 registrar 的所有登记刷新请求推荐具有相同 Call-ID。

CSeq：具有相同 Call-ID 的登记请求 CSeq 序号依次加一。

Contact：

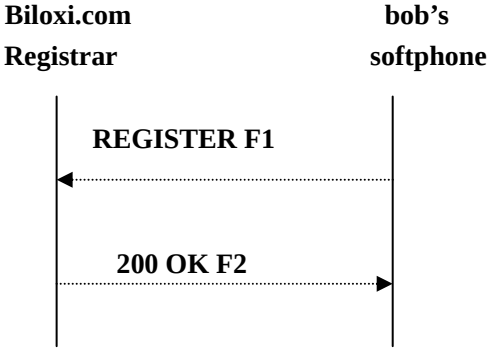
对于新建或添加登记请求消息，代表要新建，添加与 To 字段的 address-of-record 绑定的 contact address，返回的 2xx 响应的 contact 字段包含所有登记的 contact addresses)。

对于登记查询请求消息，不包含该字段，但返回 2xx 响应的 contact 字段（一个或多个）代表从 location service 中查询到的地址绑定中与 To 字段的 address-of-record 相对应的 contact addresses。

对于删除登记请求消息（expires 参数为 0），代表要删除与 To 字段的 address-of-record 绑定的 contact address。contact 字段为 * 代表要删除与 To 字段的 address-of-record 绑定的所有 contact addresses(此时 Expires 头字段必需为 0)。

Contact 字段的 expires 参数代表地址绑定的有效期限，当其为 0 时代表删除绑定，expires 参数也可用 Expires 头字段代替。

下面是登记过程的实例：



F1 REGISTER Bob -> Registrar

REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0

F2 200 OK Registrar -> Bob

SIP/2.0 200 OK
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
;received=192.0.2.4 这是传输层加的
To: Bob <sip:bob@biloxi.com>;tag=2493k59kd
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0

4．对话内（Within Dialog）消息的发送及路由处理

4．1 UAC 与 UAS 对话的建立及状态

见 RFC3261 P69-72

对话（Dialog）代表 UA 间端到端的 SIP 联系，对于某个 UA 而言，一个对话具有唯标识 **dialog-ID**，它由 **Call-ID**，**Local Tag**，**Remote Tag**。

RFC3261 规定 Dialog 的建立过程是 UAC 发出 out-of-dialog INVITE 请求，然后 UAS 返回 2xx 或 101-199 响应（由 101-199 响应创建的 Dialog 称为 **early dialog**）。

当 out-of-dialog INVITE 消息路由中的某个 SIP 代理服务器希望对话建立后的所有请求消息 **(with in dialog)**均通过通过此 Proxy 时，则在其转发的 INVITE 消息中加入 Record-Route 字段，填入能解析为其主机名或 IP 地址的 URI。

UAS 返回的 2xx 或 101-199 响应拷贝了请求消息的 Record-Route 头字段，加上 To Tag 参数，并在 Contact 字段中设置自己的直接联系 SIP URI。建立对话后，UAS 构造 UAS 端的**对话状态**（**state of dialog**），UAS 在整个对话中均保存及维护此状态，对话状态包含以下元素：

a) **secure flag**：当请求消息通过 TLS 到达时，secure flag 为 TRUE。

b) **route set**： 设为收到的 INVITE 请求的 Record-Router 头字段中的 URIs，保持原有顺序。

c) **remote target**：INVITE 请求的 Contact 字段。

d) **remote sequence number**：INVITE 请求的 CSeq 字段序号。
在同一个对话中的 UAC 和 UAS 分别维护自己的 CSeq 序号，他们发出请求的 CSeq 序号是不相关的。

e) **local sequence number**：此时为空

f) **Dialog-ID**：

- **i.** **Call-ID**：INVITE 请求的 Call-ID 字段
- **ii.** **Local Tag**：响应的 To Tag
- **iii.** **Remote Tag**：INVITE 请求的 From Tag

g) **Remote URI**：INVITE 请求的 From 字段

h) **Local URI**：INVITE 请求的 To 字段

UAC 收到 **UAS** 建立对话的 2xx 或 101-199 响应后，UAC 构造 UAC 端的**对话状态**（**state of dialog**），UAS 在整个对话中均保存及维护此状态，对话状态包含以下元素：

i) **secure flag**：当请求消息通过 TLS 到达时，secure flag 为 TRUE。

j) **route set**： 设为收到响应消息的 Record-Router 头字段的 URIs，但顺序反向。

k) **remote target**：响应消息的 Contact 字段。

l) **local sequence number**：请求消息的 CSeq 字段序号。
在同一个对话中的 UAC 和 UAS 分别维护自己的 CSeq 序号，他们发出请求的 CSeq 序号是不相关的。

m) **remote sequence number**：此时为空

n) **Dialog-ID**：

- **i.** **Call-ID**：请求的 Call-ID 字段
- **ii.** **Local Tag**：请求的 From Tag
- **iii.** **Remote Tag**：响应消息的 To Tag

o) **Remote URI**：请求的 To 字段

p) **Local URI**：请求的 From 字段

4．2 对话建立后，UAC 的 within dialog 请求消息处理

见 RFC3261 P72-74

对话建立后，作为对话一方的 UA 维护各自的对话状态，一般是在作为 UAC 发出**对话内(with in dialog)**请求（CANCEL，ACK 除外）后 **local sequence number** 加一，作为 UAS 收到请求后 **remote sequence number** 设置为该请求的 CSeq 字段序号。

当对话中的 UA 作为 UAC 发出请求消息时，请求消息中的头字段根据该 UA 的对话状态按照对应关系进行设置：

对于非 CANCEL，ACK 请求，CSeq 序号设为 **local sequence number** 加一，对于 CANCEL，ACK 请求，CSeq 序号设为与 **local sequence number** 相同。

UAC 用其对话状态中的 **remote target** 和 **route set** 来构造请求消息中的 **Request-URI** 和 **Router** 头字段，规则如下：

a) 当 **route set** 为空时，**UAC** 必须把消息的 **Request-URI** 中设置为 **remote target**。(如图 1 中的 BYE 请求)

b) 当 **route set** 为非空，且 **route set** 的第一个 URI 包含 **restrict router "lr"** 参数，则在必须把消息的 **Request-URI** 设置为 **remote target**，并依据 **route set** 在消息中加上 **Route** 头字段（如下例（1）U1 和例（2）U2 发出的 BYE 请求)。

c) 当 **route set** 为非空，且 **route set** 的第一个 URI 不包含 **restrict route "lr"** 参数，则必须把消息的 **Request-URI** 中设置为此 **route set** URI（截去不容许的 lr 等参数)，并依据 **route set** 在消息中加上 **Route** 头字段（除去第一个 **route set** URI)，此外把 **remote target** 加在 Route 头字段最后。

UAC 发送消息时要遵循以下原则：

a) 如果 route set 中的第一项是 strict router，消息的下一跳目的地置于 request 中的 request-URI。

b) 如果 route set 中的第一项是 loose router, 消息的下一跳目的地是 request 中的 route head field 中的第一项。

c) 如果不存在 route head field,则消息的下一跳目的地是 request 中的 request-URI。

4．3 Stateful Proxy 的请求消息处理

SIP 代理服务器的主要功能是为发往 UAS 的请求消息和发往 UAC 的响应消息进行路由，Proxy 在转发消息前可以对消息进行修改。**Stateful Proxy** 在收到新的请求后创建新的服务器事务（**Service Transaction**)，Stateful Proxy 其 **Proxy Core** 属于 **TU**，这里介绍的代理服务器行为是指 Proxy Core 的行为。

a. route information preprocessing

i. 代理服务器必须先检查 request 消息中的 request-URI，如果 request-URI 中的一个值是代理服务器先前放到 Record-route head field 中去的，则代理服务器将 Route head field 中的最后一项(一般是 UAC 的 remote target)放到 request-URI 中去，然后将这个值从 route head field 中去掉。理服务器再去处理这个修改过的 request.

ii. 如果 route head field 中的第一个值是这个代理服务器，就将它从 route head field 中去掉。

b. postprocessing routing information

如果 request 消息中有 route head field，则代理服务器检查它的第一个值，如果第一个值不含有 1r 这个参数（表明是 loose router）,代理服务器必须做如下操作：

首先，代理服务器将 request-URI（一般是 UAC 的 remote target）放到 route head field 中，作为它的最后一项；

其次，代理服务器将 route head field 的第一项放到 request-URI 中去，并将它从 route head field 中去掉。

c. determine next-hop address,port,and transport

i. 如果代理服务器将 request 送到一个 strict-routing element,则下一跳目的地是 request-URI，否则下一跳是 request 中的 route head field 中的第一项，

ii. 如果不存在 route head field，则消息的下一跳是 request 中的 request-URI（对于 within dialog request，request-URI 为 UAC 的 remote traget）。.

代理服务器还必须处理收到的 reponse.其中值得注意的一点是，代理服务器将 response 中的 via head field 中的第一项（即该代理服务器加上的 Via 字段）去掉。如果没有 via head field，则表明这个代理服务器是 response 的最终地，不再将它再往前送。

4．4　实例：

（1）U1 ⟶ P1 ⟶ P2 ⟶ U2

P1,P2 都是 loose router.

其中 U1 向其 outbound proxy P1 发出：

INVITE sip:callee@domain.com SIP/2.0
Contact: sip:caller@u1.example.com

P1 发现它并不负责 domain.com,它就查询 DNS，并将它发送到查到的 P2，在转发的消息中加上 record-route 头字段，填入能解析为 P1 主机名或 IP 地址的 URI。

INVITE sip:callee@domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p1.example.com;lr>

P2 得到了这个 request,它是负责 domain.com 域，所以它就去查询该域的 location service，用查到的 U2 登记的 contact address 改写 request-URI。此外，加了 record-route 头字段，由于没有 route head field，所以它解析新的 request-URI，并把它发送到那儿。

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>

在 u2.domain.com 域中的被叫方收到了 request，并且产生一个 response 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>

此时，在 u2.domain.com 中的被叫方 UAS 为建立的对话设置 UAS 端**对话状态（dialog state）**，其中 **remote target** URI 为：

sip:caller@u1.example.com

route set 是:

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

这个 response 通过 P2,P1,最后 U1 收到，U1 设置 UAC 端对话状态，其中 **remote target** URI ：

sip:callee@u2.domain.com and its

route set:

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

由于所有的 route head field 中的值都有 lr parameter, 所以 U1 产生 BYE request，其 **Request URI** 设为 U1 端对话状态的 **remote target:**

BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr>

根据前面所讲的路由规则，通过对 P1<sip:p1.example.com;lr>的 DNS 解析，U1 将它发送给 P1，P1 收到 request，发现它并不负责 Request-URI 指向的域，所以它并不重写 request-URI，此外 P1 发现 router head field 中的第一项是它自己,所以就将其从 route head field 中去掉（route information preprocessing），然后把它发送给 P2。

BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p2.domain.com;lr>

P2 依然发现它并不负责 Request-URI 指向的 u2.domain.com 域（P2 负责 domain.com 域），所以它并不重写 request-URI，并将 route head field 中对应 P2 的第一项去掉（route information preprocessing）。然后通过 DNS 查询负责 u2.domain.com 域的 Proxy，并发送过去。

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

（2） U1->P1->P2->P3->P4->U2

在此例中，对话已经建立，INVITE 消息路由中的四个代理服务器分别在转发的消息中加入了自己的 Record-Route 头字段，其中 P1,P2,P4 是 loose router，P3 是 strict router.

U2 接收到如下 INVITE 请求:

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>

U2 产生响应 200 OK。此时，U2 作为 UAS 为建立的对话设置 U2 **对话状态（dialog state）**，其中 **remote target** URI 为：

Contact: sip:caller@u1.example.com

route set 是:

(<sip:p4.domain.com;lr>,<sip:p3.middle.com>，
<sip:p2.example.com;lr>，<sip:p1.example.com;lr>)

这个 response 通过 P4，3，2，1，最后 U1 收到，U1 作为 UAC 设置 U1 对话状态，其中 **remote target** URI 设为返回响应的 contact 字段。

route set:

(<sip:p1.example.com;lr>,<sip:p2.example.com;lr>，
<sip:p3.middle.com>，<sip:p4.domain.com;lr>)

由于所有的 route head field 中的值都有 lr parameter, 所以 U1 产生 BYE request，其 Request URI 设为 U1 端对话状态的 remote target:

BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr>

对话建立后，U2 此时 UAC 根据 U2 的对话状态构造如下 BYE request（within dialog request）并发出（其 Request URI 设为 U2 端对话状态的 remote target），因为对话状态 route set 中的第一项是 loose router, 消息的下一跳目的地则是 request 中的 route head field 中的第一项，即 P4：

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 不负责 Request-URI 中指明的域。
route information preprocessing：
P4 它发现 route head field 中的第一项是它自己，它就将其从 route head field 中去掉。这时 route head field 中的第一项是 Route: <sip:p3.middle.com>
postprocessing routing information：
route head field 中的第一项是 Route: <sip:p3.middle.com>，其不含有 1r 这个参数（表明是 restrict router），代理服务器做如下操作：
首先，代理服务器 P4 将 request-URI 放到 route head field 中，作为它最后一项；
其次，代理服务器 P4 将 route head field 的第一项放到 request-URI 中去，并将它从 route head field 中去掉。

经过上述处理，P3 收到 P4 发来的消息如下：

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 是 strict router, 所以它将消息改写成:

BYE sip:p2.example.com;lr SIP/2.0
Route: <sip:p1.example.com;lr>
Route: <sip:caller@u1.example.com>

P3 将其发送给 P2。

P2 发现 request-URI 是它先前置于 Record-Route header field（route information preprocessing），所以，P2 将 Route head field 中的最后一项放到 request-URI 中去，然后将这个值从 route head field 中去掉，从而消息改写成：

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

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

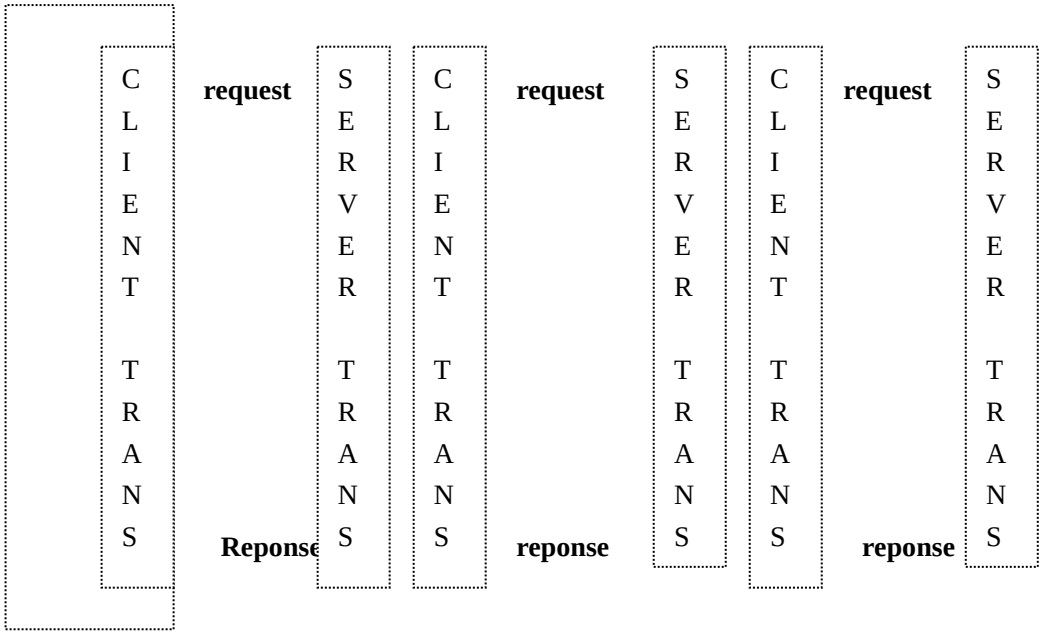
P2 不负责 u1.example.com 域，所以它并不重写 request-URI，P2 是 loose router，所以消息传递的下一跳是 P1。

P1 接收到了消息，它发现 router head field 中的第一项是它自己（route information preprocessing）。所以就将其从 route head field 中去掉，从而将 request 改写成

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

因为 P1 不负责 Request-URI 指向的 u1.example.com 域，所以它并不重写 request-URI，然后通过 DNS 查询负责 u1.example.com 域的 Proxy，并发送过去。

5．事务处理（Transaction）



每一个 stateful proxy 都有 client transaction 和 server transaction.
Client transaction 又有：INVITE client transaction, non-INVITE client transaction.
Server transaction 又有：INVITE server transaction, non-INVITE server transaction.

(1) INVITE client transaction(figure 1)

当 TU 发出 INVITE 时，TU 必须启动一个新的 client transaction,则就进入了“ calling ”状态。Client transaction 必须将 request 放到传输层以传输。如果传输层是不可靠传输，则 client transaction 必须启动 timer A，值为 T1；如果传输层是可靠传输，则一定不能启动 timer A。但是不管怎样，都必须启动 timer B，值为 64×T1。

如果 timer A 到时间了，就必须重发 request,而且 timer A 的值设为 2×T1，当 timer A 又到时间了，就必须再次重发 request,然后值设为前一次的两倍。这个过程不断重复，只要状态还处于“ calling”状态中。

如果 timer B 到时间了，但是状态还处于“ calling”状态中，client transaction 必须通知 TU 时间已经到了，client transaction 绝对不能产生 ACK 这个 request。

如果在“ calling”状态中收到了一个 provisional response，则状态从“ calling”转到了“ proceeding ”。在这个状态中，则不能再重发 request。而且，provisional response 必须给 TU，而且在“ proceeding”状态中，任何其他的 provisional response 也必须传给 TU。

无论是在“ calling ”状态还是在“ proceeding ”状态，只要收到 300－699 的 response，都会使状态进入“ completed ”。Client transaction 将收到的响应传给 TU。Client transaction 必须产生 ACK，即使传输层是可靠的。ACK 必须给送到传输层传输 request 的相同地址，相同端口。这时 client transaction 一进入“ completed ”状态就要启动 timer D.对于不可靠传输层，值至少是 32 秒；对于可靠传输层，值是 0。在“ completed ”状态中，每收到一个 final response，都会导致 ACK 的重传。但是新收到 response 不能被传给 TU.

如果 timer D 到时间了，则状态进入“ terminated ”状态。无论是在“ calling ”状态还是在“ proceeding ”状态，只要收到 2XX 响应，就会进入到“ terminated ”状态。响应必须传给 TU。对于这个响应的处理依赖于 TU 是 proxy core,还是 UAC core.后者会为这个响应产生 ACK,而前者只会将这个响应前传。

当进入到“ terminated ”状态时，这个 client transaction 就会被毁掉。这也是上述处理的原因。

（2）non-INVITE client transaction(figure 2)

当 request 到达时，TU 必须启动一个新的 client transaction,则就进入了“ trying ”状态。从 client transaction 设置 timer F，值是 64×T1。如果传输层是不可靠的，设置 timer E，值是 T1。如果 timer E 到时间了，但是还依旧处于“ trying ”状态中，则重新设置 timer E，值是 MIN(2*T1,T2),这样反复进行。

如果在“ trying ”状态中,timer F 时间到了，则 client transaction 通知 TU，而且进入到状态“ terminated ”中。

如果在“ trying ”状态中，收到了 provisional response，这个 response 必须传给 TU,client transaction 进入到状态“ proceeding ”。如果在“ trying ”状态中，收到了 final response（200－699），这个 response 必须传给 TU,client transaction 进入到状态“ completed ”。

注意，在“ proceeding ”状态，request 必须依旧重发，而且 timer E 被设置成 T2。如果 timer F 时间到了，则 client transaction 通知 TU，而且进入到状态“ terminated ”中。如果收到 final response（200－699），则状态进入到“ completed ”。

在“ completed ”状态，对于不可靠传输层，设置 timer K,值为 T4；对于可靠传输层，值为 0。这个状态只是为附加的 reponse 的重传而设置缓冲。如果 timer K 时间到了，则就进入到状态“ terminated ”。

(3) INVITE server transaction(figure 3)

当 request 到达时，TU 必须启动一个新的 server transaction,则就进入了“ proceeding ”状态。除非 server transaction 知道 TU 会在 200ms 内产生 provisional response 或者 final response，否则 server transaction 产生 100 response.这个 response 主要是为了阻止 request 的重发，以避免网络阻塞。只要 server transaction 在“ proceeding ”状态，则 TU 会传送任意多的 response 到 server transaction.当收到重传的 request,则 server transaction 就将它所收到的最近的 provisional response 放到传输层去重发。

如果处于“ proceeding ”状态，TU 将 2XXresponse 传给 server transaction，注意这里 2XX 的重传并不是有 server transaction 来控制，它必须由 TU 控制。这是 server transaction 进入到状态“ terminated ”。

在“ proceeding ”状态，TU 产生 300－699response，则状态进入“ completed ”。对于不可靠传输层，必须设置 timer G，值为 T1,如果是可靠传输层，就不必设置。

当进入“ completed ”状态，同时也要设置 timer H, 值为 64*T1。Timer G 到时间，则 response 要重传，同时设置新值。如果收到一个 request 的重传，则也必须重传 response.

在“ completed ”状态中时，如果收到 ACK，则状态就进入到“ comfirmed ”。在这里，timer G 被忽视，所有 response 的重传都被中止。如果 timer H 的时间到了，还一直没有收到 ACK，则状态进入到“ terminated ”，必须通知 TU transaction failure。

当进入“ comfirm ”状态，同时也要设置 timer I,值为 T4。如果时间到了，就进入到状态“ terminated ”。

(4) Non-INVITE server transaction(figure 4)

一旦收到非 INVITE,ACK request,状态首先进入到‘ trying ”,如果 TU 传送一个 provisional response,则 server transaction 进入到“ proceeding ”状态。任何从 TU 接收到的 provisional response 都必须送到传输层去传输。如果收到重传的 request,则最近的 response 进行重传。如果 TU 产生 final response（200－699），则状态进入到“ completed ”。而且 response 要送到传输层重传。

当 server transaction 进入到“ completed ”状态，必须设置 timer J，对于可靠传输层，值为 64×T1，对于不可靠传输层，值为 0。只要收到一个 request 的重传，都必须重传 final response.其它 response 都忽视。当 timer J 到时间了，则进入到“ terminated ”状态。

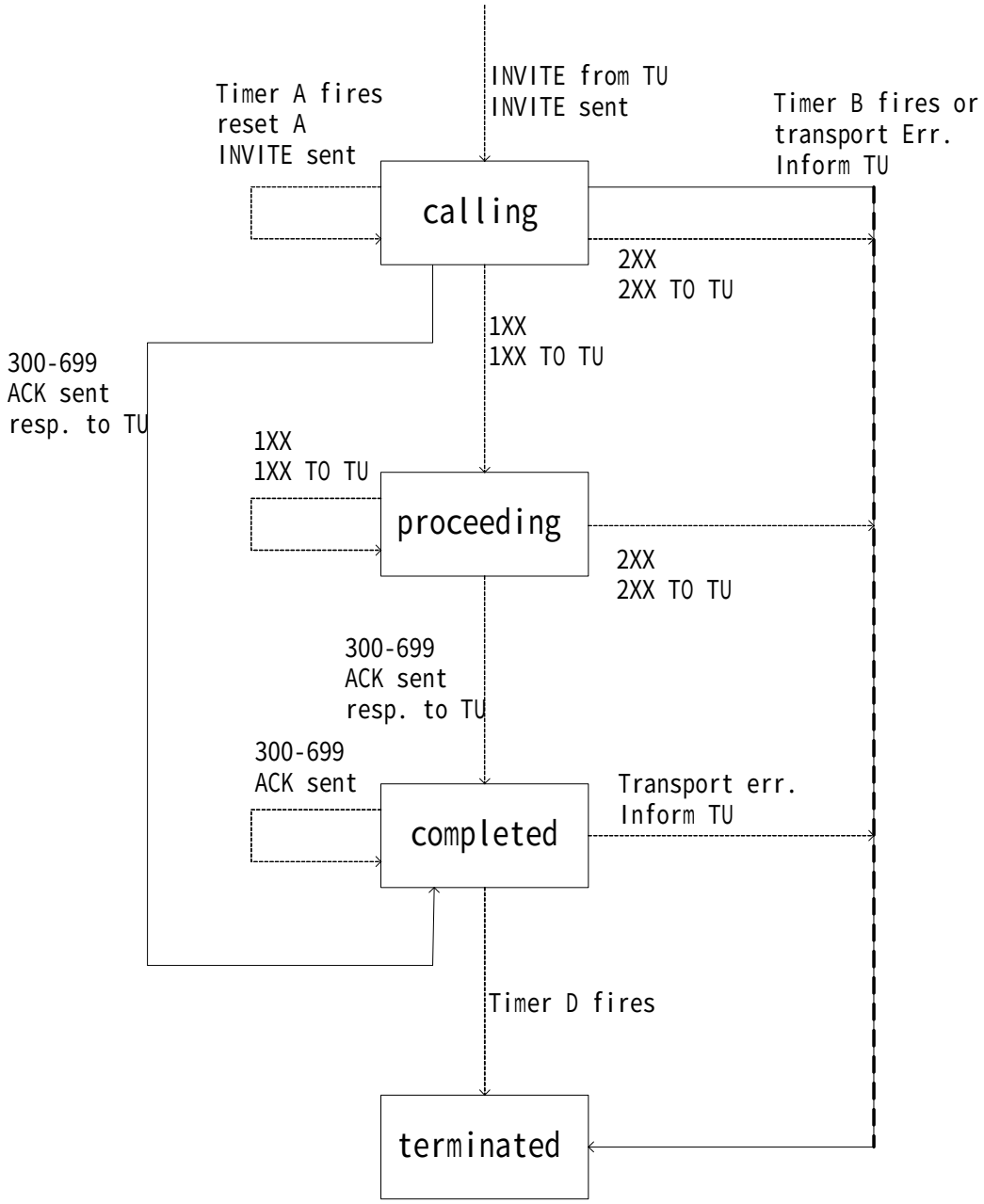


figure1 ：INVITE client transaction

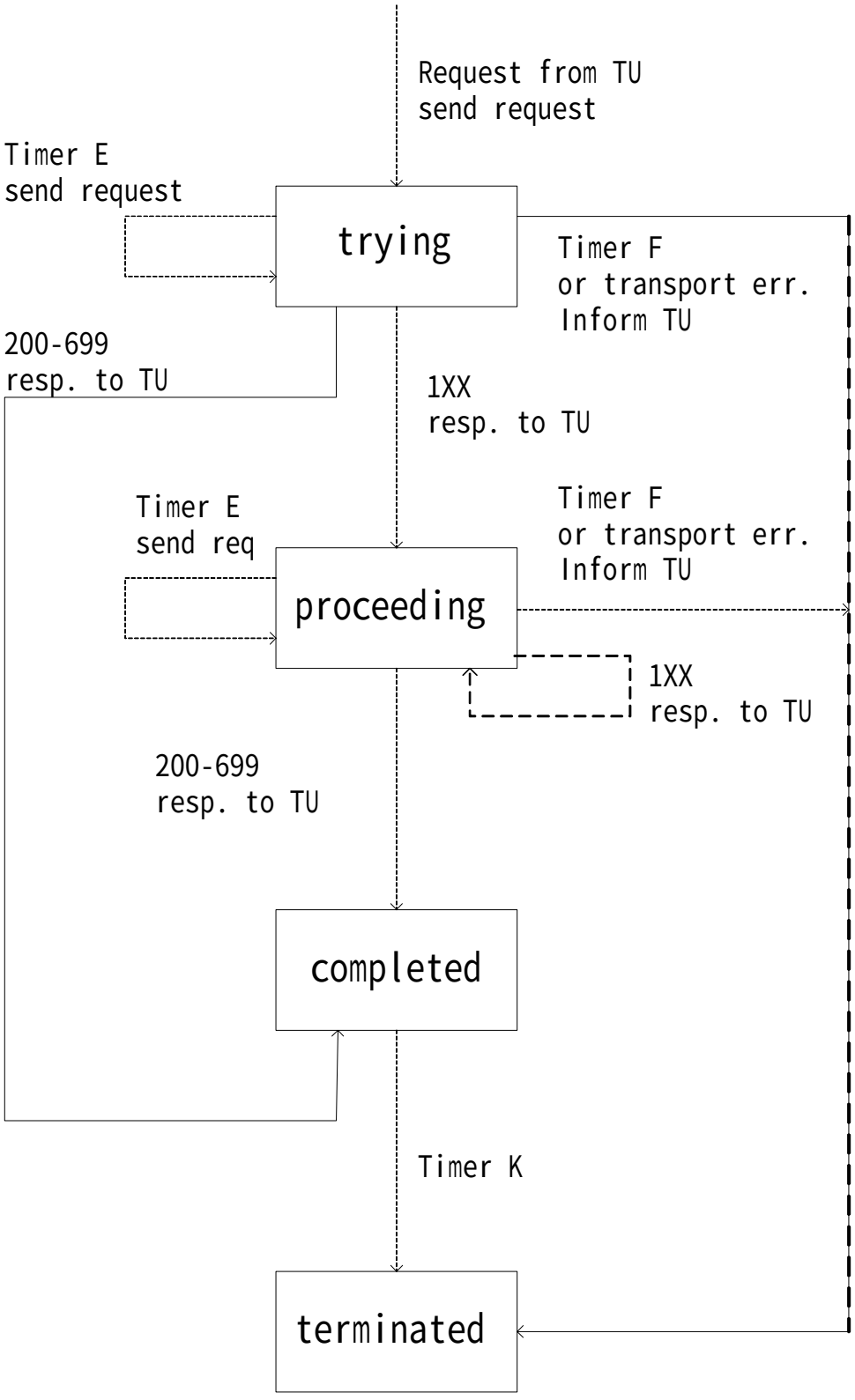


figure2 ：non-INVITE client transaction

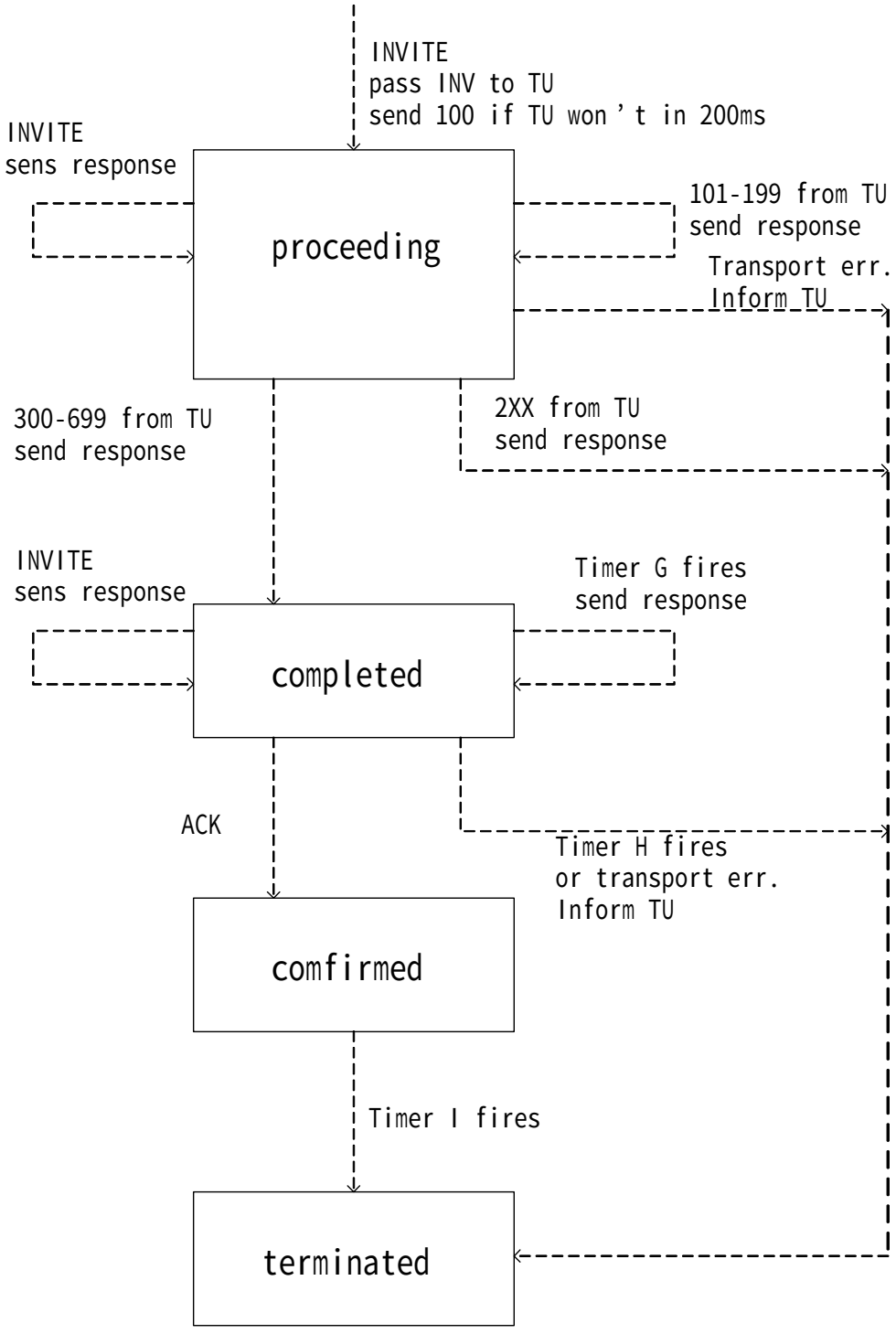


figure3 ： INVITE server transaction

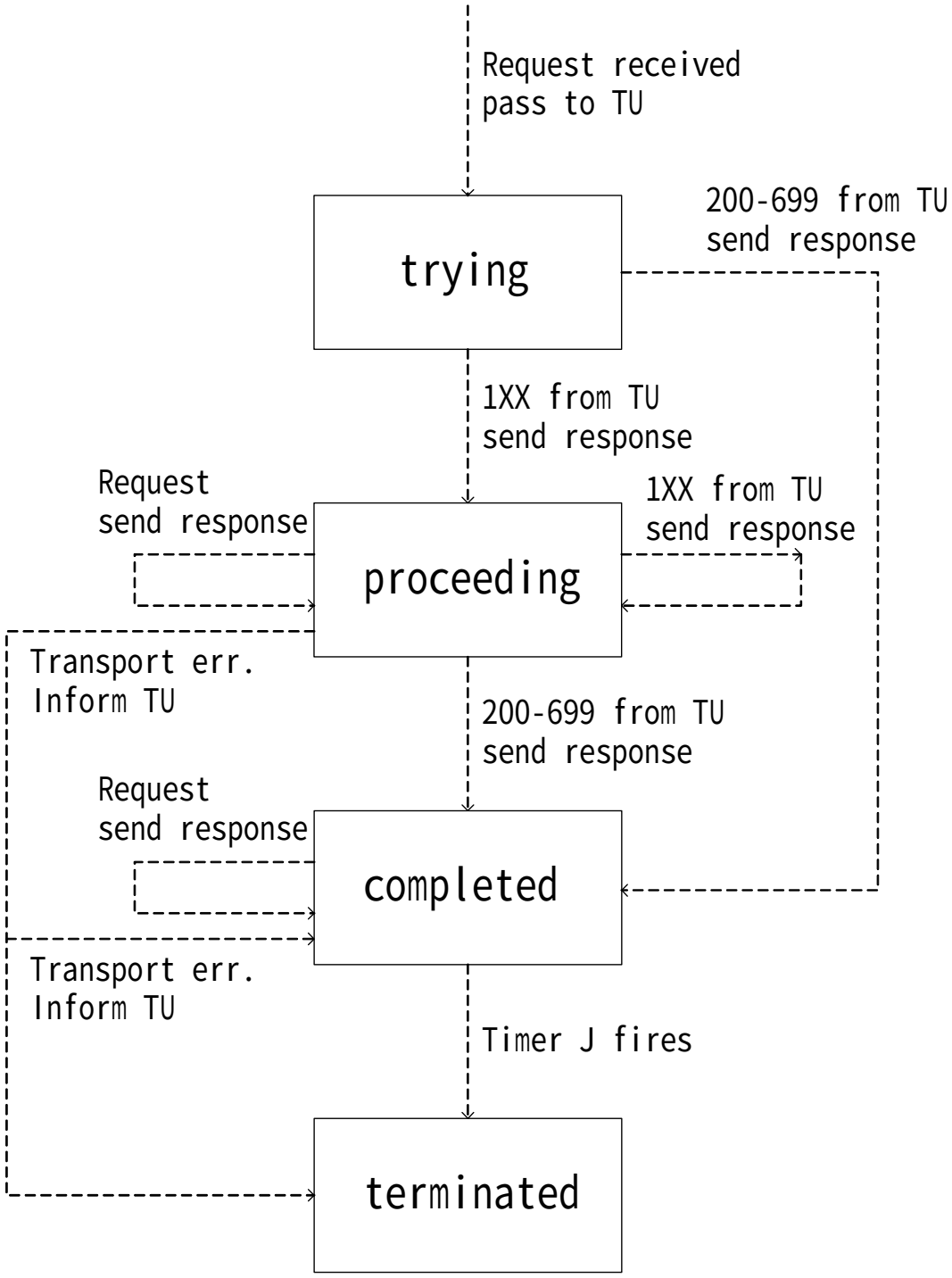


figure4 ： non-INVITE server transaction