TB640 SIP User's Guide

Document number
9010-00087-04

February 2006

This page in intentionally left blank

1 TABLE OF CONTENT

2 LIST OF FIGURES

3 LIST OF TABLES

This page in intentionally left blank

The information/code contained in this document/product is based on the best information we have available. Although it has been tested successfully with other piece of equipment, we cannot guarantee that it will conform to the usage of any particular switch in the field.

This page in intentionally left blank

4 TB640 SIP OVERVIEW

TelcoBridges SIP implementation works on top of few layers. In the following figure, grey boxes represent entities that need allocation on the TB640. The TUCL layer is transport layer used by SIP on our architecture. TUCL presents some advantages over a simple TCP/IP stack. For instance, it adds tracing facilities to any virtual interfaces.

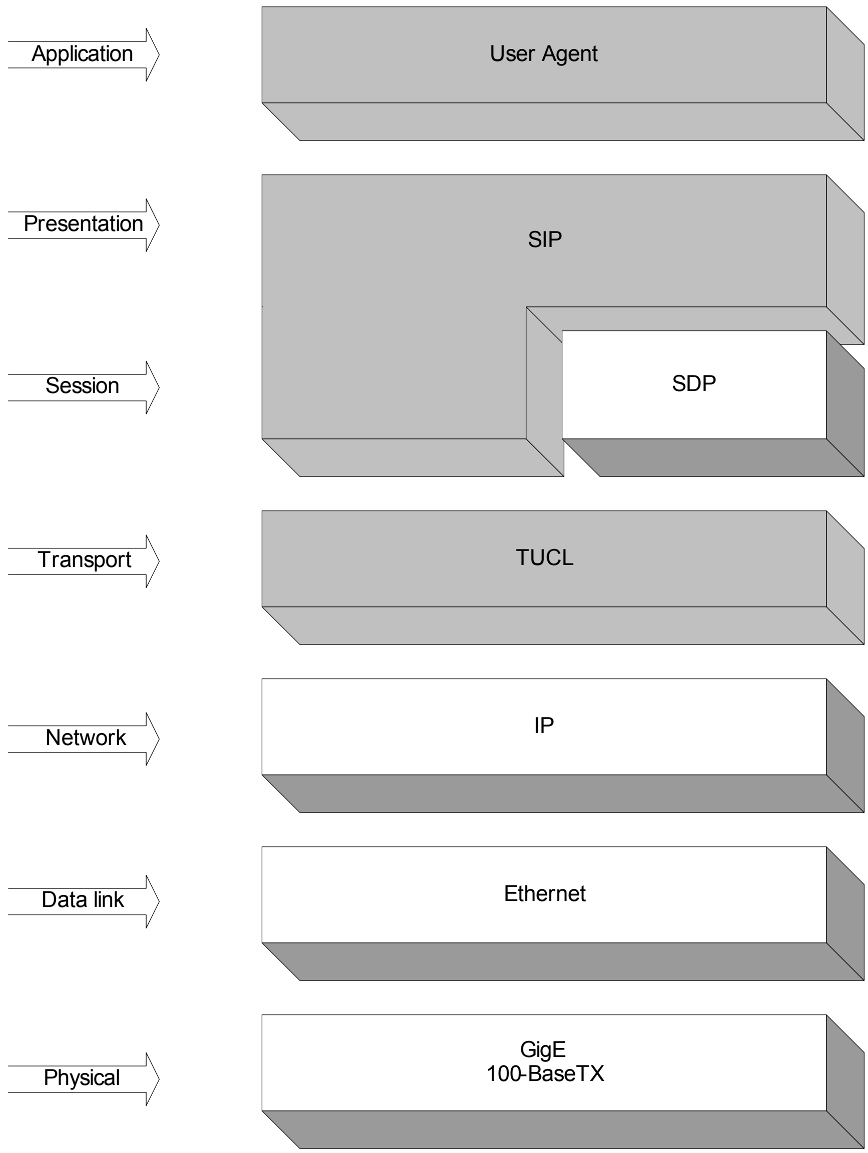


Figure 1 - TelcoBridges SIP Architecture

5 TUCL

5.1 Overview

This section gives a description of the TB640 Tucl architecture and usage.

5.1.1 Summary

TUCL stands for **T**cp/**U**dp **C**onvergence **L**ayer. TUCL offers TCP/IP and UDP/IP services to **TelcoB**ridges SIP stack. TUCL virtualizes TB640 physical Ethernet interfaces, making them globally accessible within the system. See feature list for other advantages TUCL offers.

5.1.2 Features

TUCL adds the following functionalities to SIP:

- Provides seamless access to any Ethernet interface on any TB640.
- Enables SIP to create Ethernet IP aliases (virtual IP).
- Tracing of packets to/from Ethernet interfaces.
- Eventually, will offer Ethernet redundancy.

5.1.3 Architecture

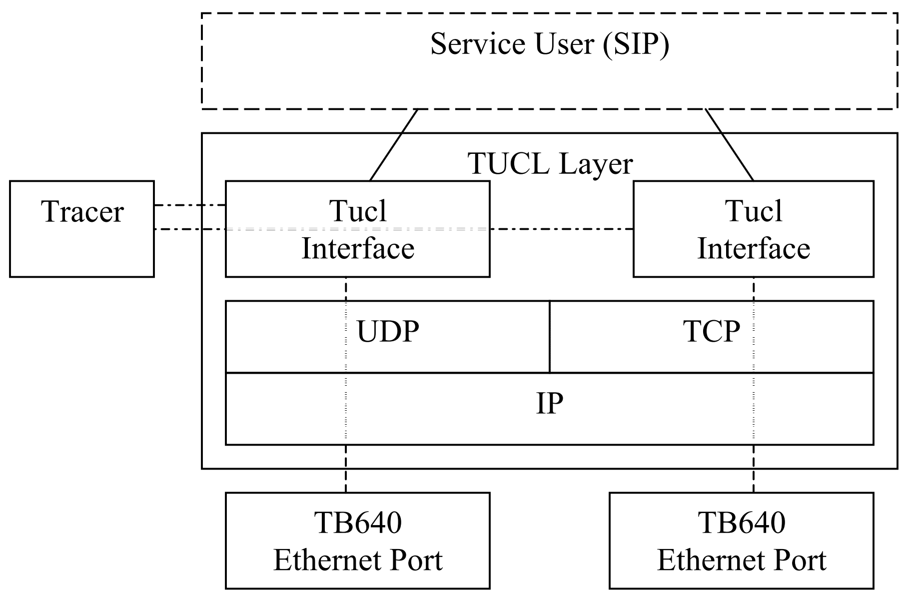


Figure 2 - TUCL Architecture

5.2 Stack Configuration

General guidelines for configuration of TUCL stack for a proper operation include the following:

1. The TUCL general allocation (TB640_MSG_TUCL_OP_ALLOC) must precede all other messages (other configuration TUCL alloc, get, states and stats). The response of this message is a TUCL handle.

2. The TUCL Interface allocation (TB640_MSG_TUCL_OP_INTERFACE_ALLOC) must be made (with the TUCL handle from **step 1**) to bind to an Ethernet port. The response of this message is a TUCL Interface handle.

✍ For any configuration with a TUCL (TB640_MSG_TUCL_OP_xyz_ALLOC) message, all configuration fields must be filled unless explicitly optional or not defined.

✍ For any reconfiguration with a TUCL (TB640_MSG_TUCL_OP_xyz_SET_PARAMS) message, all reconfigurable parameters must be filled appropriately even if the intention is to modify a single parameter.

✍ When a reconfiguration with a TUCL set params is issued, the effect of the set params may become effective immediately.

✍ For the configuration of many TUCL Interfaces (1 to maximum) you must repeated the **step 2**.

5.2.1 General Configuration

The TB640_MSG_TUCL_OP_ALLOC (request/response) message is used to initialize the general parameters of the TUCL stack.

The **request** part of the message TB640_MSG_ID_TUCL_OP_ALLOC contains the field:

```
…
TB640_SYSMGR_LAYER_HANDLE     hLayer; /* Contains the layer handle from system manager
                                         module */
…
```

The **response** part of the message TB640_MSG_ID_TUCL_OP_ALLOC contains the field:

```
…
TB640_TUCL_HANDLE             hTUCL;  /* The handle of the initialized TUCL layer */
…
```

5.2.2 Interface(s) Configuration

The TB640_MSG_TUCL_OP_INTERFACE_ALLOC (request/response) message is used to initialize the general parameters of the TUCL stack.

The **request** part of the message TB640_MSG_ID_TUCL_OP_INTERFACE_ALLOC contains the field:

```
…
TB640_TUCL_HANDLE             hTUCL;  /* Contains the TUCL layer handle */
TB640_TUCL_INTERFACE_CFG      Cfg;    /* Contains the configuration of the TUCL interface
                                         (\ref _TB640_TUCL_INTERFACE_CFG) */
…
```

The **response** part of the message TB640_MSG_ID_TUCL_OP_INTERFACE_ALLOC contains the field:

```
…
TB640_TUCL_INTERFACE_HANDLE   hTUCLIf; /* The handle of the TUCL interface */
…
```

Structure contains the stack configuration parameters for TUCL stack:

```
typedef struct _TB640_TUCL_INTERFACE_CFG
{
       TBX_UINT32              un32StructVersion;
       TB640_UNIQUE_ID         uidTUCLIf;
       TBX_CHAR                szInterfaceName[ TB640_TUCL_MAX_IF_NAME_LEN ];

} TB640_TUCL_INTERFACE_CFG, *PTB640_TUCL_INTERFACE_CFG;
```

General explanation of the parameters of interface configuration:

TUCL interface must be bound to a physical Ethernet port on the TB640 on which the TUCL stack runs. The Ethernet port's name must be correctly set in szInterfaceName from TUCL interface configuration structure. Allowed values depend on the hardware present on the TB640. Possible Ethernet port's names are: "eth0", "eth1", "voip0" and "voip1".

TUCL maintains a separate transmission queue for every connection to buffer any data received from the service user that has not been transmitted. When transmit queue congestion has started, a status indication is sent indicating start of congestion. If sending of flow control is enabled, a flow control indication is sent to the service user indicating start of flow control.

When the size of the data buffered in the transmit queue reaches an even higher limit, a status indication is sent indicating that severe congestion has been encountered. If the sending of flow control is enabled, a flow control indication is sent to the service user indicating that further data received from the service user is dropped.

When the size of the data buffered in the transmit queue falls enough, a status indication is sent indicating that congestion has ended. If the sending of flow control is enabled, a flow control indication is sent to the service user indicating the end of flow control.

An alarm is generated to the on the following transitions:

- Green to yellow zone with event as **TB640_TUCL_INFO_ALARM_EVENT_TXQ_CONG_START**.
- Yellow to red zone with event as **TB640_TUCL_INFO_ALARM_EVENT_TXQ_CONG_DROP**.
- Yellow to green zone with event as **TB640_TUCL_INFO_ALARM_EVENT_TXQ_CONG_STOP**.

It is recommended to use provided default values. See **TB640_TUCL_SET_DEFAULT_TUCL_INTERFACE_CFG** macro for default values.

5.3 Alarms

TUCL generate events on internal error conditions and on congestion zone crossing. See **TB640_TUCL_INFO_ALARM_EVENT** in tb640_TUCL.h for a complete list of alarms.

Table 1 - TUCL alarms

TUCL Alarm	Description
TB640_TUCL_INFO_ALARM_EVENT_RES_CONG_START	TUCL entered congestion conditions. Possible causes: • Heavy IP Load • Unknown
TB640_TUCL_INFO_ALARM_EVENT_RES_CONG_DROP	TUCL entered dropping data conditions because of heavy congestion. Possible causes: • Heavy IP Load • Unknown
TB640_TUCL_INFO_ALARM_EVENT_RES_CONG_STOP	TUCL leaving congestion conditions. Possible causes: • Heavy IP Load • Unknown
TB640_TUCL_INFO_ALARM_EVENT_TXQ_CONG_START	TUCL Interface entered congestion conditions. Possible causes: • Heavy IP Load • Unknown
TB640_TUCL_INFO_ALARM_EVENT_TXQ_CONG_DROP	TUCL Interface entered dropping data conditions because of heavy congestion. Possible causes: • Heavy IP Load • Unknown
TB640_TUCL_INFO_ALARM_EVENT_TXQ_CONG_STOP	TUCL Interface leaving congestion conditions. Possible causes: • Heavy IP Load • Unknown
TB640_TUCL_INFO_ALARM_EVENT_BIND_OP	Internal Error: Alarms generated while binding Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_UNBIND_OP	Internal Error: Alarms generated while unbinding Possible causes: • Bad Service User (SIP)

	Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_SERV_OPEN_REQ	Internal Error: Alarms generated while processing server opening request Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_CONN_REQ	Internal Error: Alarms generated while processing connection request Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_CONN_RSP	Internal Error: Alarms generated while processing connection response Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_DATA_REQ	Internal Error: Alarms generated while processing data tx request Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_UDATA_REQ	Internal Error: Alarms generated while processing udata tx request Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_DISC_REQ	Internal Error: Alarms generated while processing disconnect request Possible causes: • Bad Service User (SIP) Configuration • Unknown.
TB640_TUCL_INFO_ALARM_EVENT_INET	Internal Error: Identifies errors due to socket related operations Possible causes: • Bad Service User (SIP) Configuration • Unknown.

Internal error condition events should be monitored to help identify run-time problems as well as configuration problems.

5.4 States

TUCL uses Resource Congestion Thresholds to determine current resource threshold levels before reading data from the socket receive buffer. This is not related to transmission queue congestion. Three zones are defined:

1. Red Zone
2. Yellow Zone
3. Green Zone

Normal operations are permitted inside the green zone. Once the resource threshold reaches the yellow zone, requests for new servers, new clients, and any new incoming connections are rejected. However, data transfer on the existing connections is possible. On entering the red zone, no data transfer requests are allowed until the resource threshold enters the green zone again.

An alarm is generated on the following transitions:

- Green to yellow zone with event as **TB640_TUCL_INFO_ALARM_EVENT_RES_CONG_START**.
- Yellow to red zone with event as **TB640_TUCL_INFO_ALARM_EVENT_RES_CONG_DROP**.
- Yellow to green zone with event as **TB640_TUCL_INFO_ALARM_EVENT_RES_CONG_STOP**.

6 SIP

6.1 Overview

This section gives a description of the TB640 SIP architecture and usage.

6.1.1 Summary

SIP -- Session Initiation Protocol -- is an "Internet" application-layer protocol that runs in User Agent and Server Systems for controlling multimedia sessions between users, who may move from one location to another, and use terminal devices with various media capabilities. TB640 SIP runs on top of TUCL and uses the services provided by other Internet application-layer protocols such as DNS.

SIP is based on an HTTP-like request/response transaction model. Each transaction consists of a request that invokes a particular method, or function, on the server and at least one response. The TB640 SIP stack is used as a User Agent (UA) element.

6.1.2 Features

Here’s the list of the Sip features we’ll be supporting:

1. RFC3261: Session Initiation Protocol
   TB640 implementation complies fully with RFC3261. It supports the following methods from RFC 3261 :

   - INVITE

- ACK
- BYE
- CANCEL
- REGISTER
- OPTIONS
- All 1xx, 2xx, 3xx, 4xx, 5xx and 6xx responses are supported.
- DNS queries: The SIP layer supports DNS domain resolution as specified in the RFC3263.
- All SIP message retransmissions and protocol timers
- Transport of messages using TCP or UDP.
- Reliability mechanisms for SIP messages transported on UDP

2. Reliable provisional responses
   - RFC 3262: Reliability of Provisional Responses in SIP

   This extension ensures that provisional responses are delivered reliably, independent of the underlying transport mechanism. A new method, PRACK, is issued as a provisional ACK, to confirm the receipt of a reliable provisional response. Allows automatic sending of reliable response, PRACKs and response to PRACKs, including all the headers specific to these messages (RSeq & RAck). This feature can be turned off if not required.

3. IP access to telephone call services **(parsing capabilities)**
   - RFC 2848: PINT - IP Access to Telephone Call Services
   - RFC 2806: URLs for Telephone Calls
   - draft-yu-tel-url-07: Extension to Tel URL to support Number Portability and Freephone Service.

   The TB640 implementation allows a user to invoke certain telephone call services from the internet. The milestone services that are invoked on the telephone network are phone calls, requests for sending fax contents, and requests to speak, send, or play content. The generic scenario follows:

   1. An IP host sends a request to a server on an IP network.
   2. The server relays the request into a telephone network.
   3. The telephone network performs the requested call service.

   A user who wishes to invoke a service within the telephone network uses SIP to invite a remote PINT server into a session. The TB640 implementation supports encoding and decoding of URL tags that are used to specify telephone network parameters. The SDP information specifies the type of media. It is up to the service user to interpret the SDP information.

4. Call transfer
   - RFC 3515: The REFER Method
   - draft-ietf-sipping-cc-transfer-05: Call Transfer
   - draft-ietf-sip-referredby-00: The Referred-By Mechanism
   - draft-ietf-sip-replaces-03: The SIP Replaces Header

The TB640 implementation allows a user to send and receive REFER and NOTIFY messages. REFER is a method that can be used for call-transfer. A user can use the REFER and NOTIFY messages to initiate call transfer and detect the result of a call transfer.

5. Offer-Answer Model **(parsing capabilities)**

- RFC 3264: An Offer Answer Model with Session Description

The software provides support for basic Offer-Answer exchange as described in RFC 3261.

6. Real Time Facsimile Communication over IP Networks **(parsing capabilities)**

- ITU-T Recommendation T.38: Procedures for real-time Group 3 facsimile communication over IP networks.

TB640 implementation supports fax related parameters as a part of SDP content in the SIP messages.

7. 3GPP Support

- draft-ietf-sipping-3gpp-r5-requirements-00: 3rd Generation Partnership Project (3GPP) Release 5 requirements on the Session Initiation Protocol

SIP implementation supports a number of headers and features related to use of SIP in 3G Networks. The following features are supported:

a) Privacy & Authentication support

- RFC 3313: Private Session Initiation Protocol (SIP) Extensions for Media Authorization.
- RFC 3323: A Privacy mechanism for the Session Initiation Protocol
- RFC 3325: Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks.
- RFC 3329: Security Mechanism Agreement for the SIP

The SIP layer supports a number of headers that can be used by the user to fill privacy & authentication information in SIP messages. These headers are required for privacy features in the 3G domain.

b) Registration Path & Service Route

- RFC 3327: Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts
- RFC 3608: Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration

The SIP implementation also supports the insertion of the Service Route & Path headers in Register messages. These headers carry route information to be used for future registrations.

8. Support for Reason Header

- RFC 3326: The Reason Header for the Session Initiation Protocol

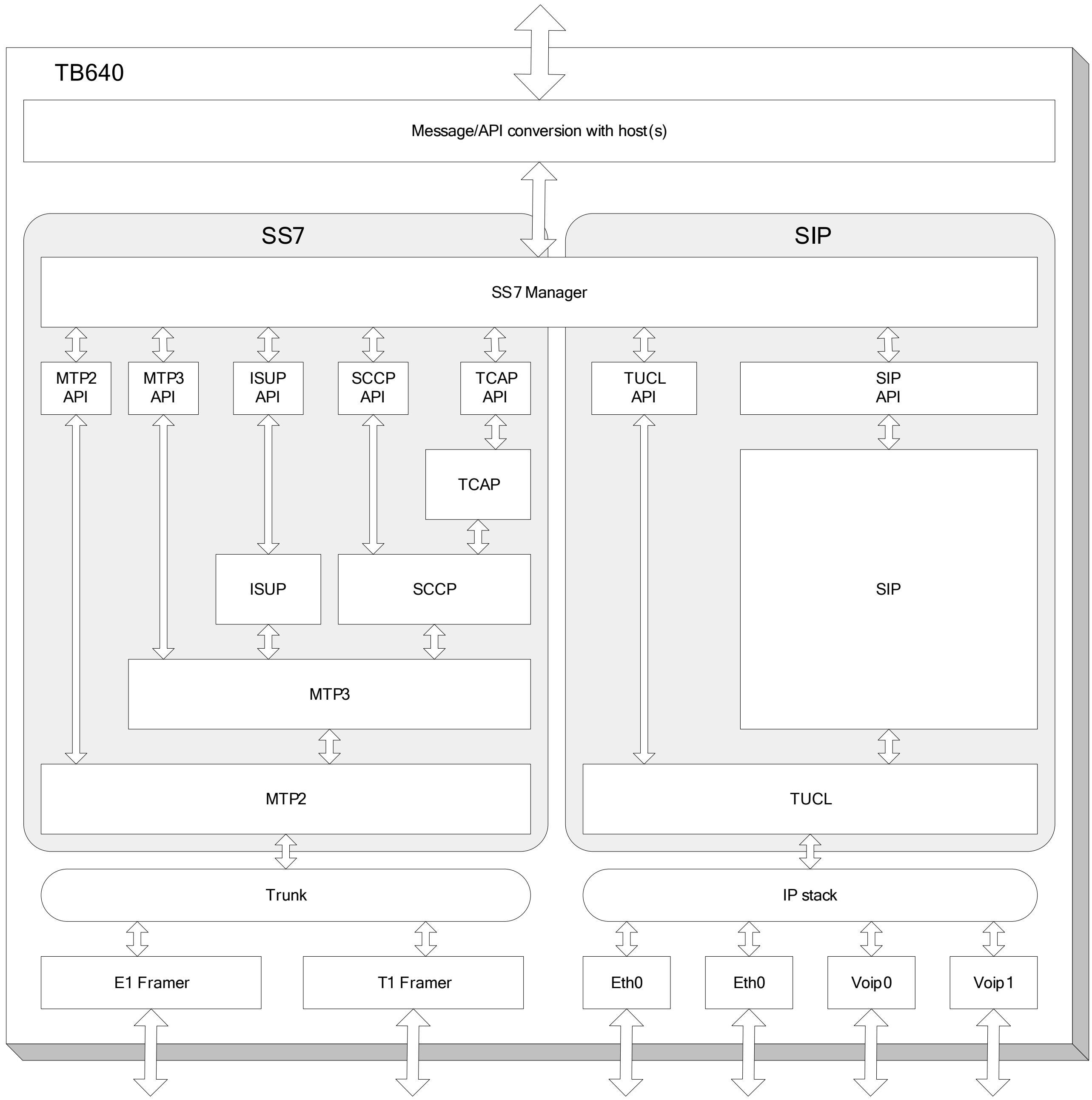
The SIP request can be issued for a variety of reasons. For example, a SIP CANCEL request can be issued if the call has completed on another branch or was abandoned before answer. While the protocol and system behavior is the same in both cases the user interface may well differ. In the second case, the call may be logged as a missed call, while this would not be appropriate if the call was picked up elsewhere. The Reason RFC provides a new header that can be used in requests to provide reason for generation of that request. SIP supports the use of this header. The layer doesn't fill the reason header, but it can successfully parse the header, if provided by the user or the remote end.

6.1.3 Specification

The SIP software supports the following standards:

1. RFC 3261: Session Initiation Protocol
2. RFC 3262: Reliability of Provisional Responses in SIP
3. RFC 3263: SIP: Locating SIP Servers
4. RFC 3264: An Offer Answer Model with Session Description
5. RFC 3313: Private Session Initiation Protocol (SIP) Extensions for Media Authorization.
6. RFC 3323: A Privacy mechanism for the Session Initiation Protocol
7. RFC 3325: Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks.
8. RFC 3326: The Reason Header for the Session Initiation Protocol
9. RFC 3327: Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts
10. RFC 3515: The REFER Method
11. RFC 3608: Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration
12. RFC 2806: URLs for Telephone Calls
13. RFC 2848: PINT - IP Access to Telephone Call Services
14. draft-ietf-sipping-cc-transfer-05: Call Transfer
15. draft-ietf-sip-referredby-00: The Referred-By Mechanism
16. draft-ietf-sip-replaces-03: The SIP Replaces Header
17. draft-ietf-sipping-3gpp-r5-requirements-00: 3rd Generation Partnership Project (3GPP) Release 5 requirements on the Session Initiation Protocol
18. draft-ietf-sip-nat-02: An Extension to the Session Initiation Protocol for Symmetric Response Routing
19. draft-ietf-sipping-mwi-02.txt: A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol.
20. draft-ietf-simple-presence-10: SIP Extensions for Presence.
21. draft-ietf-simple-im-sdp-00: SIP-SDP Extensions for SIP Instant Message Sessions
22. ITU-T Recommendation T.38: Procedures for real-time Group 3 facsimile communication over IP networks

6.1.4 Architecture within the TB640



6.2 Configuration

6.2.1 Stack Configuration

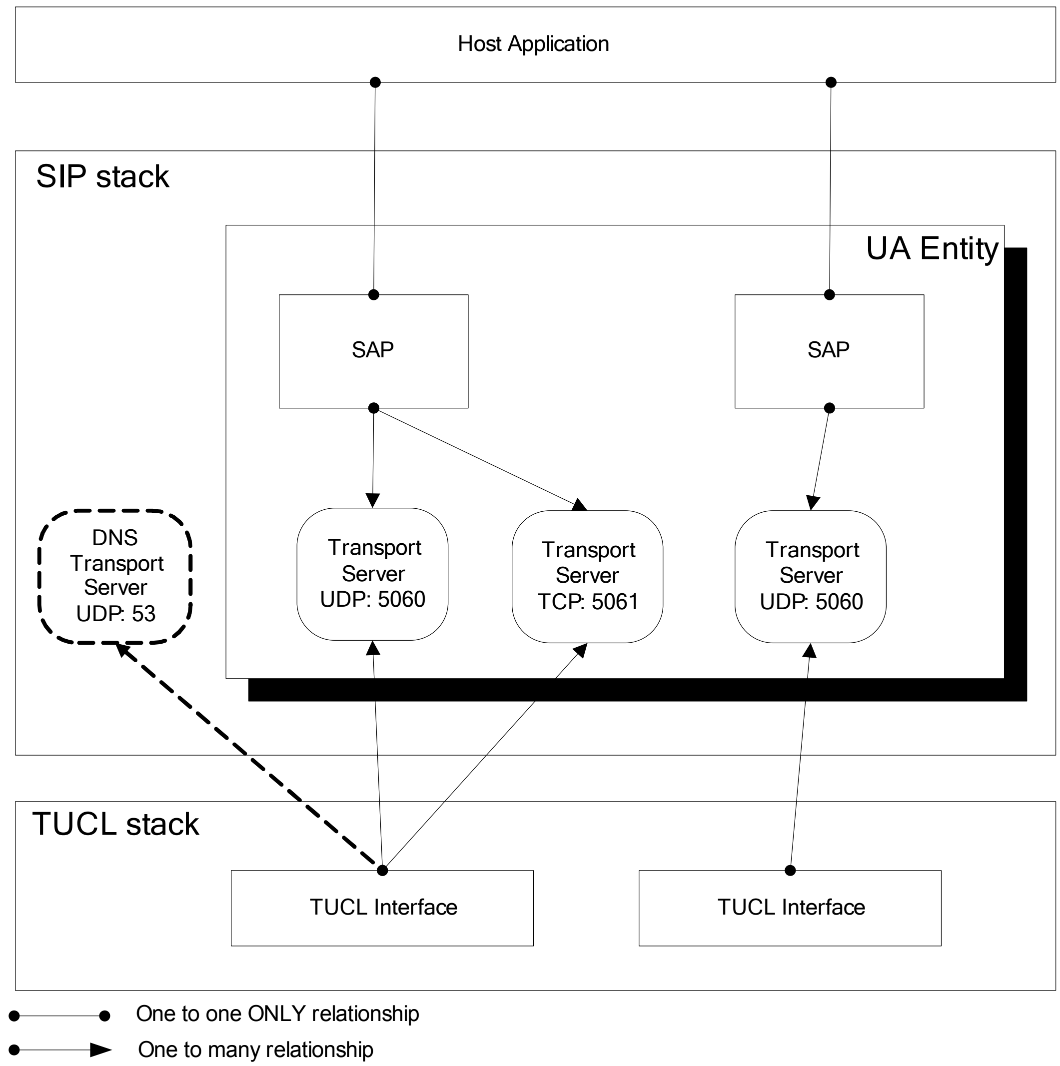


Figure 3 - SIP stack elements relationship

General guidelines for configuration of SIP stack for a proper operation include the following:

1. The SIP general allocation (TB640_MSG_ID_SIP_OP_ALLOC) must precede all other messages (other configuration SIP alloc, get, states and stats). The response of this message is a SIP handle.

2. The SIP User Agent (UA) entity allocation (TB640_MSG_ID_SIP_OP_UA_ALLOC) must be made (with the SIP handle from **step 1**). The response of this message is a SIP entity handle.

3. The SIP Service Access Point (SAP) allocation (TB640_MSG_ID_SIP_OP_SAP_ALLOC) must be made (with the SIP entity handle from **step 2**). The response of this message is a SIP SAP handle.

✍ For any configuration with a SIP (TB640_MSG_ID_SIP _OP_xyz_ALLOC) message, all configuration fields must be filled unless explicitly optional or not defined.

✍ For any reconfiguration with a SIP (TB640_MSG_ID_SIP_OP_xyz_SET_PARAMS) message, all reconfigurable parameters must be filled appropriately even if the intention is to modify a single parameter.

✍ The **step 1** can be done once per adapter.

✍ The **step 2** can be done 1 to maximum of Entity.

✍ The **step 3** can be done 1 to maximum of SAP.

6.2.1.1 General configuration

The TB640_MSG_ID_SIP_OP_ALLOC (request/response) message is used to initialize the general parameters of the SIP stack. These parameters are common to every other SIP entity instance created afterward.

The **request** part of the message TB640_MSG_ID_SIP_OP_ALLOC contains the field:

```
…
TB640_SYSMGR_LAYER_HANDLE     hLayer; /* Contains the layer handle from system manager
                                         module */
TB640_SIP_CFG                 Cfg;    /* Contains the configuration of the SIP layer */
…
```

This structure contains the general configuration parameters for SIP stack:

```
typedef struct _TB640_SIP_CFG
{
        TBX_UINT32                      un32StructVersion;
        TBX_UINT32                      un32ReqRetransmitT1;
        TBX_UINT32                      un32ReqRetransmitT2;
        TBX_UINT32                      un32ReqRetransmitT4;
        TBX_UINT32                      un32HostTimeSince1970;
        TBX_UINT8                       aun8Padding0 [4]; /* Align struct on 64 bits */
        TB640_SIP_DNS_CFG               DnsCfg;

} TB640_SIP_CFG, *PTB640_SIP_CFG;
```

✍ All parameters specifically noted with * are reconfigurable.

un32StructVersion: Version of the structure. Should be set to 1.

un32ReqRetransmitT1*: This is the time-out value (specified in timer units) to control retransmitting requests/responses. Timer units is 100 ms (1 timer unit = 100 ms). Recommended value: 5 (equivalent of 500ms).

un32ReqRetransmitT2*: This is the upper limit on the exponential back-off for the request/response retransmission timer (specified in timer units). All subsequent retransmissions assume this cap value. Timer units is 100 ms (1 timer unit = 100 ms). Recommended value: 40 (equivalent of 4s).

un32ReqRetransmitT4*: This is the upper limit to clear the transaction for unreliable transport mechanism. Timer units is 100 ms (1 timer unit = 100 ms). Recommended value: 50 (equivalent of 5s).

un32HostTimeSince1970 *: Host time to synchronize the TB SIP stack. Arithmetic time (time elapsed since Jan 1 1970 00:00:00). If not used, it must be set to 0. It is **important** to note that there might be delay between the time the message is sent and when this setting is set due to loading time. To minimize this delay it is recommended to issue a TB640_MSG_ID_SIP_OP_SET_PARAM after the allocation is done (since no loading is involved for that operation, the delay will be trimmed down to only network delay). To prevent time slipping it is also recommended to issue TB640_MSG_ID_SIP_OP_SET_PARAM periodically.

DnsCfg*: Domain Name Server configuration (TB640_SIP_DNS_CFG).

```
typedef struct _TB640_SIP_DNS_CFG
{
       TBX_BOOLEAN                    fEnable;
       TBX_BOOLEAN                    fUseCache;
       TBX_UINT8                      aun8Padding0 [2];
       TBX_UINT32                     un32MaxCacheExp;
       TBX_UINT16                     un16QueryTimer;
       TBX_UINT8                      un8MaxQueryRetry;
       TBX_UINT8                      aun8Padding1 [5]; /* Align struct on 64 bits */
       TB640_SIP_TRANSPORT_ADDRESS    DnsTransportAddr;
       TB640_SIP_TRANSPORT_SERVER     TransportServer;

} TB640_SIP_DNS_CFG, *PTB640_SIP_DNS_CFG;
```

fEnable: Enable the DNS capability. If TBX_FALSE, all the parameter of this structure are ignored.

un32MaxCacheExp: The maximum expiry time to be used for a DNS cache record[1]. Recommended value is 4320000 (equivalent of 12 hours).

un16QueryTimer: The timer used for DNS queries[2]. Recommended value is 200 (equivalent of 20 seconds). If a response to a DNS query is not received within this time, the query is retransmitted until a maximum of *un8MaxQueryRetry* times.

un8MaxQueryRetry: The maximum number of times to retry a DNS query before the query can be considered failed. Recommended value is 3.

DnsTransportAddr: DNS server transport address (TB640_SIP_TRANSPORT_ADDRESS).

[1] Time unit is expressed in 100 ms time step (1 timer unit = 100ms).
[2] Same as note 1.

TransportServer: Transport Server definition (TB640_SIP_TRANSPORT_SERVER). Refer to section 6.2.1.4 Transport Server configuration.

The **response** part of the message TB640_MSG_ID_SIP_OP_ALLOC contains the field:

```
…
TBX_RESULT                    Result;
TB640_SIP_HANDLE              hSip;
…
```

6.2.1.2 UA entity configuration

The TB640_MSG_ID_SIP_OP_UA_ALLOC (request/response) message is used to initialize the parameters of a SIP User Agent entity. These parameters are common to every other SIP SAP created afterward into this entity.

The **request** part of the message TB640_MSG_ID_SIP_OP_UA_ALLOC contains the field:

```
…

TB640_SIP_HANDLE              hSip;   /* Contains the SIP handle */
TB640_SIP_USER_AGENT_CFG      Cfg;    /* Contains the configuration of the SIP UA entity */
…
```

This structure contains the general configuration parameters for SIP User Agent entity:

```
typedef struct _TB640_SIP_USER_AGENT_CFG
{
       TBX_UINT32                           un32StructVersion;
       TBX_UINT8                            aun8Padding0 [4]; /* Align struct on 64 bits */
       /* Common configuration */
       TB640_SIP_COMMON_ENTITY_CFG          EntityCfg;
       /* UA configuration */
       TBX_UINT32                           un32DefaultRegisterExpires;
       TBX_UINT32                           un32DefaultInviteExpires;
       TB640_SIP_REGISTRY_REFRESH_MODE      RegRefreshMode;
       TBX_BOOLEAN                          fAccountingIndication;
       TBX_BOOLEAN                          fInsertTimestampHeader;
       TBX_BOOLEAN                          fUseIpInContactHeader;
       TBX_BOOLEAN                          fRequireReliableProvResp;
       TBX_BOOLEAN                          fSupportReliableProvResp;
       TBX_UINT8                            aun8Padding1 [5]; /* Align struct on 64 bits */
       TB640_SIP_PROXY_CFG                  DefaultProxyCfg;
       TB640_SIP_REGISTRAR_CFG              RegistrarCfg;

} TB640_SIP_USER_AGENT_CFG, *PTB640_SIP_USER_AGENT_CFG;
```

✍ All parameters specifically noted with * are reconfigurable.

un32StructVersion: Version of the structure. Should be set to 1.

EntityCfg: SIP common entity configuration (TB640_SIP_COMMON_ENTITY_CFG).

```
typedef struct _TB640_SIP_COMMON_ENTITY_CFG
{
       /* General */
       TBX_CHAR                      szDomainName[TB640_SIP_MAX_HOSTNAME_SIZE];
       TBX_BOOLEAN                   fAddRportByDefault;
       TBX_BOOLEAN                   fAlwaysSend100;
       TBX_BOOLEAN                   fUseCompact;
       TBX_BOOLEAN                   fAllowRecurse;
       TBX_BOOLEAN                   fDecodeSdp;
       TBX_BOOLEAN                   fDecodeHeader;
       TBX_UINT8                     aun8Padding0 [2]; /* Align struct on 64 bits */
       TBX_UINT32                    un32TransportConnTimer;
```



```
        /* Header manipulation configuration */
        TBX_BOOLEAN                     fInsertDate;
        TBX_BOOLEAN                     fInsertAllow;
        TBX_BOOLEAN                     fInsertExpires;
        TBX_BOOLEAN                     fInsertSupported;
        TBX_BOOLEAN                     fInsertAccept;
        TBX_UINT8                       un8MaxForward;
        TBX_UINT8                       aun8Padding1 [2]; /* Align struct on 64 bits */
        TBX_CHAR                        szOrganization[TB640_SIP_MAX_STR_SIZE];
        TBX_CHAR                        szSubject[TB640_SIP_MAX_STR_SIZE];

} TB640_SIP_COMMON_ENTITY_CFG, *PTB640_SIP_COMMON_ENTITY_CFG;
```

szDomainName: Specifies the domain name of the SIP entity.

fAddRportByDefault: Specifies whether to add the Rport parameter to the via header. If TBX_TRUE, the Rport parameter is added to the via header field of all outgoing messages.

fAlwaysSend100*: Specifies whether SIP stack entity should automatically generate 100 trying response when receiving an INVITE.

✍ This parameter only applies to a UA.

fUseCompact*: This object specifies whether the SIP entity uses the compact encoding form in the messages it sends. Allowable values: TBX_TRUE implies all outgoing messages are encoded using the compact encoding form. TBX_FALSE implies full encoding is used on requests and the encoding form used on responses is the same as for the corresponding incoming request.

fAllowRecurse*: This specifies whether SIP automatically tries addresses returned in contact header(s) in a 3xx redirect response. If TBX_FALSE, no recursive searches are performed. The 3xx response is sent upstream if the entity is a NS. If the entity is a UA, the message is sent up to call control.

fDecodeSdp*: Specifies whether SIP entity should decode the SDP body information into decoded structures before sending the message up to the user. If set to TBX_FALSE, SDP information in a SIP message is conveyed to the user as an opaque buffer.

fDecodeHeader*: Specifies whether SIP entity should decode the Header information into decoded structures before sending the message up to the user. If set to TBX_FALSE, header information in a SIP message is conveyed to the user as an opaque buffer.

un32TransportConnTimer*: Specifies the time for which a transport server connection (TCP or virtual UDP) is maintained before it is brought down. Timer units is 100 ms (1 timer unit = 100 ms).

fInsertDate*: Specifies whether to insert the date general-header field into invite, ack, bye, cancel, option, register, subscribe, notify, message, and refer requests and responses.

fInsertAllow*: Specifies whether the Allow header field is to be inserted into an outgoing response. The Allow header field contains the methods supported by this entity. In all cases where the entity has to generate a reply containing the capabilities and it is not mandatory to include the Allow header field (such as the cases designated by "should" or "may" in the RFC3261), the Allow header field is inserted into the response only if TBX_TRUE. Thus, in the case where it is mandatory to insert the Allow header field (such as in a 405 method not allowed response), the Allow header is always inserted by the SIP entity. In the case of a 200 response to the initial invite to a call, options responses and the final responses to all other methods, the Allow header field is only inserted if TBX_TRUE.

fInsertExpires*: Specifies whether to insert the expires header field into invite requests.

fInsertSupported*: Specifies whether the supported general-header field is to be inserted into outgoing requests. The supported general-header field enumerates the capabilities of this entity. This header field is not mandatory ("should" instead of "must" is used in the RFC3261); therefore, this configuration parameter is used to control its insertion into the methods. If set to TBX_TRUE, the supported header is inserted in all requests (except acknowledgement) and in all responses. Session timers functionality can be supported only if this parameter is set to TBX_TRUE.

fInsertAccept*: Specifies whether to insert the Accept header field into invite requests.
un8MaxForward: Specifies the value to insert into the max-forwards general header field. Max-forwards controls how many hops a request is allowed to endure before a 483 response is generated. If zero, then SIP does not insert max-forwards header fields into outgoing requests. The max-forwards header field can be used with register, options, info, cancel, invite, acknowledgement, bye, subscribe, notify, message, and refer requests.

Organization*: Specifies the organization general header field to insert in outgoing requests. If the entity is an outbound proxy, then this value is added to all outgoing requests. Organization general-header can be inserted in invite, register, options, subscribe, notify, message, and refer requests. The string must be null terminated '\0'. If the first character is a null character, it is considered not present.

Subject*: Specifies the subject general header field to insert in outgoing requests. The string must be null terminated '\0'. If the first character is a null character, it is considered not present.

un32DefaultRegisterExpires*: Default registration expiry for a contact at the UA (in seconds). When a user registers a contact without specifying an expiry time, then this value is inserted into the expires header in an outgoing register request to the registrar. Suggested default value: 3600 seconds. If this value is zero, then no expires header is filled in an outgoing message. If the response to the registration does not contain an expires header and *un32DefaultRegisterExpires* is zero, then a default timer is started for one hour.

un32DefaultInviteExpires*: Default expiry time of an outgoing invite request (in seconds). If a user sends an invite request at this end-point without specifying an expiry time, then an expires header containing this value is inserted into the outgoing invite request. The default value is 120 seconds. A value of zero implies infinite time, and this parameter is ignored.

RegRefreshMode*: Possible Registry refresh mode upon registration expiry (TB640_SIP_REGISTRY_REFRESH_MODE). When the SIP layer receives a response for the registration request sent out for a contact, it checks *RegRefreshMode* is set to either TB640_SIP_REGISTRY_REFRESH_MANUAL or TB640_SIP_REGISTRY_REFRESH_AUTO. If yes, it starts a registration refresh timer based on the expires information in the response. When this timer expires, if the manual refresh mode is set, it issues an indication to the user indicating expiry of the timer. The user then needs to refresh the contact information by sending out another registration. However, if the auto refresh mode is set, on expiry of the timer, SIP automatically generates a refresh for the registration. In such cases, the user is not informed. If the none refresh mode is set, then no timer is started for registration and no information about the registrations is maintained in the SIP module.

fCheckLocalUsrReg*: This parameter specifies whether unresolved requests would be sent up to call control. An unresolved call is one for which there is no registered local user. For the UA to operate as a Gateway server, this value must be set to TBX_TRUE.

fInsertTimestampHeader*: Specifies the time-stamp general-header field is to be inserted into outgoing requests. This information issued to determine round-trip times. If TBX_TRUE, the time-stamp can be inserted into ACK, CANCEL, INFO, OPTIONS, REGISTER, INVITE, BYE, SUBSCRIBE, NOTIFY, MESSAGE, and REFER requests.

fUseIpInContactHeader*: This parameter specifies whether the user agent should use IP addresses or hostnames in the contact header field of requests. The use of IP addresses in contact headers is useful for NAT traversal. This is only applicable where hostname values are available for the user agent to insert.

fRequireReliableProvResp*: This parameter specifies whether the reliability of provisional responses (PRACK) is required. If this flag is TBX_TRUE, the UAC demands that either the UAS send non-100 provisional responses reliably, or the UAS reject the initial request with a 420 response code. Thus, this flag should only be used if the user wants to make sure that all non 100 provisional replies received by this UAC are reliable.

fSupportReliableProvResp*: This parameter specifies whether the reliability of provisional responses (PRACK) is supported.

DefaultProxyCfg*: SIP proxy configuration (TB640_SIP_PROXY_CFG).

```
typedef struct _TB640_SIP_PROXY_CFG
{
        TBX_BOOLEAN                         fUseOutboundProxy;
        TBX_BOOLEAN                         fRcvFromInboundProxyOnly;
        TBX_BOOLEAN                         fLooseRouter;
        TBX_BOOLEAN                         fSigCompSupp;
        TB640_SIP_PROXY_LOCATION_TYPE       LocationType;
        union
        {
                TBX_CHAR                    szDomainName[TB640_SIP_MAX_HOSTNAME_SIZE];
                TB640_SIP_TRANSPORT_ADDRESS Addr;
        };
```

```
} TB640_SIP_PROXY_CFG, *PTB640_SIP_PROXY_CFG;
```

fUseOutboundProxy: Specifies whether all outgoing messages need to be sent to a default outbound proxy server.

fRcvFromInboundProxyOnly: Specifies whether to only receive messages from a default inbound proxy server.

fLooseRouter: Specifies whether Proxy is loose router(compliant to loose routing mechanism defined in RFC3261) or not.

fSigCompSupp: Specifies whether Proxy support signaling compression or not. If supported, the stack add "sigComp" parameter to the URL.

LocationType: Specifies whether the default proxy is specified as IP address or as domain name (TB640_SIP_PROXY_LOCATION_TYPE).

szDomainName: Proxy domain name if *LocationType* is set to TB640_SIP_PROXY_ LOCATION_DOMAIN_NAME.

Addr: Transport address (TB640_SIP_TRANSPORT_ADDRESS), if *LocationType* is set to TB640_SIP_PROXY_LOCATION_TRANSPORT_ADD.

RegistrarCfg*: Registrar configuration (TB640_SIP_REGISTRAR_CFG).

```
typedef struct _TB640_SIP_REGISTRAR_CFG
{
        TBX_BOOLEAN                     fUseRegistrar;
        TB640_SIP_TRANSPORT_ADDRESS     RegistrarServerAdd;

} TB640_SIP_REGISTRAR_CFG, *PTB640_SIP_REGISTRAR_CFG;
```

fUseRegistrar: Specifies whether Registrar server is used.

RegistrarServerAdd: Transport address of the registrar server (TB640_SIP_TRANSPORT_ ADDRESS).

The **response** part of the message TB640_MSG_ID_SIP_OP_UA_ALLOC contains the field:

```
…
TBX_RESULT                  Result;
TB640_SIP_ENTITY_HANDLE     hSipUa;
…
```

6.2.1.3 SAP configuration

The TB640_MSG_ID_SIP_OP_SAP_ALLOC (request/response) message is used to initialize the Service Access Point parameters of a SIP stack entity.

The **request** part of the message TB640_MSG_ID_SIP_OP_SAP_ALLOC contains the field:

```
…

TB640_SIP_HANDLE            hSip;   /* Contains the SIP handle */
```



```
	TB640_SIP_USER_AGENT_CFG	Cfg;	/* Contains the configuration of the SIP UA entity */
	…
```

This structure contains the configuration parameters for SIP SAP:

```
typedef struct _TB640_SIP_SAP_CFG
{
	TBX_UINT32			un32StructVersion;
	TBX_UINT32			un32NbTransportServer;
	TB640_SIP_TRANSPORT_SERVER	aTransportServer[TB640_SIP_MAX_NB_TRANSPORT_SERVER];

} TB640_SIP_SAP_CFG, *PTB640_SIP_SAP_CFG;
```

✍ All parameters specifically noted with * are reconfigurable.

un32StructVersion: Version of the structure. Should be set to 1.

un8NbTransportServer: Number of transport server configured in *aTransportServer*.

aTransportServer: Array of SIP Transport Server (TB640_SIP_TRANSPORT_SERVER). The maximum number of element in the list is TB640_SIP_MAX_NB_TRANSPORT_SERVER. Refer to section 6.2.1.4 Transport Server configuration.

The **response** part of the message TB640_MSG_ID_SIP_OP_SAP_ALLOC contains the field:

```
	…
	TBX_RESULT			Result;
	TB640_SIP_SAP_HANDLE		hSipSap;
	…
```

6.2.1.4 Transport Server configuration

The SIP stack uses transport server to send/receive message from the IP network. The transport server is defined by a transport address (IP address and port), a protocol type, a TUCL interface and a host name.

✍ When a SIP message is sent, the stack will choose the first available SAP transport server matching the protocol type. To choose a specific transport server, the customer must include a VIA header.

The TB640_SIP_TRANSPORT_SERVER contains the configuration parameters for a transport server:

```
typedef struct _TB640_SIP_TRANSPORT_SERVER
{
	TBX_UINT32				un32TransportSrvId;
	TB640_TUCL_INTERFACE_HANDLE		hTuclIf;
	TB640_SIP_TRANSPORT_ADDRESS		Add;
	TB640_SIP_TRANSPORT_PROTOCOL_TYPE	Prot;
	TBX_CHAR				szHostName[TB640_SIP_MAX_HOSTNAME_SIZE];

} TB640_SIP_TRANSPORT_SERVER, *PTB640_SIP_TRANSPORT_SERVER;
```

un32TransportSrvId: Contains a unique transport server identifier. This identifier can be used back to matched the alarms with TB640_SIP_ALARM_INFO. The identifier values must be higher than 0.

hTuclIf: Contains the TUCL Interface handle.

Add: Transport address configuration for this Transport SAP server listen address (TB640_SIP_TRANSPORT_ADDRESS). IP address and port.

Prot: Transport protocol configuration for this Transport SAP server (TB640_SIP_TRANSPORT_ PROTOCOL_TYPE). Either UDP, TCP or TLS_TCP.

szHostName: The host part of the host name in the SIP URL that is associated with this transport address. The domain name *szDomainName* is found in the associated SIP entity configuration structure (TB640_SIP_COMMON_ENTITY_CFG). The hostname is used by the SIP layer while generating the call ID.

6.2.2 System Manager Layer configuration

The SIP and TUCL protocols stack operate inside the System manager's fault-tolerant architecture event if they are not yet supporting the fault tolerance. This will enable the SIP and TUCL protocols for future FT support without API modification. The system must be initialized with only one Active System Manager and no Standby.

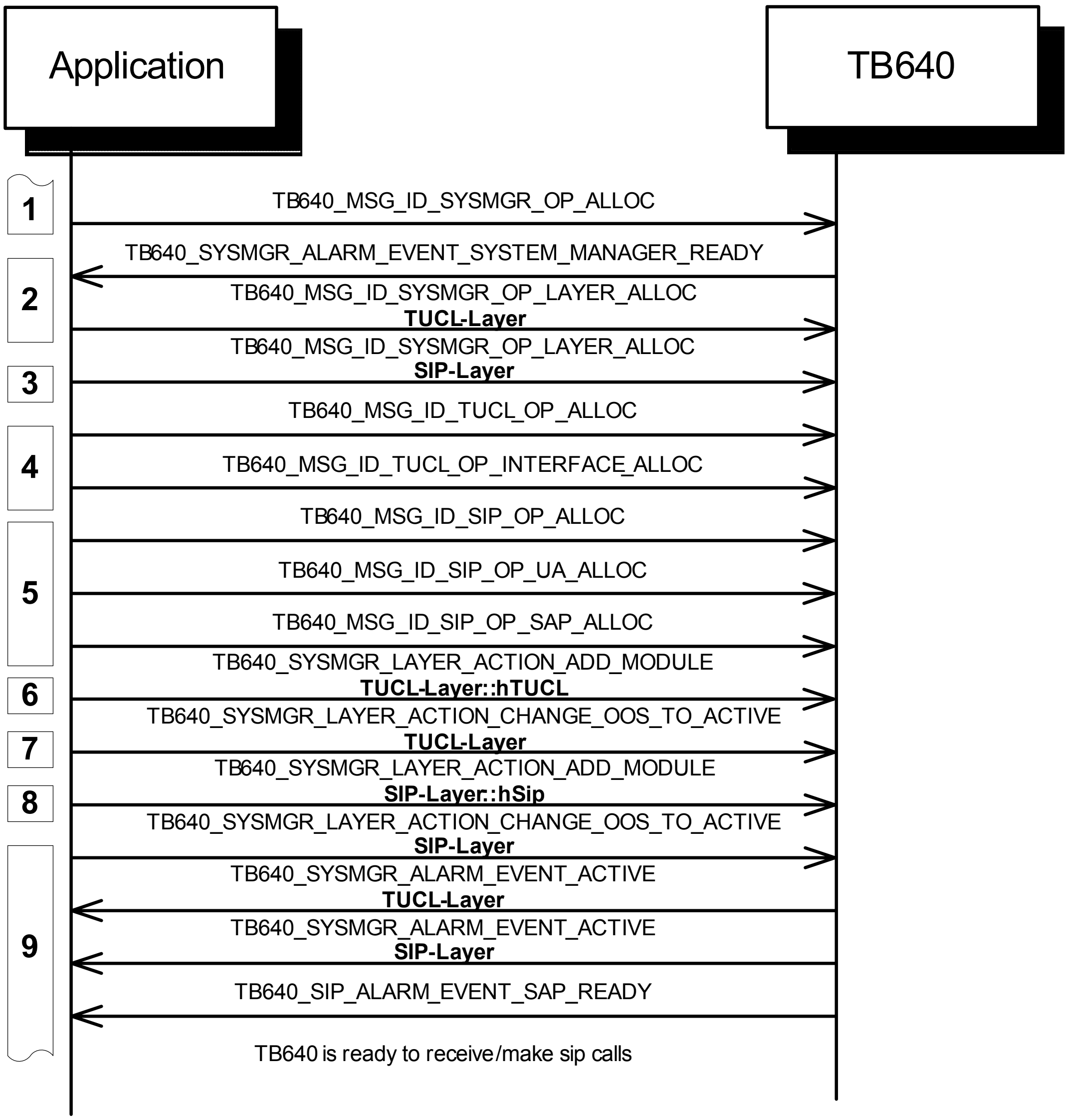


Figure 4 - System Manager Layer message configuration flow

General guidelines for configuration of SIP and TUCL protocol stack inside the System manager's fault-tolerant architecture include the following steps:

1. The system manager general allocation (TB640_MSG_ID_SYSMGR_OP_ALLOC), for the master, **must precede** all other messages (other protocol stack configuration alloc, set, get, states and stats) on any adapter(s) of its system. The return value for this operation is a system manager handle.

2. System manager TUCL layers allocation (TB640_MSG_ID_SYSMGR_OP_LAYER_ALLOC) must be made (*using the handle of the master system manager from **step 1***). The return value for this operation is a Layer handle.

3. System manager SIP layers allocation (TB640_MSG_ID_SYSMGR_OP_LAYER_ALLOC) must be made (*using the handle of the master system manager from **step 1***). The return value for this operation is a Layer handle.

4. TUCL stack allocation (*using the handle of the system manager layer from **step 2***). Refer to TUCL section 5.2 Stack Configuration.

5. SIP stack allocation (*using the handle of the system manager layer from **step 3***). Refer to SIP section 6.2.1 Stack Configuration.

6. Add TUCL stack module to system manager layer. Use system manager layer action TB640_MSG_ID_SYSMGR_CMD_LAYER_ACTION message with Action = TB640_SYSMGR_ LAYER_ACTION_ADD_MODULE (*with the handle of a Layer from **step 2** and with the handle of hTucl module from **step 4***).

7. Set TUCL stack module in-service inside system manager layer. Use system manager layer action TB640_MSG_ID_SYSMGR_CMD_LAYER_ACTION message with Action = TB640_SYSMGR_LAYER_ACTION_CHANGE_OOS_TO_ACTIVE (*with the handle of a Layer from **step 2** and with the handle of hTucl module from **step 4***).

8. Add SIP stack module to system manager layer. Use system manager layer action TB640_MSG_ID_SYSMGR_CMD_LAYER_ACTION message with Action = TB640_SYSMGR_ LAYER_ACTION_ADD_MODULE (*with the handle of a Layer from **step 3** and with the handle of hSip module from **step 5***).

9. Set SIP stack module in-service inside system manager layer.. Use system manager layer action TB640_MSG_ID_SYSMGR_CMD_LAYER_ACTION message with Action = TB640_SYSMGR_ LAYER_ACTION_ADD_MODULE (*with the handle of a Layer from **step 3** and with the handle of hSip module from **step 5***).

✍ The **step 2** must wait the reception of system manager allocation ready event TB640_SYSMGR_ALARM_ EVENT_SYSTEM_MANAGER_ READY.

✍ The SIP stack is ready to receive/send SIP message once those events have been received:

1. TB640_SYSMGR_ALARM_EVENT_ACTIVE with the handle of a Layer from **step 2** and with the handle of *hTucl* module from **step 4.**
2. TB640_SYSMGR_ALARM_EVENT_ACTIVE with the handle of a Layer from **step 3** and with the handle of *hSip* module from **step 5.**
3. TB640_EVT_SIP_NOTIF_ALARM with the *Event* TB640_SIP_ALARM_EVENT_SAP_ READY.

6.3 User Agent Functionalities

6.3.1 SIP message request

The TB640 SIP request message API follows one of the following formats:

TB640_MSG_ID_SIP_CMD_UA_xxx_REQ : Represents outgoing SIP message request command from the Host application to the TB640 stack.

TB640_MSG_ID_SIP_NOTIF_UA_xxx_IND : Represents incoming SIP message request notification from the TB640 stack to the Host application.

For a complete list of TB640 SIP message request description, see *TB640_sip.h*.

6.3.2 SIP message response

The TB640 SIP response message API follows one of the following formats:

TB640_MSG_ID_SIP_CMD_UA_xxx_RSP : Represents outgoing SIP message response command from the Host application to the TB640 stack.

TB640_MSG_ID_SIP_NOTIF_UA_xxx_CFM : Represents incoming SIP message response notification from the TB640 stack to the Host application.

For a complete list of TB640 SIP message response description, see *TB640_sip.h*.

6.3.3 SIP API message identifiers

The SIP protocol is a transactional based protocol. One of the most important transactions of the SIP protocol is the session initiation INVITE request transaction. In the TB640 API messages, the session also refers to the call itself. Both, the session and the call are tightly related. Inside a session, multiple peer-to-peer SIP relationship between two UAs may persists for some time, it's called a Dialog. It is recommended to use an incremental identifier allocation scheme since identifiers may still be active when a call is terminated (the SIP protocol has timers that go up to 32 seconds). The following parameters are used to identify the sip session/call and its associated dialogs and transactions.

6.3.3.1 Session Identifier

An incoming or outgoing SIP call/session is uniquely identified by a session identifier. In SIP architecture, it is possible that a single SIP session consists of multiple independent dialogs. All the dialogs within a given SIP session share the same session ID. Multiple dialogs within a SIP session are distinguished using another parameter called the "dialog ID".

Each session within a given SIP stack entity is identified by a unique "Session ID". When a new SIP session is created from the service user (e.g, INVITE transaction) it must allocate a session ID within the range 0x80000000 to 0xFFFFFFFF.

Similarly, if the TB640 SIP stack initiates a new session (e.g, INVITE received from peer node), it allocates a session ID within the range 0x00000001 to 0x7FFFFFFF.

6.3.3.2 Dialog Identifier

SIP session may consist of multiple dialogs (or call legs). One example of SIP session consisting of multiple dialogs is when downstream network server performs forking. In such a case, SIP stack may receive multiple 200 response and multiple dialogs will be created - one for each destination.

Each dialog within a given SIP session is identified by a unique "dialog ID". Since they belong to same session, they share the same session ID. When a service user initiates a new dialog (e.g, establishing a SIP session), it must allocate dialog ID within the range 0x80000000 to 0xFFFFFFFF.

Similarly, if a SIP stack initiates a new dialog (e.g, INVITE received from peer node), it allocates a dialog ID within the range 0x00000001 to 0x7FFFFFFF.

6.3.3.3 Transaction Identifier

SIP dialog may consist of multiple transactions. For example, SIP dialog may consist of active OPTIONS and REFER transaction.

Each transaction within a given Dialog is identified by a unique "Transaction ID". When a new SIP request transaction is created from the service user (e.g, INVITE transaction for session establishment), it must allocate a transaction ID within the range 0x80000000 to 0xFFFFFFFF.

Similarly, if the TB640 SIP stack initiates a transaction (e.g, INVITE received from peer node), it allocates transaction Id. The transaction Id's allocated by the TB640 SIP stack lies within the range 0x00000001 to 0x7FFFFFFF.

6.3.4 Scenarios showing uses of API message identifiers

6.3.4.1 Outgoing call setup and termination

The next figure shows two complete call setup initiated by the Host application. It also shows how generate and use identifiers in a call setup. In this example, the Session 2 is terminated by the peer user agent so the transaction identifier is provided by the TB640 stack.

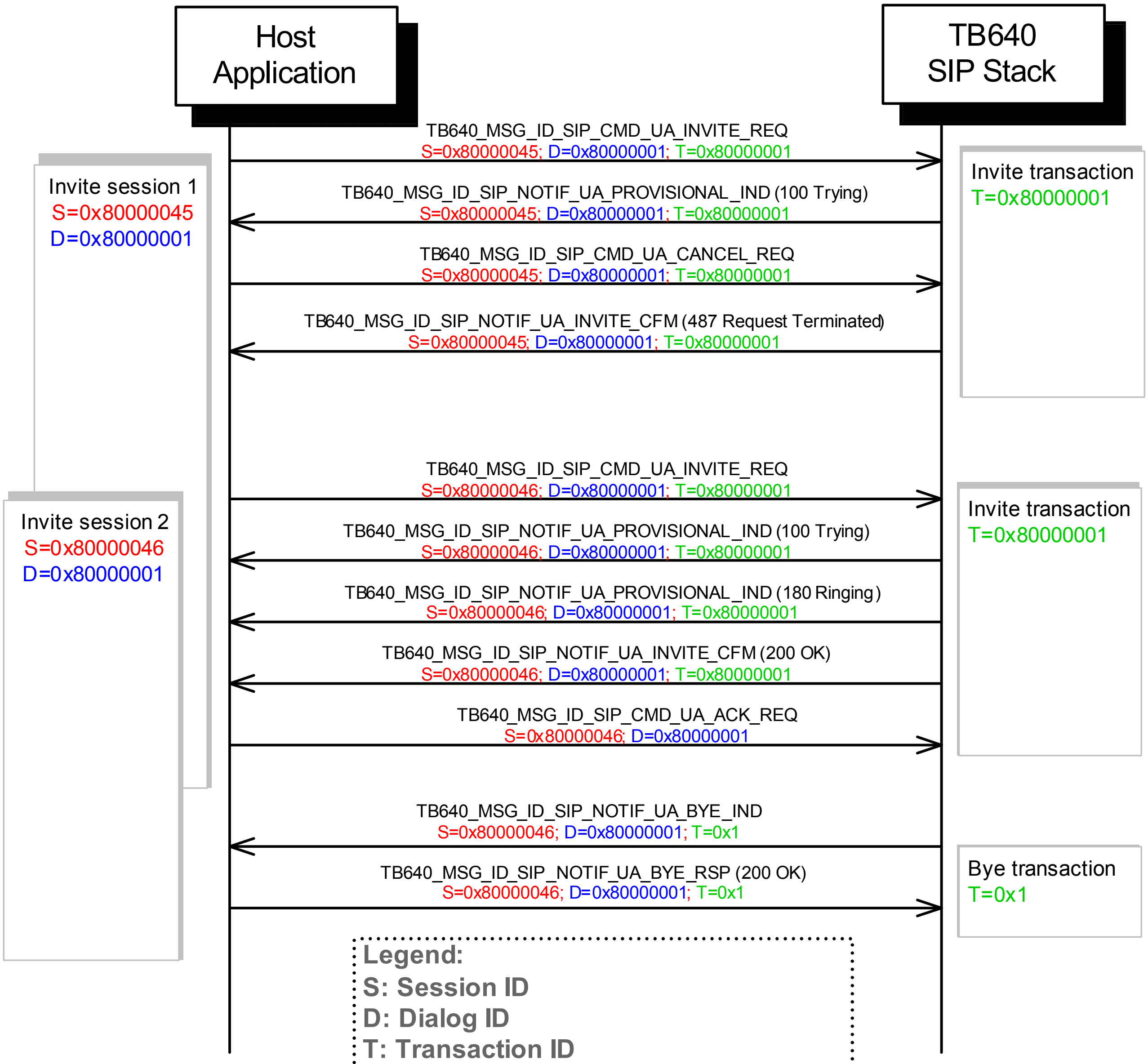


Figure 5 - Outgoing call setup and termination

6.3.4.2 Incoming call setup and termination

The next figure shows two complete call setup initiated by a peer end user. It also shows how the identifier must be used when generated by the TB640 SIP stack. In this example, the Session 1 is terminated by the Host Application and the transaction ID is provided by it.

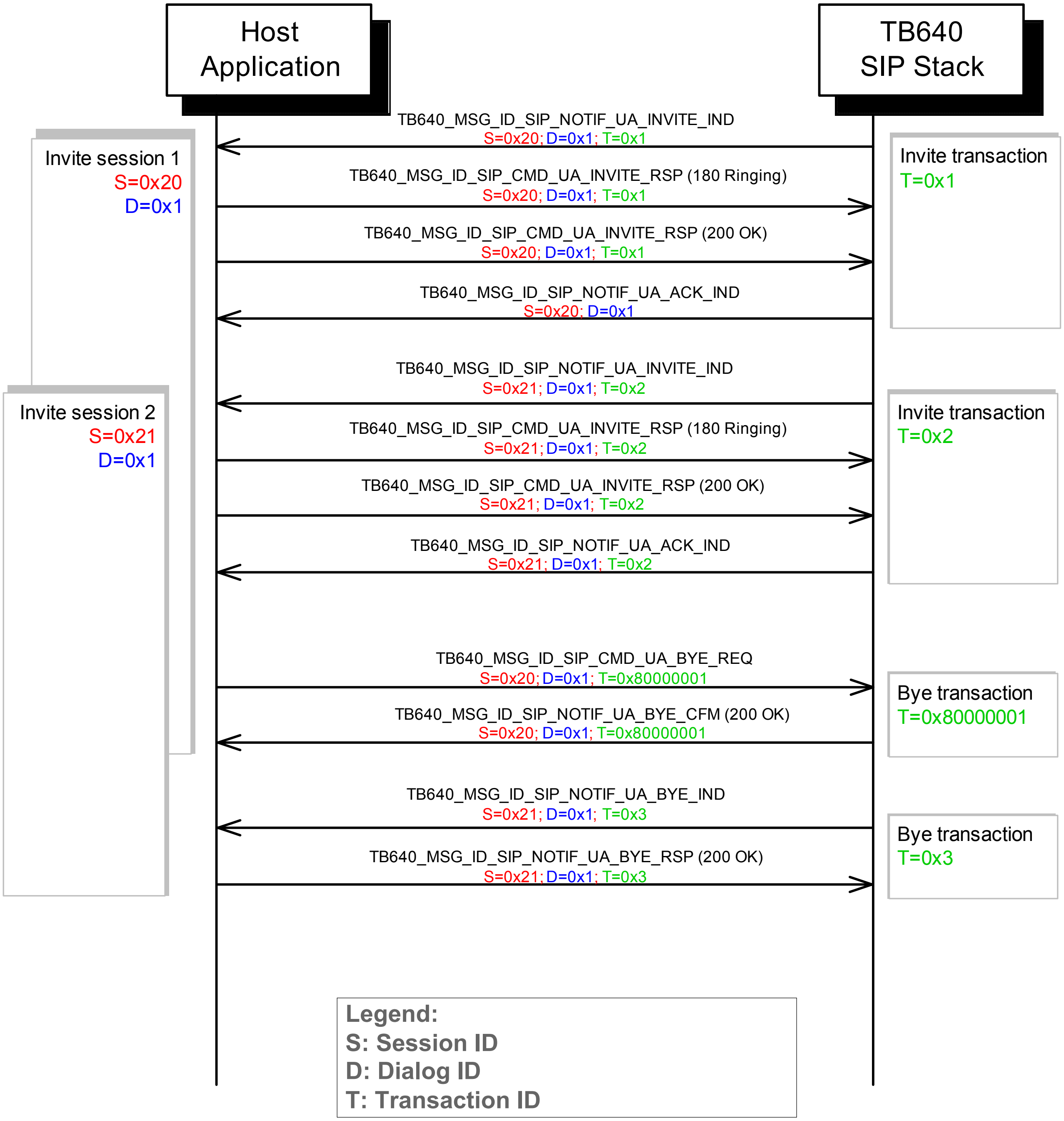


Figure 6 - Incoming call setup and termination

6.4 SIP/SDP Information element usage

6.4.1 Overview

SIP headers and SDP fields are being parsed, or decoded, on the TB640. The decoded format used is based on the MSGSET API, see tbx_msgset.h and associated doxygen documentation. The MSGSET is an API allowing the creation of recursive "linked list" of elements inside a TBX message.

The MSGSET API was considered the right choice on how to represent SIP headers and SDP since some SIP headers may include any number of parameters, and those parameters may include headers themselves.

6.4.2 SIP Utility Library

To help creating and filling SIP headers and SDP elements, a library is provided along with its source code. It is located in the host package, under *apps/lib* sub folder. The source code can be found in *apps/src*. This library allows copying; adding and filling of commonly used SIP headers and SDP fields.

The naming convention informs exactly what the each library function is used to.

TB640AAAYYY_XXX_xxx_xxx

AAA: Structure name
YYY: Action
XXX: Base-struct member
xxx: Sub-struct member

Example 1: Filling the SIP request line.
TB640SipRequestLineFill_Sip():

- Fills structure TB640_SIP_REQUEST_LINE with a "sip:" address scheme. See TB640_SIP_ADDRSPEC for a list of supported schemes.

Example 2: Adding a "From:" SIP header.
TB640SipHeaderAdd_From_Sips():

- Adds a From (TB640_SIP_HEADER_FROM) header into a TB640_SIP_HEADER_MSGSET with a "sips:" address scheme.

Note: Add Action adds an ELEMENT into a "linked list", thus it can only be done on a MSGSET.

Example 3: Copying a From SIP header.
TB640SipHeaderFromCopy()

- Copies a TB640_SIP_HEADER_FROM header from one TBX message to another.

See *apps/inc/tb640_sip_util.h* for a complete list of SIP utility functions.

6.4.3 SIP Message

A SIP Message is composed of a SIP header and a body. The SIP header is either a Request Header or a Response Header. The Request Header's first line is a Request-Line and the Response Header's first line is a Status-Line. Other than this difference, both Request Header and Response Header are much alike. Let's focus on the SIP Request Header.

A SIP request message is composed of a SIP header and a body.

```
/*!
 * \struct _TB640_SIP_MESSAGE_REQUEST
 *
 * \brief      Sip Request Message
 *
 * abnf        SIP-message = request CRLF [ message-body ]
 *
 * \param      Header          : Sip Request Header
 * \param      Body            : Message body
 */
typedef struct _TB640_SIP_MESSAGE_REQUEST
{
       TB640_SIP_HEADER_REQUEST      Header; /*!< Sip Request Header */
       TB640_SIP_BODY                Body;   /*!< Message body */

} TB640_SIP_MESSAGE_REQUEST, *PTB640_SIP_MESSAGE_REQUEST;
```

6.4.3.1 Sip Message Header

The request SIP header may be in decoded format, or in opaque format. Decoded format uses "C" structures and opaque format uses plain text directly.

```
/*!
 * \brief      Sip Request Header
 *
 * abnf        request         = decoded / opaque
 *
 * \param      Type            : Sip Message type, TB640_SIP_HEADER_MSG_TYPE_DECODED
 *                                              or TB640_SIP_HEADER_MSG_TYPE_OPAQUE
 * \param      DecodedMsg      : Decoded sip message header (structured)
 * \param      strOpaqueMsg    : Opaque sip message header (text) Composed of one or many strings
 */
typedef struct _TB640_SIP_HEADER_REQUEST
{
       TB640_SIP_HEADER_MSG_TYPE     Type;          /*!< Sip Message type */

       union
       {
              TB640_SIP_REQUEST      Request;       /*!< TB640_SIP_HEADER_MSG_TYPE_DECODED */
              TB640_SIP_STRING       strOpaqueMsg;  /*!< TB640_SIP_HEADER_MSG_TYPE_OPAQUE */
       };

} TB640_SIP_HEADER_REQUEST, *PTB640_SIP_HEADER_REQUEST;
```

The request header is composed of:

- The Request member is a "C" declared structure representation of a SIP message.
- The strOpaqueMsg is a MSGSET of strings. It may be composed of one or many NULL-terminated strings. The utility function TB640SipStringAdd() may be used to insert a completely preformatted SIP header in its text format.

✍ When using opaque messages, all mandatory fields must be filled.


```
/*!
 * \brief      Sip Request
 *
 * abnf        Request        = Request-Line *( message-header )
 *
 * \param      RequestLine    : Request-Line
 * \param      setHeader      : Set of one or many message-header(s)
 *
 */
typedef struct _TB640_SIP_REQUEST
{
       TB640_SIP_REQUEST_LINE        RequestLine;   /*!< Request-Line */
       TB640_SIP_HEADER_MSGSET       setHeader;     /*!< Set of one or many message-header(s) */

} TB640_SIP_REQUEST, *PTB640_SIP_REQUEST;
```

The SIP request is composed of:

- The RequestLine, that may be filled using TB640SipRequestLineFill_XXX() utility function.
- The setHeader, which is a MSGSET. It represents a “linked list” of 0 to many SIP headers (ie: From:, To:, Contact:, etc). Utility functions like TB640SipHeaderAdd_XXX () may be used to add all desired SIP headers.

6.4.3.2 Sip Message Body

The body is usually single-part SDP. But the body could also be multipart, or absent. The SDP part of the body may be in decoded format or in opaque format. Decoded format uses “C” structures and opaque format uses plain text directly.

```
/*!
 * \brief      Sip Body
 *
 *             The sip message body may be absent, sdp single-part or multi-part.
 *
 * \param      Type                 : Sip body type, single-part or multi-part
 * \param      SinglePart           : Single part body
 * \param      setMultiPartBodyElm  : One or many Multi-part body elements
 *
 */
typedef struct _TB640_SIP_BODY
{
       TB640_SIP_BODY_TYPE           Type;   /*!< Sip body type, single-part or multi-part */
       union
       {
              TB640_SIP_BODY_SINGLE_PART           SinglePart;
              TB640_SIP_BODY_MULTI_PART_MSGSET     setMultiPartBody;
       };

} TB640_SIP_BODY, *PTB640_SIP_BODY;
```

The SIP body is either:

- Absent
- SinglePart: Single part body
- setMultiPartBody: A MSGSET of 0 to n multipart bodies. (of type TB640_SIP_BODY_MULTI_PART)

```
/*!
 *
 *
 * \brief      Sip multi-part body structure definition
```

```
 *
 *           This structure represent a multi-part body.  A multi-part body
 *           is composed of two or more parts.  Any parts within a multi-part
 *           may itself be mutli-part.
 *
 *           Each multi-part must contain:
 *                  \ref TB640_SIP_HEADER_MIME_VERSION header
 *                  \ref TB640_SIP_HEADER_CONTENTTYPE
 *                                       with MediaType set to \ref
TB640_SIP_MEDIA_TYPE_MULTIPART
 *                                       with a unique "boundary" parameter
 *
 * \param     setHeader      : Set of one or many message-header(s)
 * \param     Body           : Body of this Multi part body
 *
 */
typedef struct _TB640_SIP_BODY_MULTI_PART
{
       TB640_SIP_HEADER_MSGSET       setHeader;     /*!< Set of one or many message-header(s) */
       TB640_SIP_BODY                Body;          /*!< Body of this Multi part body */

} TB640_SIP_BODY_MULTI_PART, *PTB640_SIP_BODY_MULTI_PART;
```

The multipart body is composed of:

- setHeader: 0 to n Headers fields. (ie:Mime-Version:, Content-Type:,etc)
- Another Body, which may itself be single-part or multipart.

```
/*!
 * \brief     Sip single part body structure definition
 *
 *           This structure represent a single part within a body.  This single
 *           part may be part of a multi-part body.
 *           This structure to send any type of payload given that Type is set
 *           to \ref TB640_SIP_BODY_PART_TYPE_OPAQUE and that the payload is pre-fromated.
 *
 *           The body part type \ref TB640_SIP_BODY_PART_TYPE_SDP allows usage of
 *           the decoded sdp body part as defined in tb640_sdpie.h
 *
 * \param     Type           : Type of this body, either single-parted or multi-parted.
 * \param     SdpInfo        : Decoded Sdp body part if type is SDP as defined in tb640_sdpie.h
 * \param     setOpaqueMsg   : Body part is pre-encoded sdp, or a simple byte array.
 *
 */
typedef struct _TB640_SIP_BODY_SINGLE_PART
{
       TB640_SIP_BODY_PART_TYPE      Type
       union
       {
              TB640_SDP_INFO         SdpInfo;
              TB640_SIP_BYTES        sbyOpaqueMsg;
       };

} TB640_SIP_BODY_SINGLE_PART, *PTB640_SIP_BODY_SINGLE_PART;
```

The single-part body is composed of:

- SdpInfo: SDP body decoded into "C" structures.
- The sbyOpaqueMsg is a MSGSET of bytes. It may be composed of one byte arrays. The utility function TB640SipBytesAdd() may be used to insert a complete preformatted single-part body in its text or byte format.

6.4.4 SIP Header information element

All SIP headers and information elements composing sip headers are described in "C" structure, with their associated ABNF equivalent in the declaration's comment header.

```
/*!
 *
 *
 * \brief      Sip From Header - RFC3261
 *
 * abnf        From    = ( "From" / "f" ) HCOLON ( name-addr / addr-spec ) *( SEMI addr-param )
 *
 *             Examples:
 *                    From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
 *
 * \param      HdrType        : Header Type \ref TB640_SIP_HEADER_TYPE_GEN_FROM
 *                                          \ref TB640_SIP_HEADER_TYPE_GEN_FROMF
 * \param      NameAddr       : Name-addr pair
 * \param      setAddrParam   : Set of optional address parameter(s)
 *
 * \ref TB640_SIP_HEADER_TYPE_GEN_FROM
 * \ref TB640_SIP_HEADER_TYPE_GEN_FROMF
 *
 */
typedef struct _TB640_SIP_HEADER_FROM
{
       TB640_SIP_HEADER_TYPE          HdrType;        /*!< Header Type */

       TB640_SIP_NAME_ADDR            NameAddr;       /*!< Name-addr pair */
       TB640_SIP_ADDR_PARAM_MSGSET    setAddrParam;   /*!< Set of optional address parameter(s) */

} TB640_SIP_HEADER_FROM, *PTB640_SIP_HEADER_FROM;
```

The From: header is composed of:

- Header type, either TB640_SIP_HEADER_TYPE_GEN_FROM or TB640_SIP_HEADER_TYPE_GEN_FROMF (compact form).
- Mandatory (name-addr / addr-spec) Pair.
- Optional set of address parameters.

Optional fields use MSGSET API. A MSGSET is composed of "linked list" 0 to n ELEMENTs. ELEMENTs type is deduced from the name of the type, by removing the "_MSGSET" suffix. For instance, TB640_SIP_ADDR_PARAM_MSGSET is a "linked list" of structure of type TB640_SIP_ADDR_PARAM.

Most common headers and parameters may be added or filled through the SIP utility library. In some cases, direct usage of the MSGSET API may be needed. Please refer to tbx_msgset.h documentation.

6.4.5 SDP information element

All SDP information elements are described in "C" structure, with their associated ABNF equivalent in the declaration's comment header (in other words just like the SIP structures). The SDP information element has many sub-elements but the most important ones are the media descriptions. The easiest way to extract useful information from the SDP information element is by using TB640SdpParse (). This fills the TBX_SDP_INFO structure.

```
/*!
```

```
 * \struct _TBX_SDP_INFO
 *
 * \brief      Session Description Info structure
 *
 *             Structure used to describe a Media Session.
 *
 * \param      Capabilities           Media Capabilities
 * \param      un8NbConnections       Number of connections in aConnections
 * \param      fIsCapabilityMaster    Tells if this SDP Info describes a RTP entity that is
 *                                    "master" (else "slave") in the capability choice.
 *                                    A "master" will always choose codec to send according to what
 *                                    it prefers receiving for itself. A "slave" will always
 *                                    choose codec to send according to what remote peer prefers to
 *                                    receive. Note: This notion is important for H324M devices,
 *                                    but SIP devices are always slave.
 * \param      aConnections           Array of connections. Index 0 is reserved for session level
 *                                    connection
 */
typedef struct _TBX_SDP_INFO
{
       TBX_MEDIA_CAPABILITIES         Capabilities;

       TBX_UINT8                      un8NbConnections;
       TBX_BOOLEAN                    fIsCapabilityMaster;
       TBX_UINT8                      un8Padding[ 6 ];

       TBX_SDP_MEDIA_CONNECTION       aConnections[ TBX_MEDIA_MAX_NB_OF_SIMULTANEOUS_CAPABILITES ];

} TBX_SDP_INFO, *PTBX_SDP_INFO;
```

TBX_MEDIA_CAPABILITIES contains the information from the media description lines. The information has been decoded so it is now possible to retrieve the encoding parameters easily.

It is possible to translate this structure directly to opaque SDP using TBXMediaLibSdpGenerate () which is in the media library.

6.4.6 Example: Building a SIP Invite Request

This example shows how to fill a SIP Message for an Invite Request. The Sip Message is composed of the following:

- The RequestLine, with an “sip:” address scheme
- The From: header fields
- The To: header fields
- A single-part opaque body, which is a preformatted SDP body.

Note: Some mandatory SIP headers fields like CSeq:, Via:, Call-Id: are added by the TB640 sip stack. See tb640_sip.h or the call flow section in this document for more details.

```
/* Example taken from sip sample */

/* Static array used to build opaque SDP */
TBX_CHAR       g_abySdpBuffer[ 1024 ];

SipSendCmdInviteRequest(
  IN    PSIP_ADAPTER_CONTEXT   in_pAdapterCtx,
  IN    PSIP_DIALOG_CONTEXT    in_pDlgCtx,
  IN    PSIP_TRANS_CONTEXT     in_pTransCtx )
{
TBX_RESULT                                          Result;
TBX_MSG_HANDLE                                      hMsg;
PTB640_MSG_SIP_CMD_UA_INVITE_REQ                    pMsg;
TBX_UINT32                                          un32MsgSize;
PTB640_SIP_REQUEST                                  pSipRequest;
PTB640_SIP_BODY                                     pSipBody;
TBX_UINT64                                          un64UserCtx1;
TBX_UINT64                                          un64UserCtx2;
PSIP_SAP_CONTEXT                                    pSipSapCtx;
PSIP_SESS_CONTEXT                                   pSessCtx;

/* Compute message size */
un32MsgSize = sizeof( *pMsg ) + 4096 /* Payload space for MSGSET */;

/* Get a message buffer */
Result = TBXGetMsg( g_pContext->hTbxLib, un32MsgSize, &hMsg );
if( TBX_RESULT_FAILURE( Result ) )
{
       TBX_EXIT_ERROR( Result, 0, "Failed to get message buffer.");
}

/* Set the message header */
TBX_FORMAT_MSG_HEADER
(
       hMsg,
       TB640_MSG_ID_SIP_CMD_UA_INVITE_REQ,
       TBX_MSG_TYPE_REQUEST,
       sizeof( *pMsg ),
       in_pAdapterCtx->AdapterInfo.hAdapter,
       un64UserCtx1,
       un64UserCtx2
);

/* Initialize message */
TBXMsgElementInit( hMsg, sizeof( *pMsg ) );

/* Fill the request */
pMsg                                 = TBX_MSG_PAYLOAD_POINTER (hMsg);
pMsg->Request.Message.Header.Type    = TB640_SIP_HEADER_MSG_TYPE_DECODED;
pSipBody                             = &pMsg->Request.Message.Body;
pSipRequest                          = &pMsg->Request.Message.Header._MsgType;
```

```
pMsg->Request.un32MsgVersion          = 1;
pMsg->Request.hSipSap                 = pSipSapCtx->hSipSap;
pMsg->Request.un32SessionId           = pSessCtx->un32SessionId;

Result = TB640SipRequestLineFill_Sip(
        hMsg,
        &pSipRequest->RequestLine,
        TB640_SIP_METHOD_TYPE_INVITE,
        TB640_SIP_USERINFO_TYPE_USER,
        in_pDlgCtx->szRemUser,
        NULL, /* password */
        TB640_SIP_HOST_TYPE_IPV4ADDRESS,
        SIP_INET_NTOA( &in_pDlgCtx->SipRemAdd ),
        in_pDlgCtx->SipRemAdd.Ipv4Add.un16Port );
if( TBX_RESULT_FAILURE( Result ) )
{
        TBX_EXIT_ERROR (Result, 0, "TB640SipRequestLineFill_Sip Failed" );
}

Result = TB640SipHeaderAdd_To_Sip(
        hMsg,
        &pSipRequest->setHeader,
        NULL, /* DisplayName */
        TB640_SIP_USERINFO_TYPE_USER,
        in_pDlgCtx->szRemUser,
        NULL /* password */,
        TB640_SIP_HOST_TYPE_IPV4ADDRESS,
        SIP_INET_NTOA( &in_pDlgCtx->SipRemAdd ),
        in_pDlgCtx->SipRemAdd.Ipv4Add.un16Port );
if( TBX_RESULT_FAILURE( Result ) )
{
        TBX_EXIT_ERROR( Result, 0, "TB640SipHeaderAdd_To_Sip Failed" );
}

Result = TB640SipHeaderAdd_From_Sip(
        hMsg,
        &pSipRequest->setHeader,
        NULL, /* DisplayName */
        TB640_SIP_USERINFO_TYPE_USER,
        in_pDlgCtx->szLclUser,
        NULL /* password */,
        TB640_SIP_HOST_TYPE_IPV4ADDRESS,
        SIP_INET_NTOA( &in_pDlgCtx->SipLclAdd ),
        in_pDlgCtx->SipLclAdd.Ipv4Add.un16Port );
if( TBX_RESULT_FAILURE( Result ) )
{
        TBX_EXIT_ERROR( Result, 0, "TB640SipHeaderAdd_From_Sip Failed" );
}

/* Build Sdp Opaque Body */
un16SdpBufferLen = sizeof( abySdpBuffer );
TBXMediaLibSdpGenerate( in_pDlgCtx->Info.pSdpInfo, (PTBX_CHAR)abySdpBuffer, &un16SdpBufferLen,
                        TBX_MEDIA_SDP_GENERATE_OPTION_MEDIA_DESC_CONN_ONLY );

/* Util Lib will set the Body type as single-part opaque */
Result = TB640SipBodyFill_SinglePart_sbyOpaqueMsg(
                hMsg,
                pSipBody,
                &pSipRequest->setHeader,
                g_abySdpBuffer,
                (TBX_UINT16)strlen( g_abySdpBuffer ) );
if( TBX_RESULT_FAILURE( Result ) )
{
        TBX_EXIT_ERROR (Result, 0, "TB640SipBodyFill_SinglePart_sbyOpaqueMsg Failed" );
}
}
/* Example - End */
```


6.5 Call flow and scenarios

The TB640 SIP stack offloads the host in many ways. This section will cover the call establishment and release in details in order to understand what exactly the stack does.

Here are the offloading features of the TB640 SIP stack:

- Handling of retransmission Requests upon packets losses
- Handling of retransmission Responses upon reception of duplicates Requests
- Automatic transmission of Provisional Status (ie: 100 Trying)
- Context wise filling of mandatory Sip Message Header Fields:
    - *Cseq:* Header field
    - *Call-Id:* Header field
    - *Via*: Header field and *branch* parameter
    - *tag*= parameter
    - *Contact:* Header field (If *From:* user is locally registered)
    - *From:* (Except on an INVITE REQUEST)
    - *To:* (Except on an INVITE REQUEST)
    - Parts of the Status-Line depending
- Configuration wise filling of some optional Sip Message Header Fields:
    - Allow:
    - Date:
    - Expires:
    - Supported:
    - Max-Forwards:
    - Subject:
    - Organization:

For a complete list of Header Fields that needs to be added or not, see the tb640_sip.h API documentation.

6.5.1 Outgoing call state diagram

The next figure shows the basic call states that an application must follow for an outgoing call establishment.

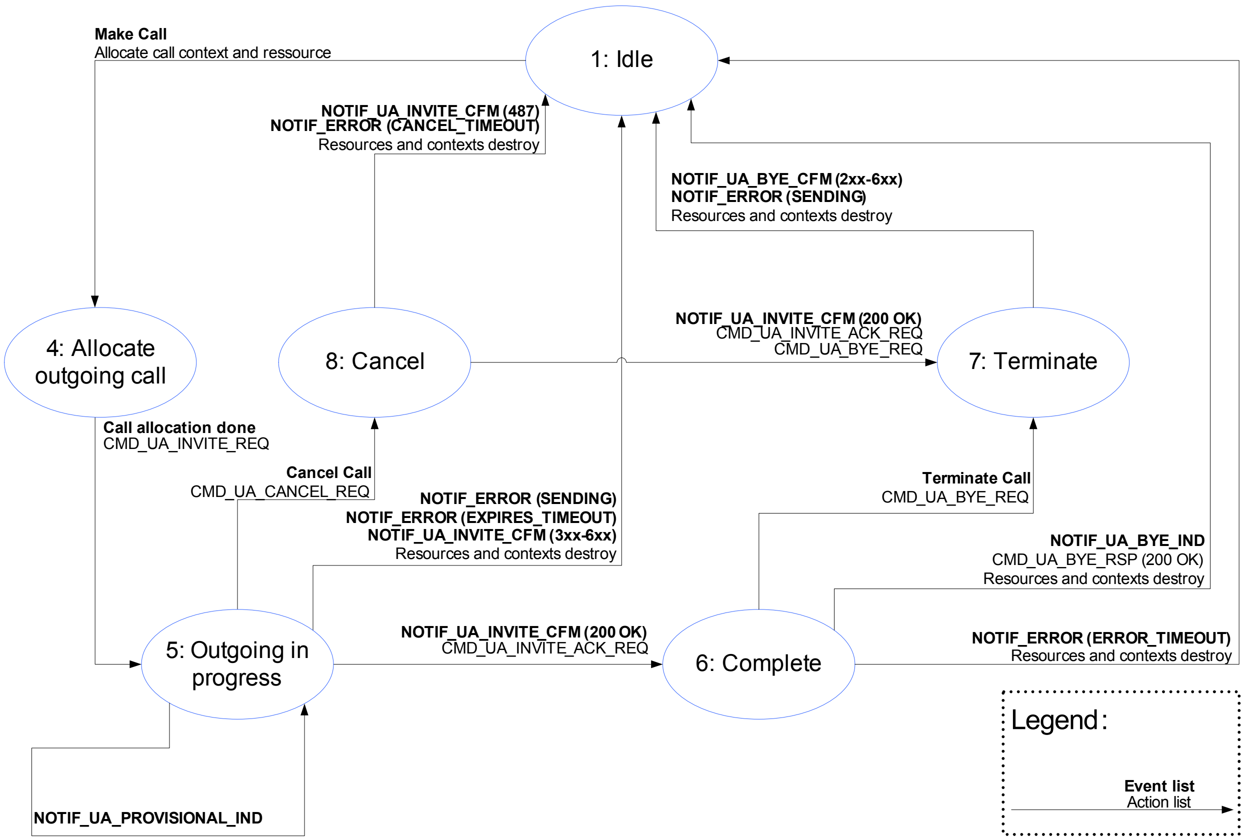


Figure 7 - Outgoing call states

6.5.2 Incoming call state diagram

The next figure shows the basic call states that an application must follow for an incoming call.

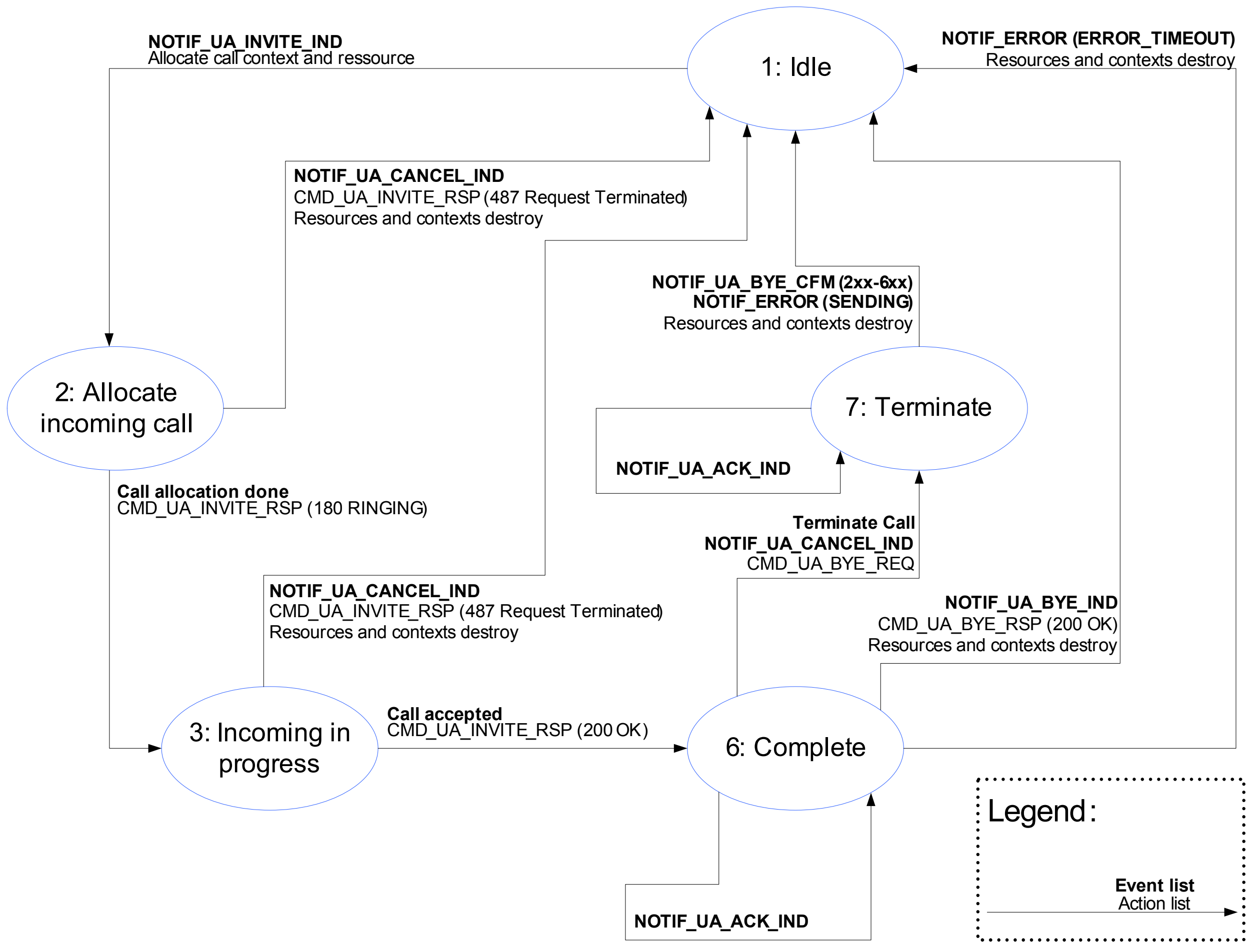


Figure 8 - Incoming call states

6.5.3 Establishing an outgoing call

This section will go through the establishment of an outgoing call in details. Packet losses scenarios are covered in order to understand the complete call flow. Then, each messages of the call flow will be detailed in order to see SIP Message Header offloading by the TB640 SIP stack.

6.5.3.1 Call flow with packet losses

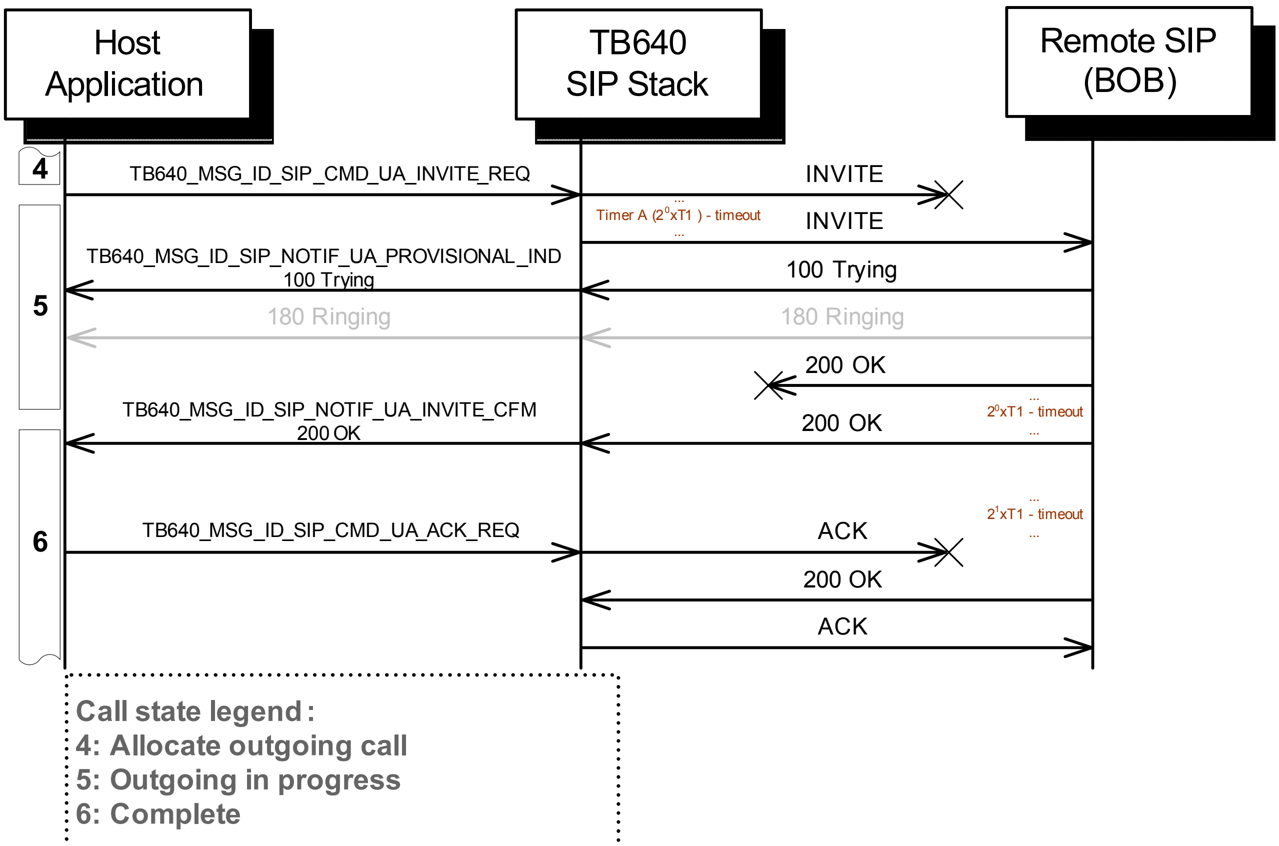


Figure 9 - Outgoing call flow with packet losses

In this call flow, the local host initiates an INVITE request. Unluckily, the INVITE request is lost in between the TB640 and the Remote SIP. After T1 milliseconds, the Timer A of the INVITE transaction expire and the request gets retransmitted by the TB640 SIP stack without the host having to take any action.

6.5.3.2 Call flow with error sending

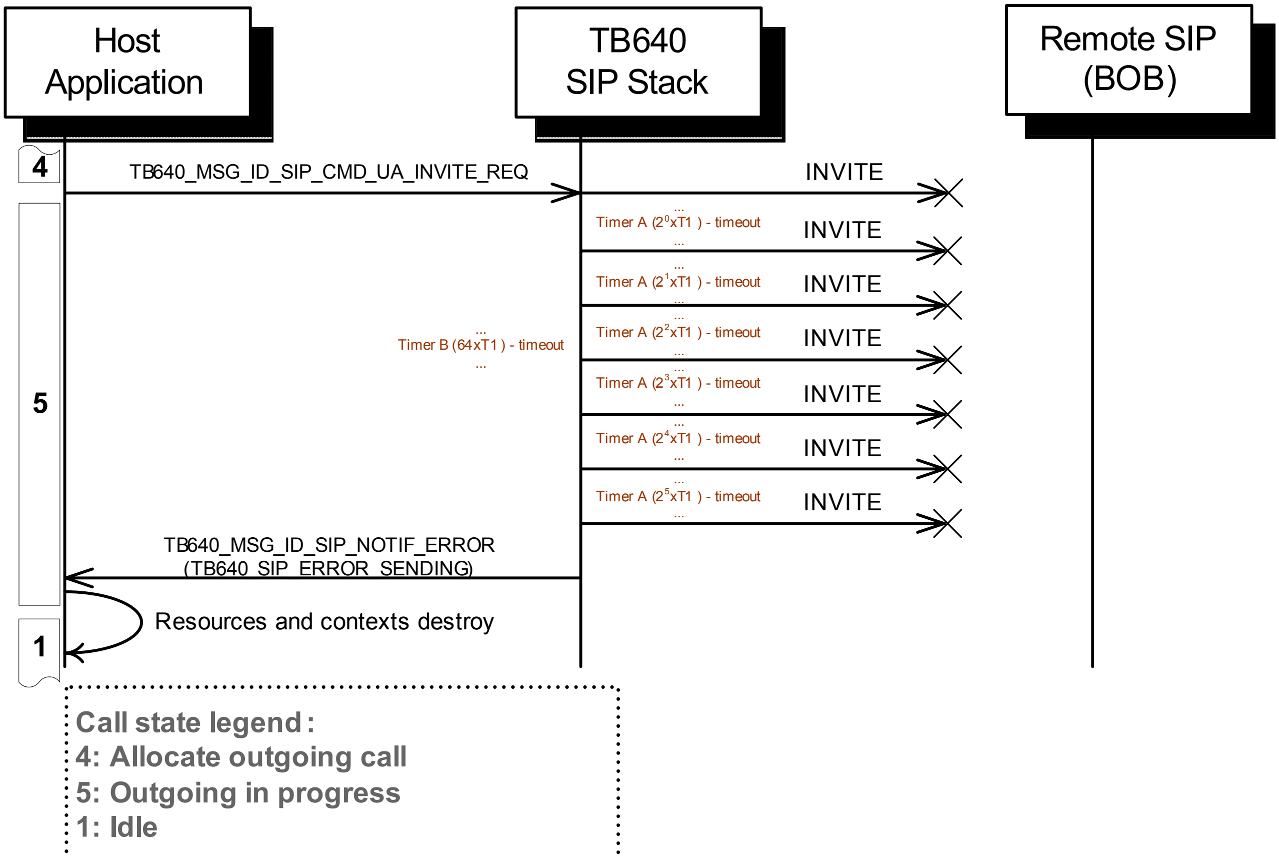


Figure 10 – Outgoing call flow with error sending

In this call flow, the local host initiates an INVITE request. Unluckily, the INVITE request never reaches the Remote SIP host (BOB). After T1 milliseconds, the Timer A of the INVITE transaction expire and the request gets retransmitted by the TB640 SIP stack without the host having to take any action. Each time the timer A expired, its value is multiplied by 2. After 64T1, if no response has been received by the stack, a TB640_MSG_ID_SIP_NOTIF_ERROR is generated by the stack with an error code TB640_SIP_ERROR_SENDING.

6.5.3.3 Call flow with remote expires

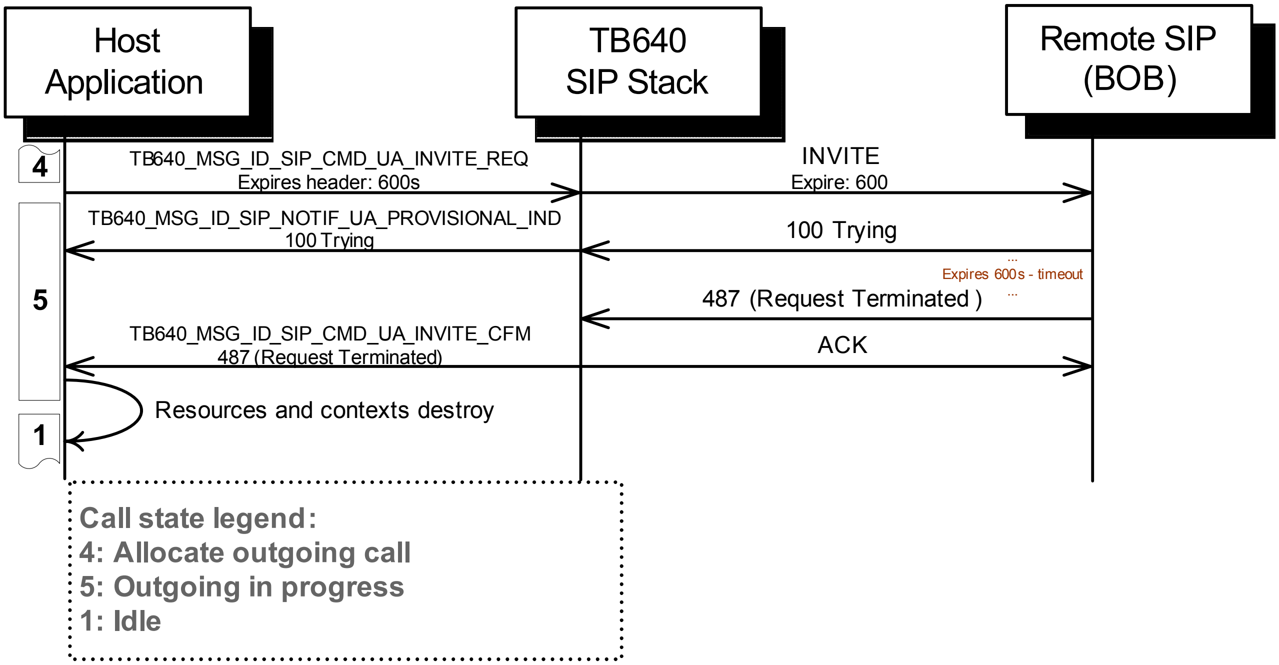


Figure 11 - Outgoing call flow with remote expires

In this call flow, the local host initiates an INVITE request with the Expires header set to 600 seconds. After 600 seconds, the remote SIP host (BOB) hasn't received any response from the host application. The remote Expires timer expire and a 487 (Request Terminated) INVITE response is generated by the remote SIP host and received by the TB640 SIP stack. It automatically generates the ACK.

6.5.3.4 Call flow with local expires

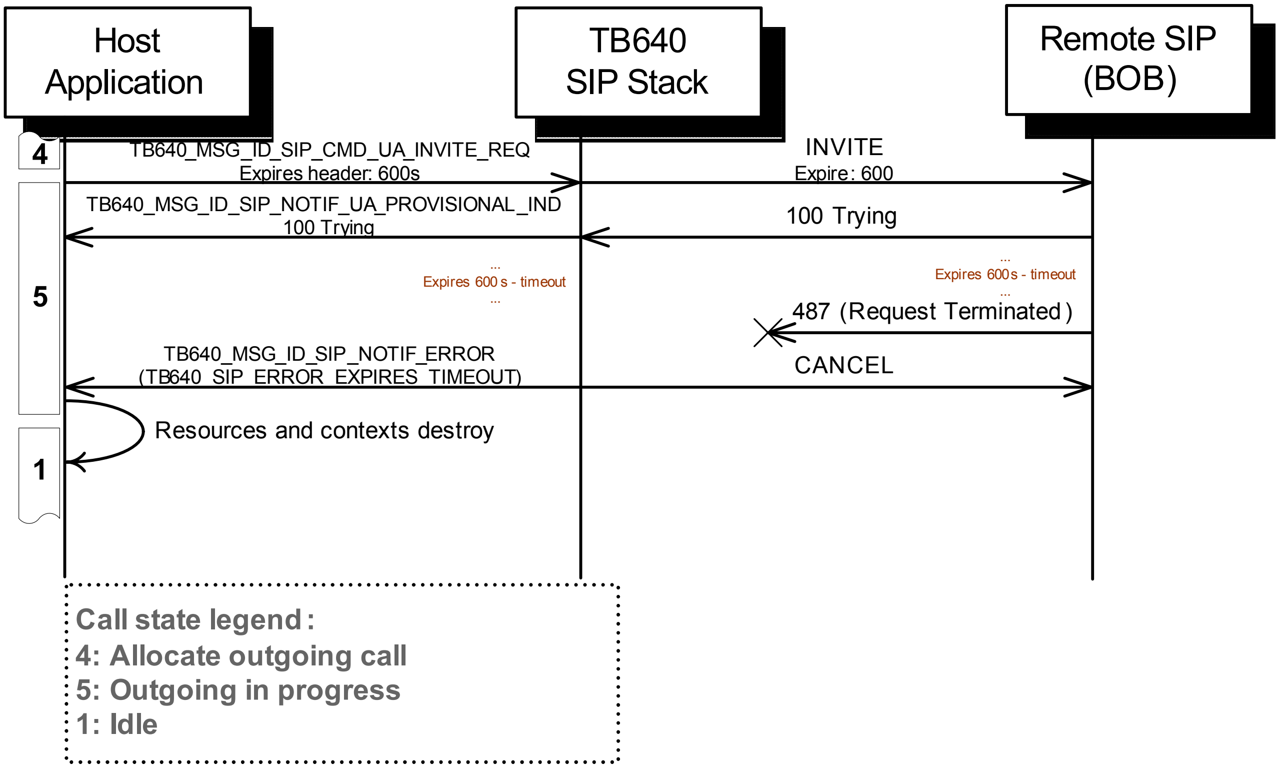


Figure 12 - Outgoing call with local expires

In this call flow, the local host initiates an INVITE request with the Expires header set to 600 seconds. After 600 seconds, the remote SIP host (BOB) hasn't received any response from the host application. The remote Expires timer expire and a 487 (Request Terminated) INVITE response is generated by the remote SIP host and not received by the TB640 SIP stack. The TB640 SIP stack Expires timer expired after 600 seconds and a TB640_MSG_ID_SIP_NOTIF_ERROR is generated by the stack with an error code TB640_SIP_ERROR_EXPIRES_TIMEOUT. The TB640 stack automatically generates a CANCEL message and if no response is received for the CANCEL, it generates a TB640_MSG_ID_SIP_NOTIF_ERROR is generated by the stack with an error code TB640_SIP_ERROR_CANCEL_TIMEOUT.

6.5.3.5 Call flow with host Cancel

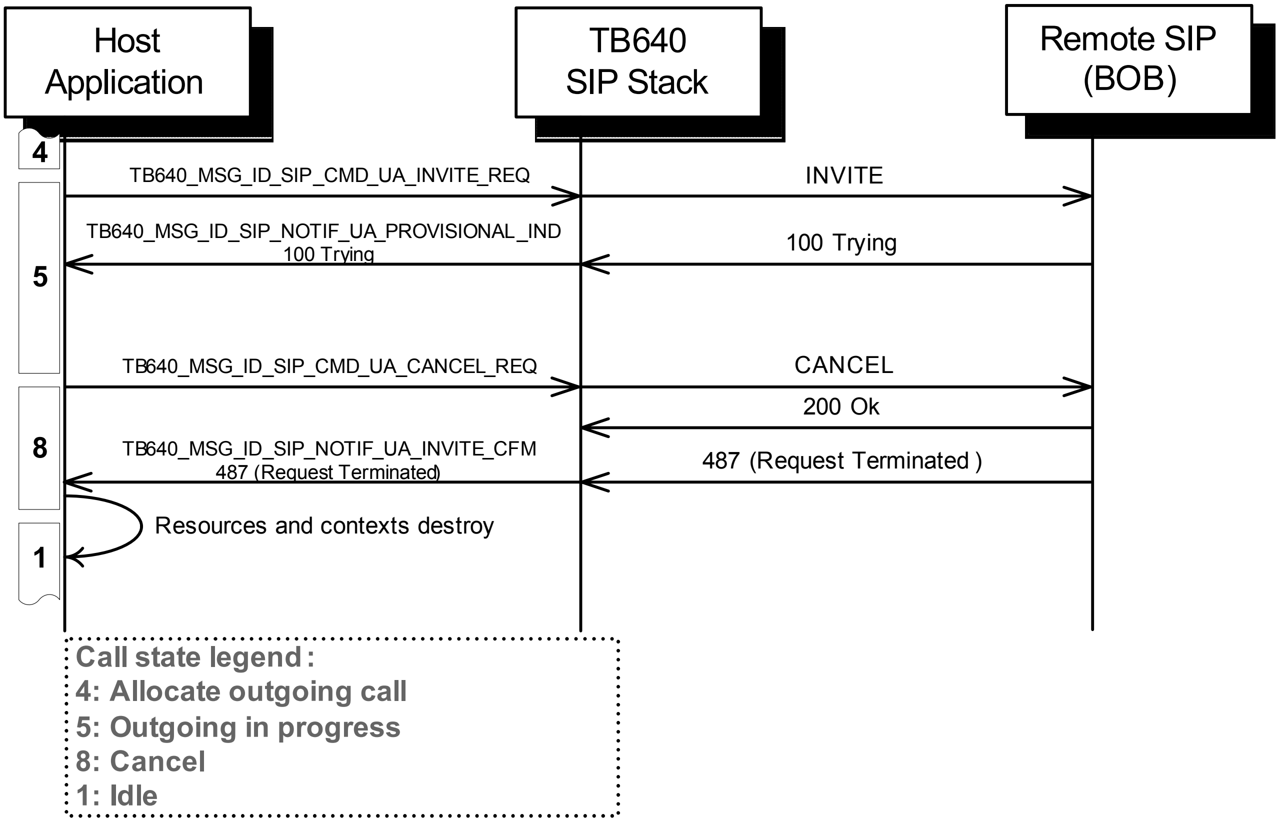


Figure 13 - Outgoing call flow with host cancel

In this call flow, the local host initiates an INVITE request and after a little while the customer application sends a CANCEL request toward the TB640 SIP stack using the transaction ID of the INVITE request to cancel. The local host application will only receives the INVITE 487 Request Terminated response, the CANCEL OK response been absorbed by the TB640 SIP stack.

6.5.3.6 Call flow with no expire header

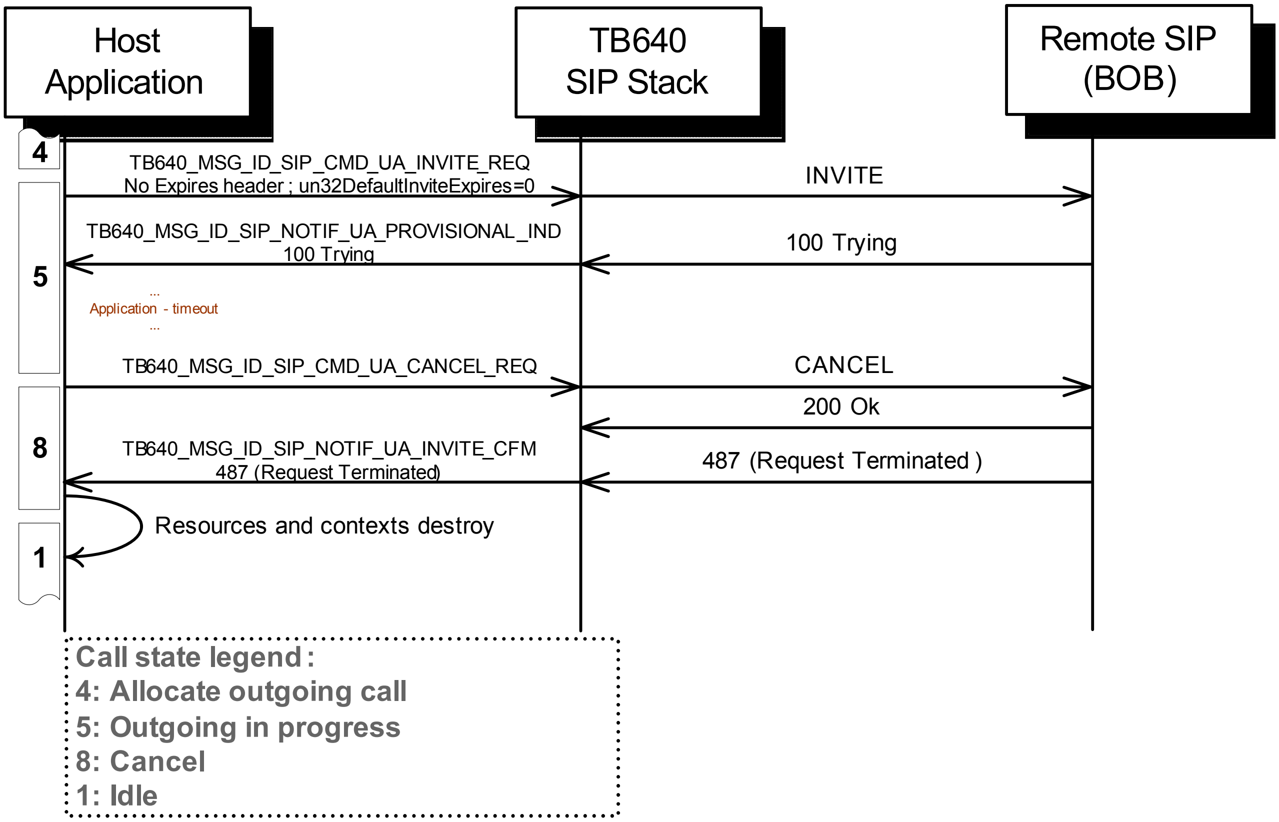


Figure 14 - Outgoing call flow with not expire header

In this call flow, the local host initiates an INVITE request with no Expires header but with the *un32DefaultInviteExpires* set to 0 seconds (see 6.2.1.2 UA entity configuration). The Remote SIP host (BOB) is waiting for a user call acceptation. After a certain time, the local host application timeout and sends a CANCEL request toward the TB640 SIP stack.

6.5.3.7 Call flow with host Cancel timeout

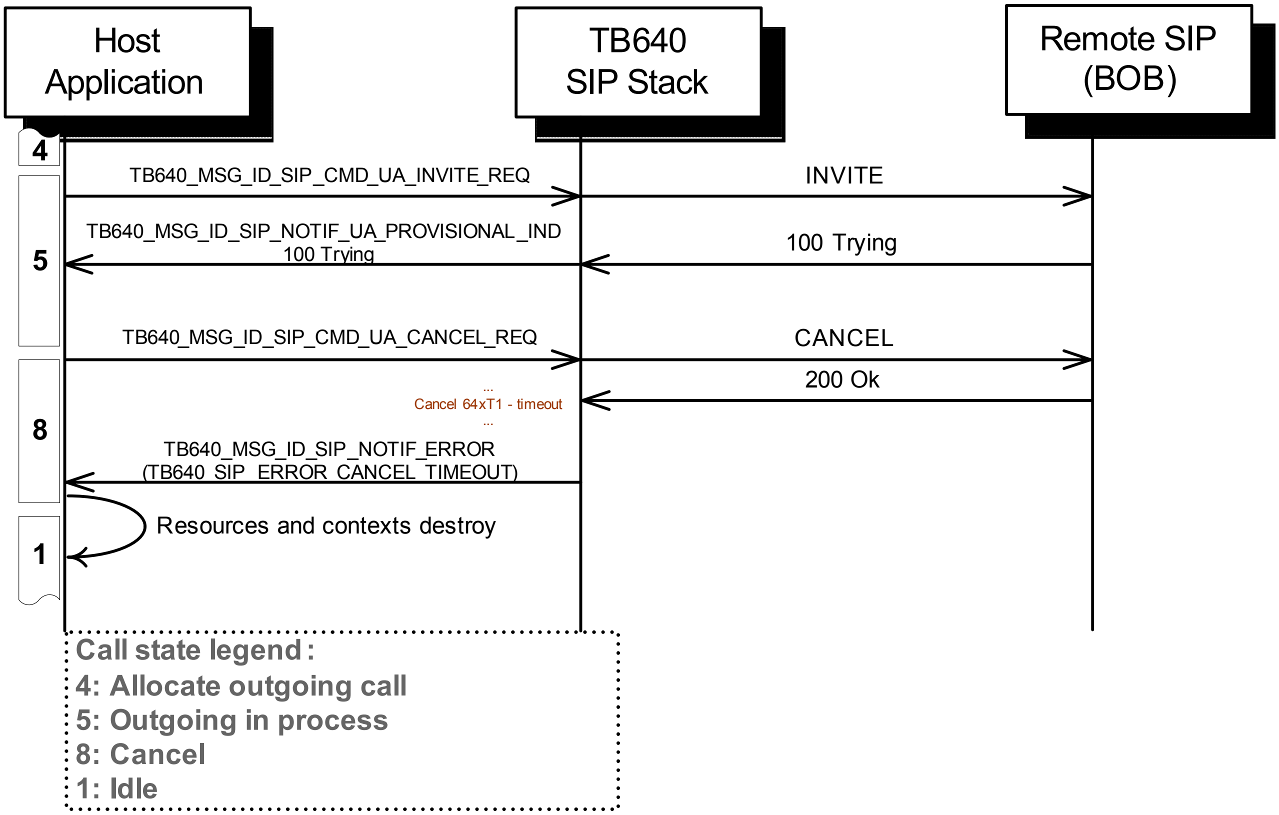


Figure 15 - Outgoing call flow with host Cancel timeout

In this call flow, the local host initiates an INVITE request and after a little while the customer application sends a CANCEL request toward the TB640 SIP stack. After 64T1, if no 487 response has been received by the stack, a TB640_MSG_ID_SIP_NOTIF_ERROR is generated by the stack with an error code TB640_SIP_ERROR_CANCEL_TIMEOUT.

6.5.3.8 Call flow with host Cancel cross over

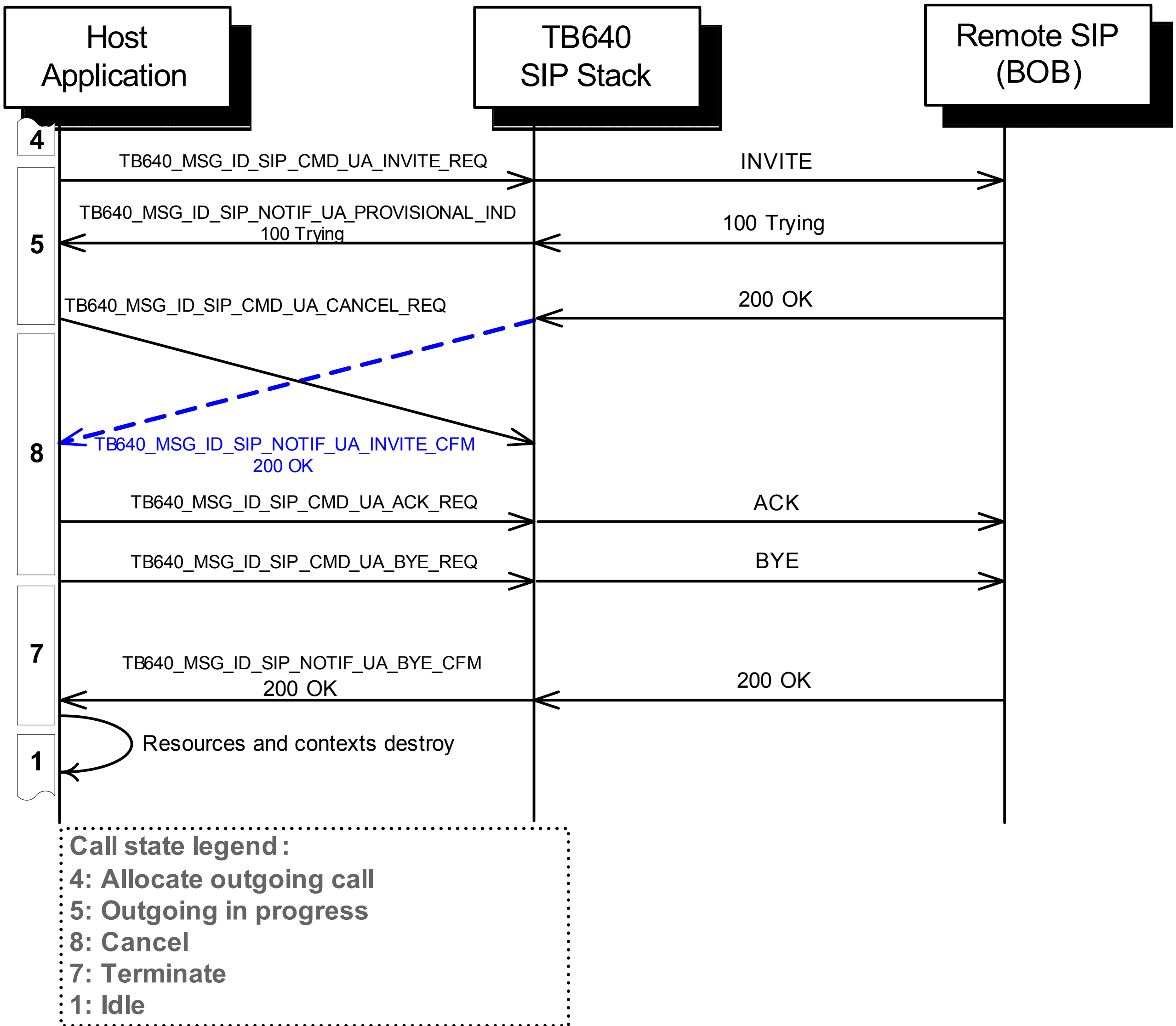


Figure 16 - Outgoing call flow with host Cancel cross over

In this call flow, the local host initiates an INVITE request and after a little while the customer application sends a CANCEL request toward the TB640 SIP stack. Meanwhile, the TB640 SIP stack receives the INVITE response from the remote SIP host (BOB). The CANCEL and INVITE message cross over the networks. Is such a situation, the host application must acknowledge the INVITE response and generate a BYE requests.

6.5.3.9 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

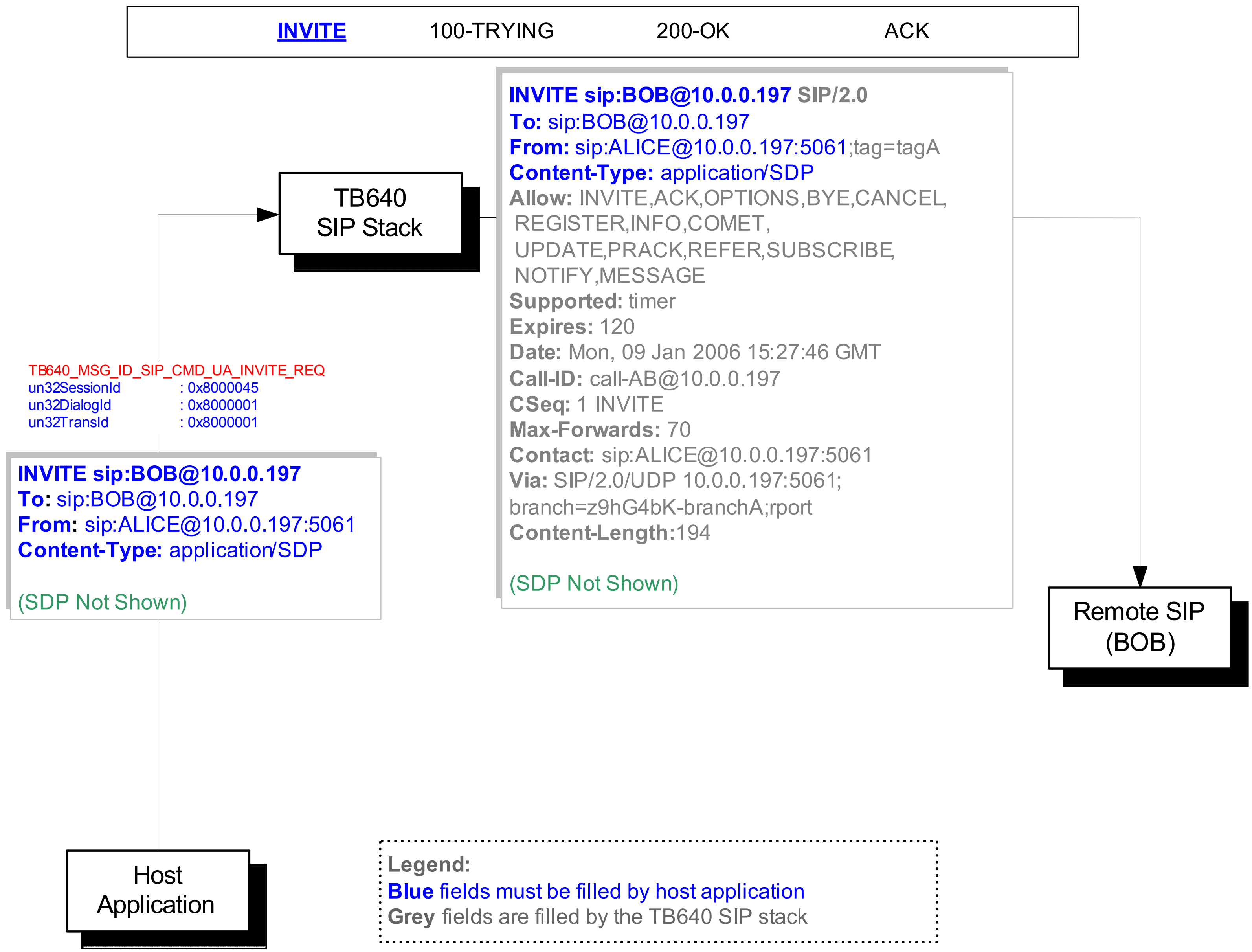


Figure 17 - INVITE Request

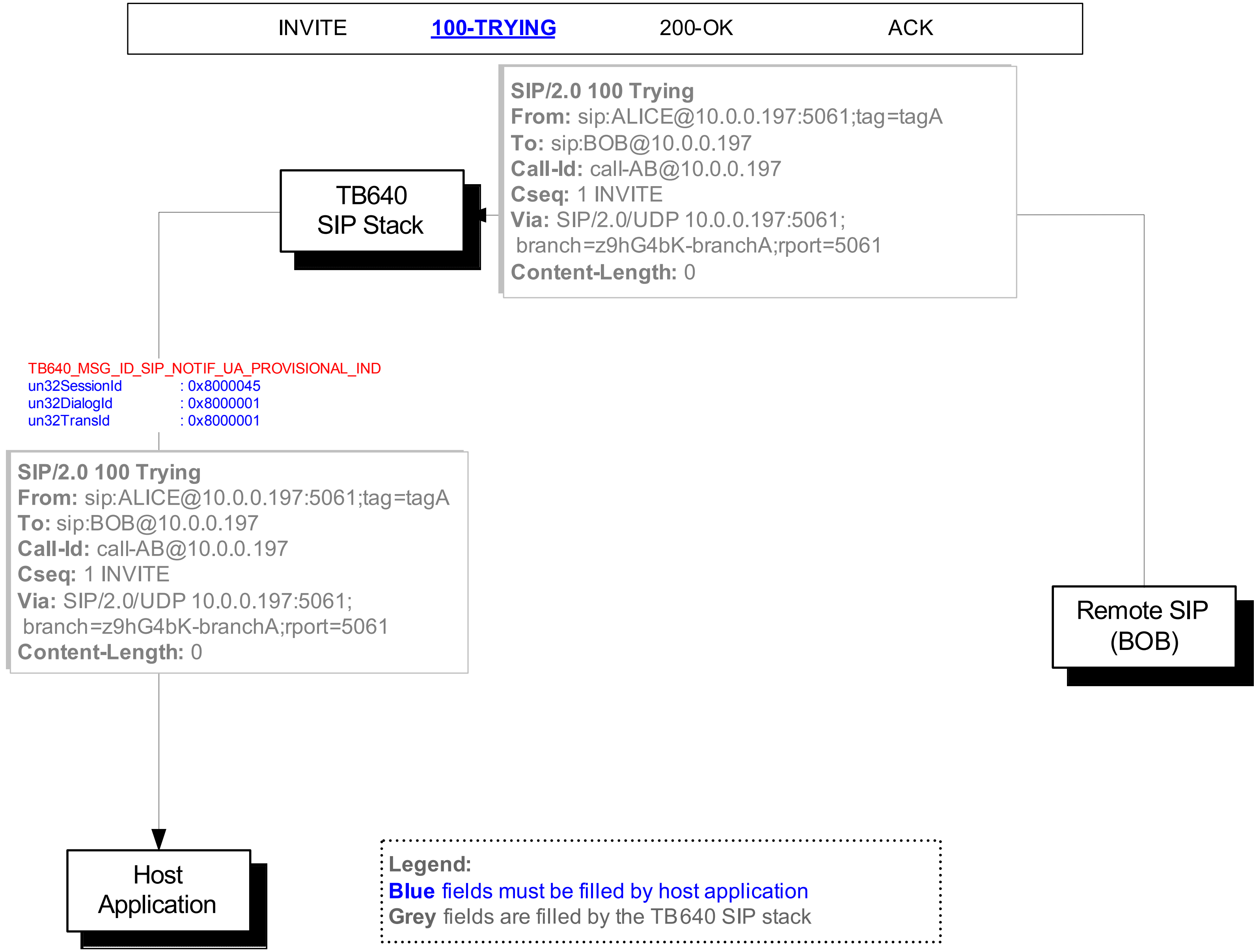


Figure 18 - PROVISIAL Response – Status 100 Trying

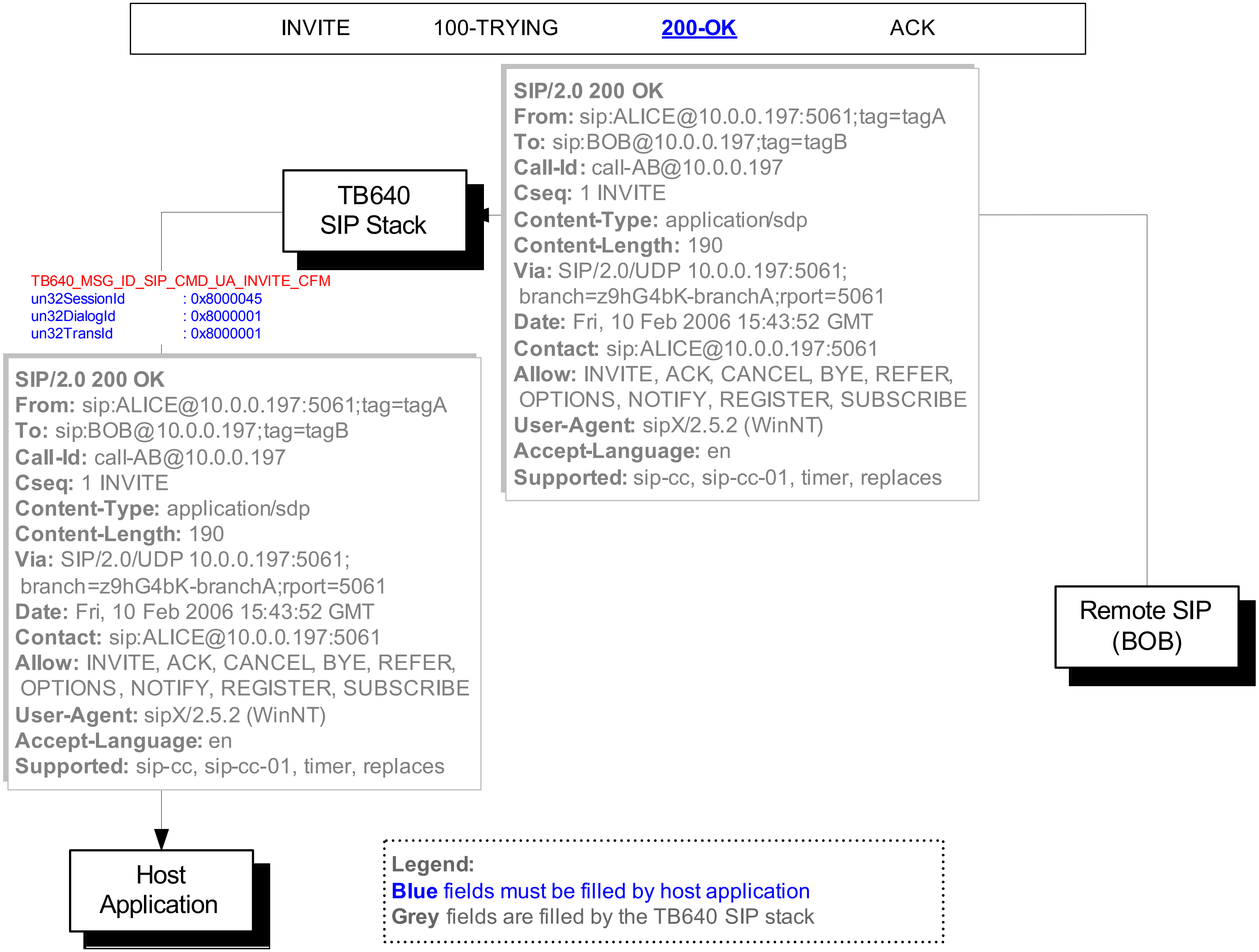


Figure 19 - INVITE Response – Status 200 Ok

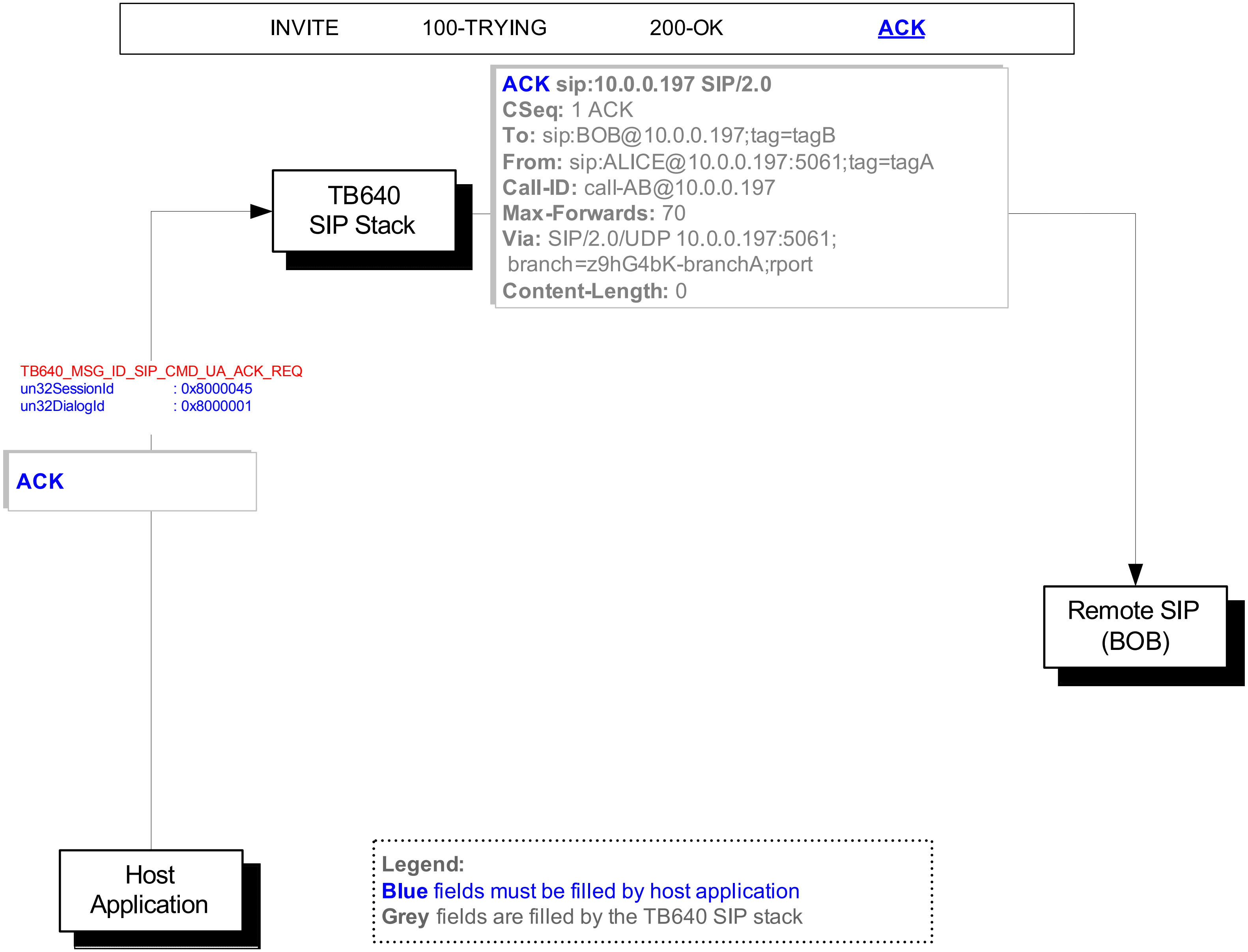


Figure 20 - ACK Request

6.5.4 Receiving a call

This section will go thru the establishment of an incoming call in details. Packet losses scenarios are covered in order to understand the complete call flow. Then, each messages of the call flow will be detailed in order to see SIP Message Header offloading by the TB640 SIP stack.

6.5.4.1 Call flow with packet losses

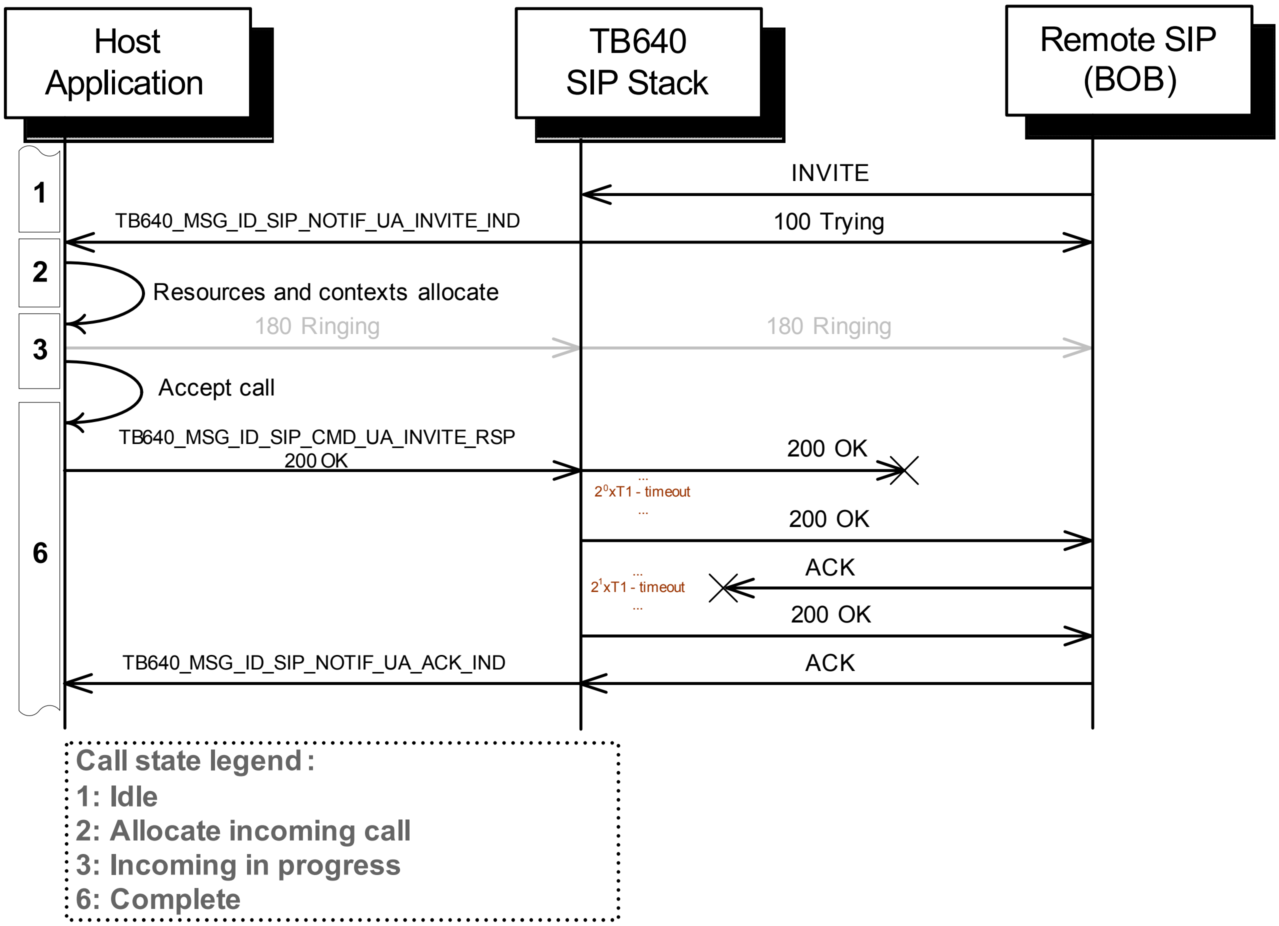


Figure 21 - Incoming call flow with packet losses

In this call flow, the Remote host (BOB) initiates an INVITE request. The TB640 automatically sends a PROVISIONAL Status 100-Trying to the Remote host (as configured by *fAlwaysSend100* in section 6.2.1.2 UA entity configuration). The local host accepts the call and issues a 200-Ok Response to the SIP stack on the TB640. The TB640 fills missing SIP Headers and then forwards the 200 Ok to the Remote Host. Unluckily, the INVITE response is lost in between the TB640 and the Remote SIP. After T1 seconds, the INVITE response gets retransmitted by TB640 SIP stack. The same is true if the ACK Request is lost in between the Remote SIP and the TB640.

6.5.4.2 Call flow with error timeout

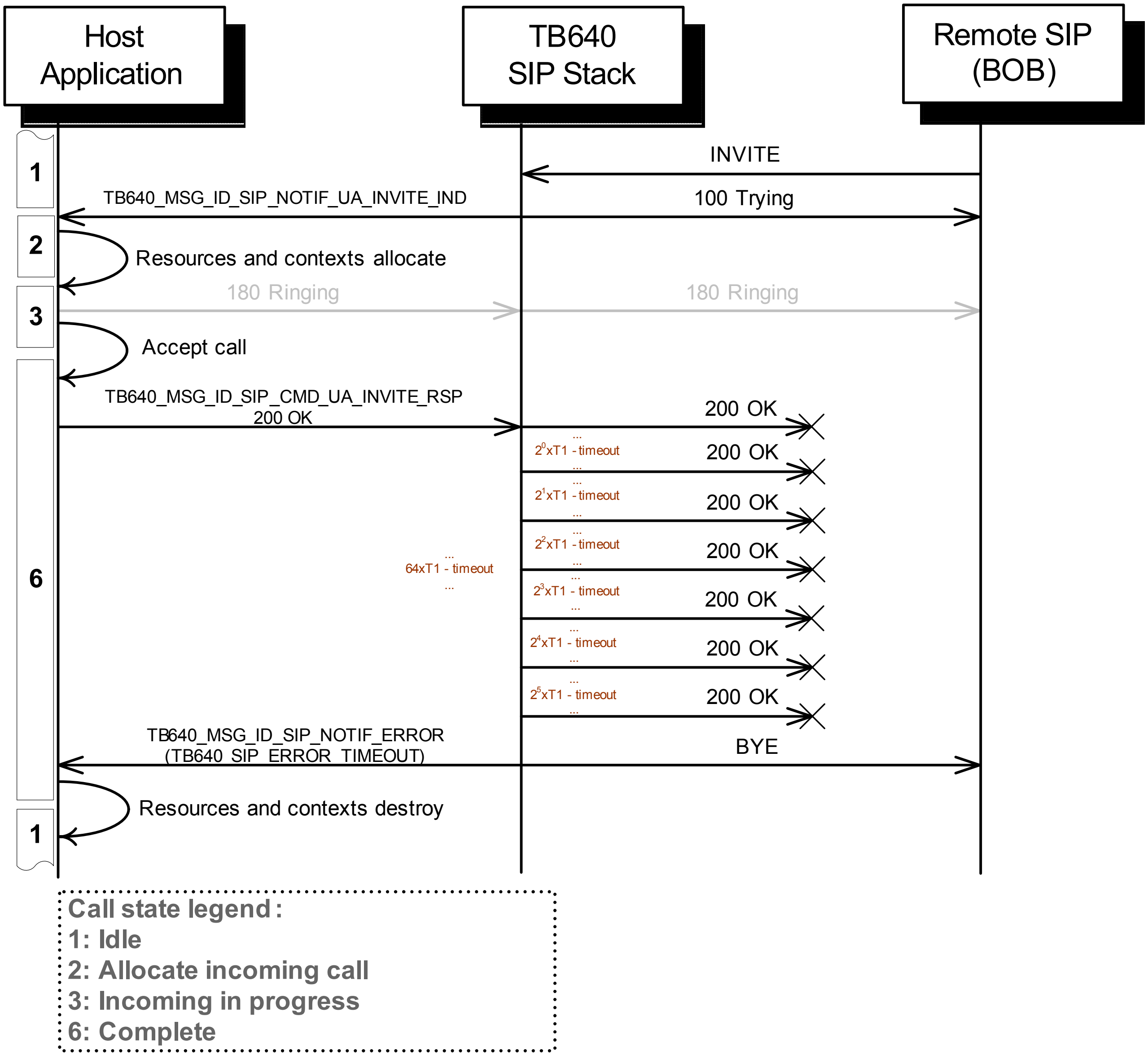


Figure 22 - Incoming call flow with error timeout

In this call flow, the Remote host (BOB) initiates an INVITE request. The TB640 automatically sends a PROVISIONAL Status 100-Trying to the Remote host (as configured by *fAlwaysSend100* in section 6.2.1.2 UA entity configuration). The local host accepts the call and issues a 200-Ok Response to the SIP stack on the TB640. The TB640 fills missing SIP Headers and then forwards the 200 Ok to the Remote Host. Unluckily, all the INVITE response re-transmission never reaches the Remote SIP host (BOB). After 64T1 seconds, the TB640 SIP stack send a CANCEL and generates a TB640_MSG_ID_SIP_NOTIF_ERROR with an error code TB640_SIP_ERROR_TIMEOUT.

6.5.4.3 Call flow with remote host Cancel

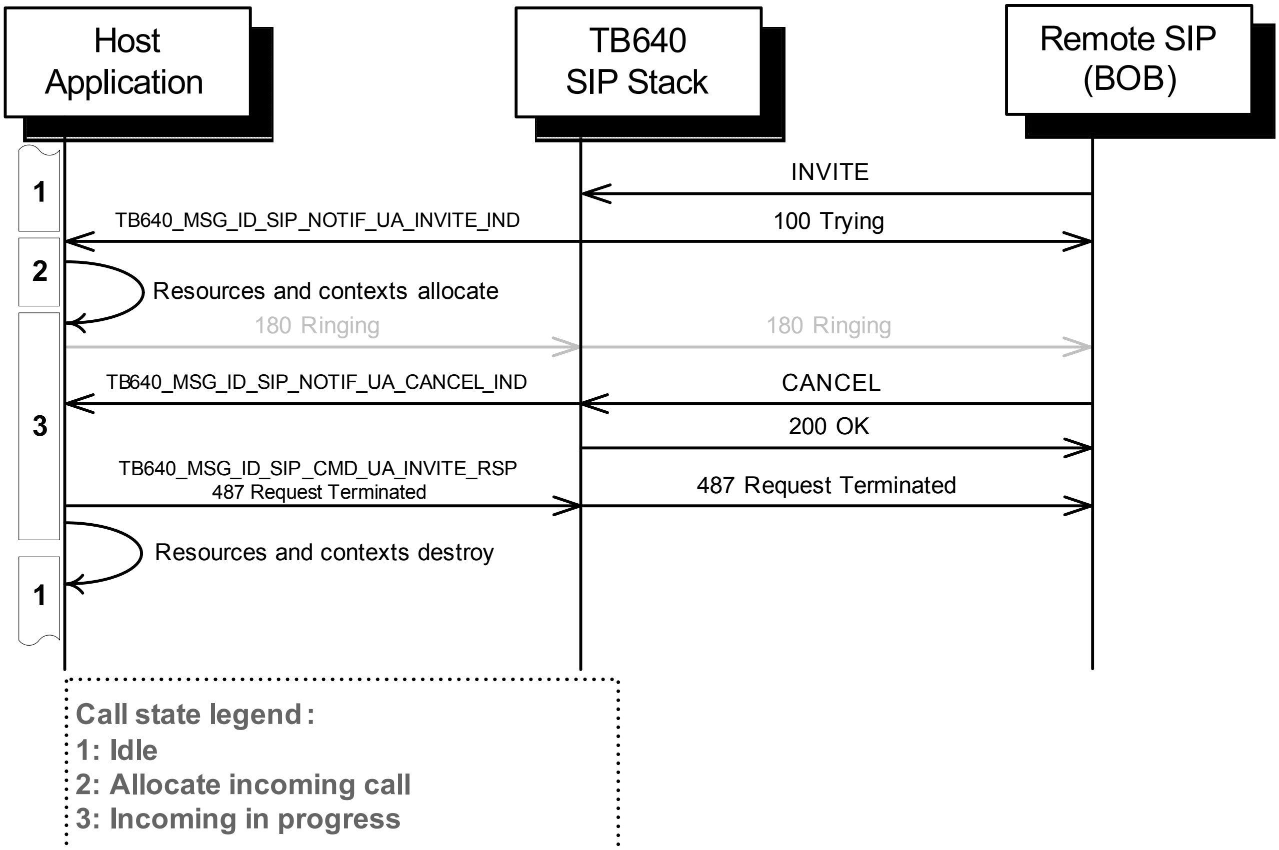


Figure 23 - Incoming call flow with remote host Cancel

In this call flow, the Remote host (BOB) initiates an INVITE request. The TB640 automatically sends a PROVISIONAL Status 100-Trying to the Remote host (as configured by *fAlwaysSend100* in section 6.2.1.2 UA entity configuration). The local host application receives a CANCEL indication and destroys all its resources and context associated to that received call. The TB640 stack generates a CANCEL OK response. The host application must generate an INVITE 487 response.

6.5.4.4 Call flow with remote host Cancel cross over

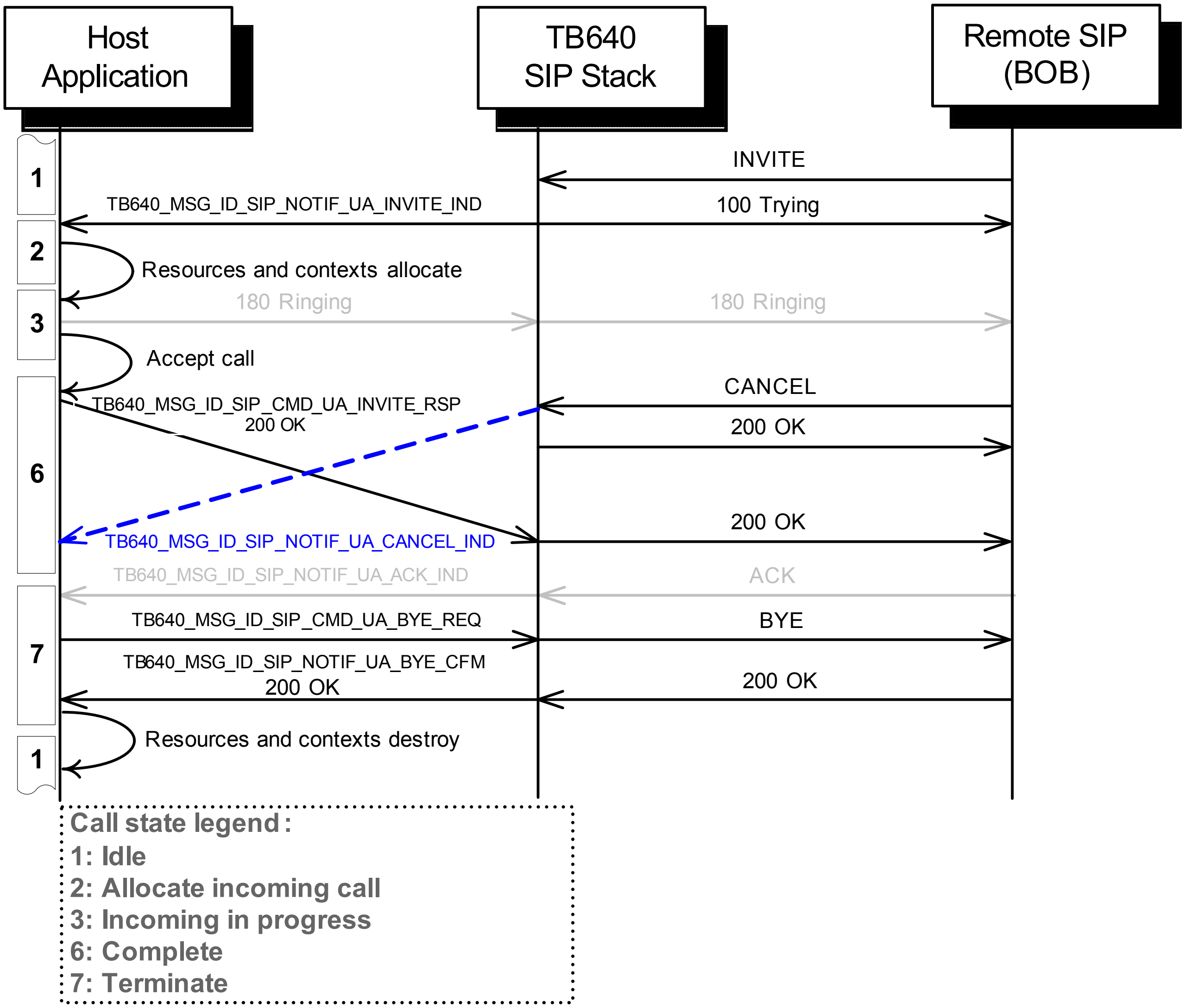


Figure 24 - Incoming call flow with remote host Cancel

In this call flow, the Remote host (BOB) initiates an INVITE request. The TB640 automatically sends a PROVISIONAL Status 100-Trying to the Remote host (as configured by *fAlwaysSend100* in section 6.2.1.2 UA entity configuration). The local host application receives a CANCEL indication and destroys all its resources and context associated to that received call. The CANCEL OK response is generated by the TB640 SIP stack. Meanwhile, the host application sends a 200 OK response to the TB640 and both messages cross over the networks. The host application

6.5.4.5 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

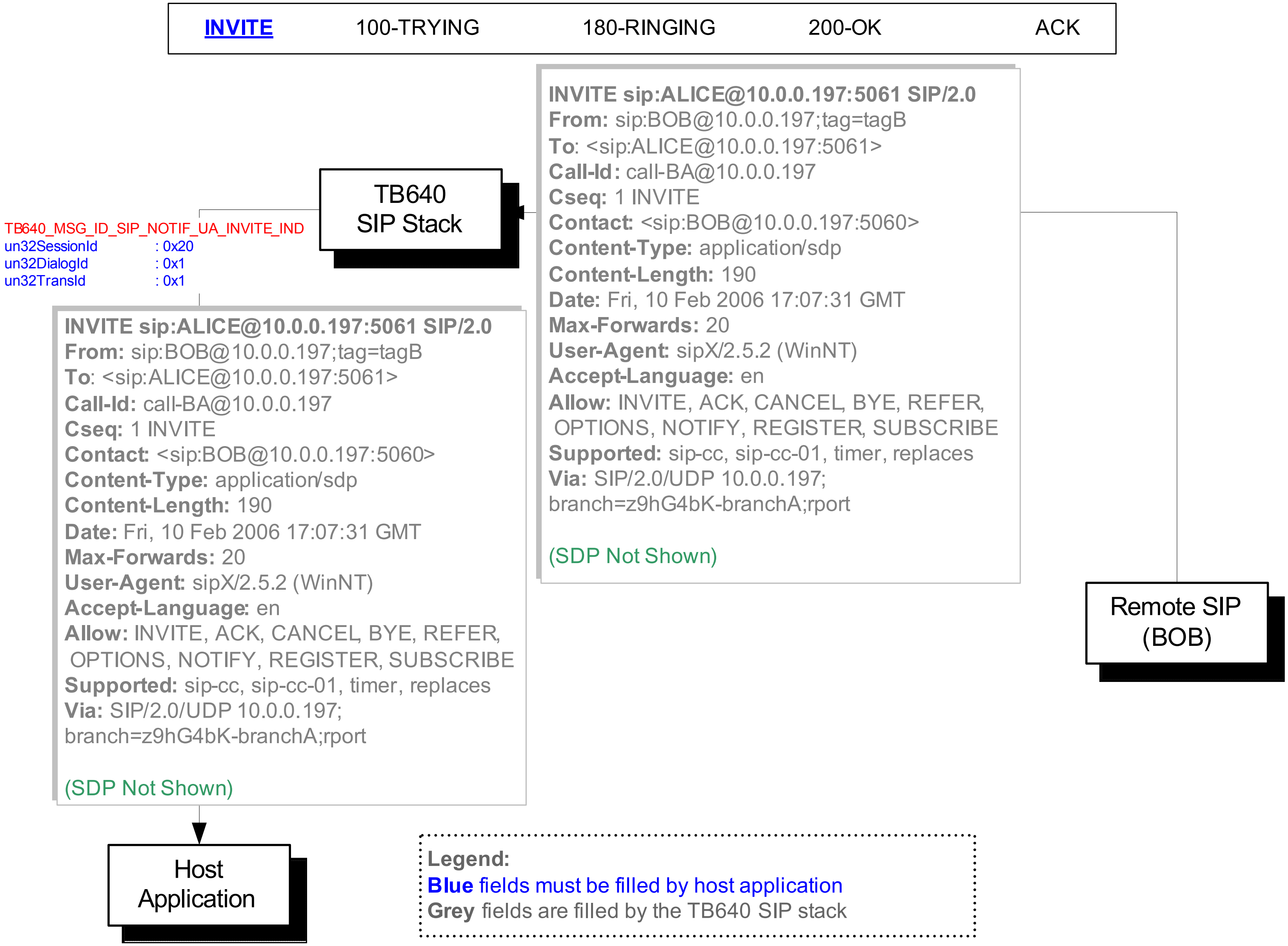


Figure 25 - INVITE Indication

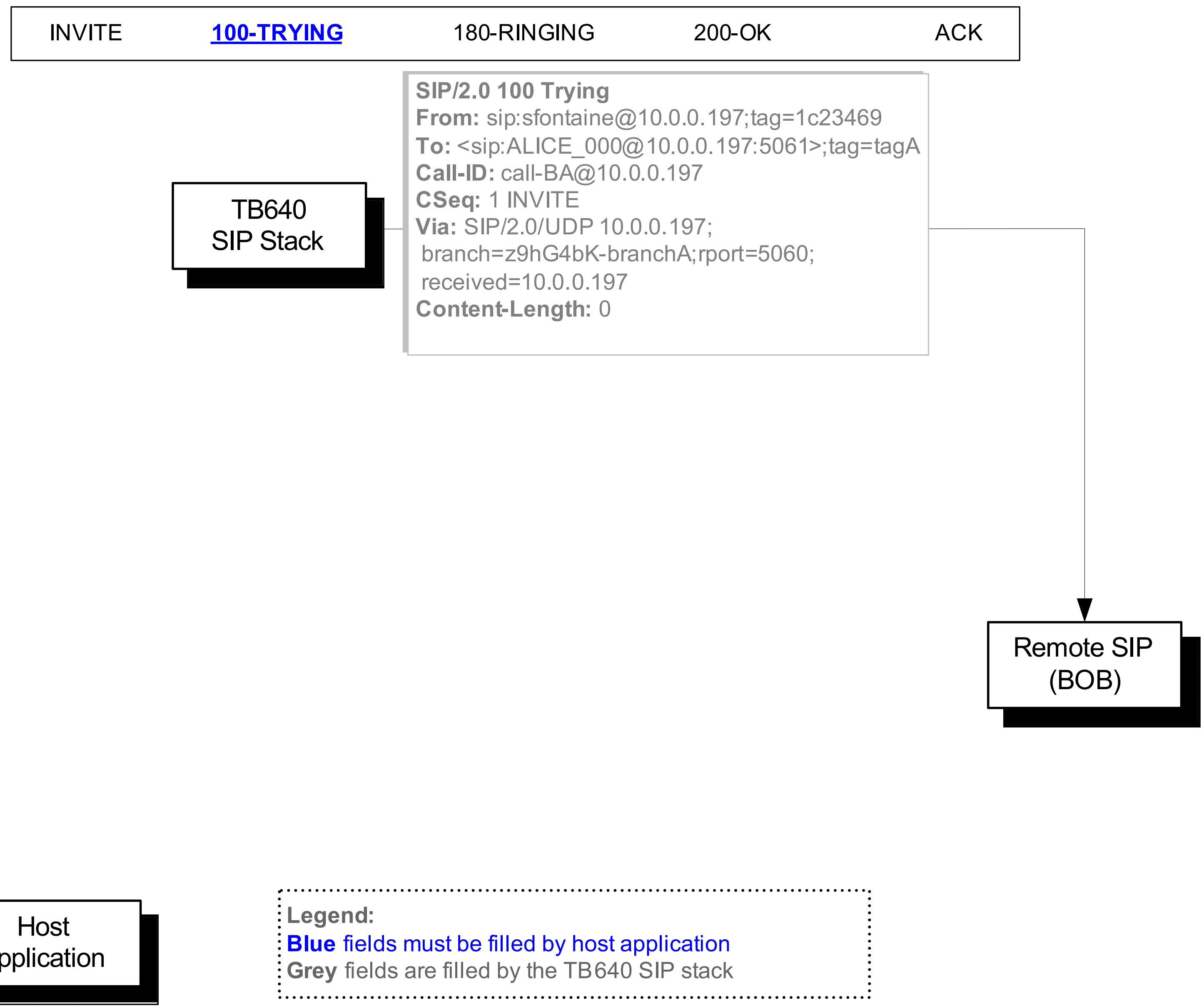


Figure 26 - PROVISIONAL Response – Status 100 Trying

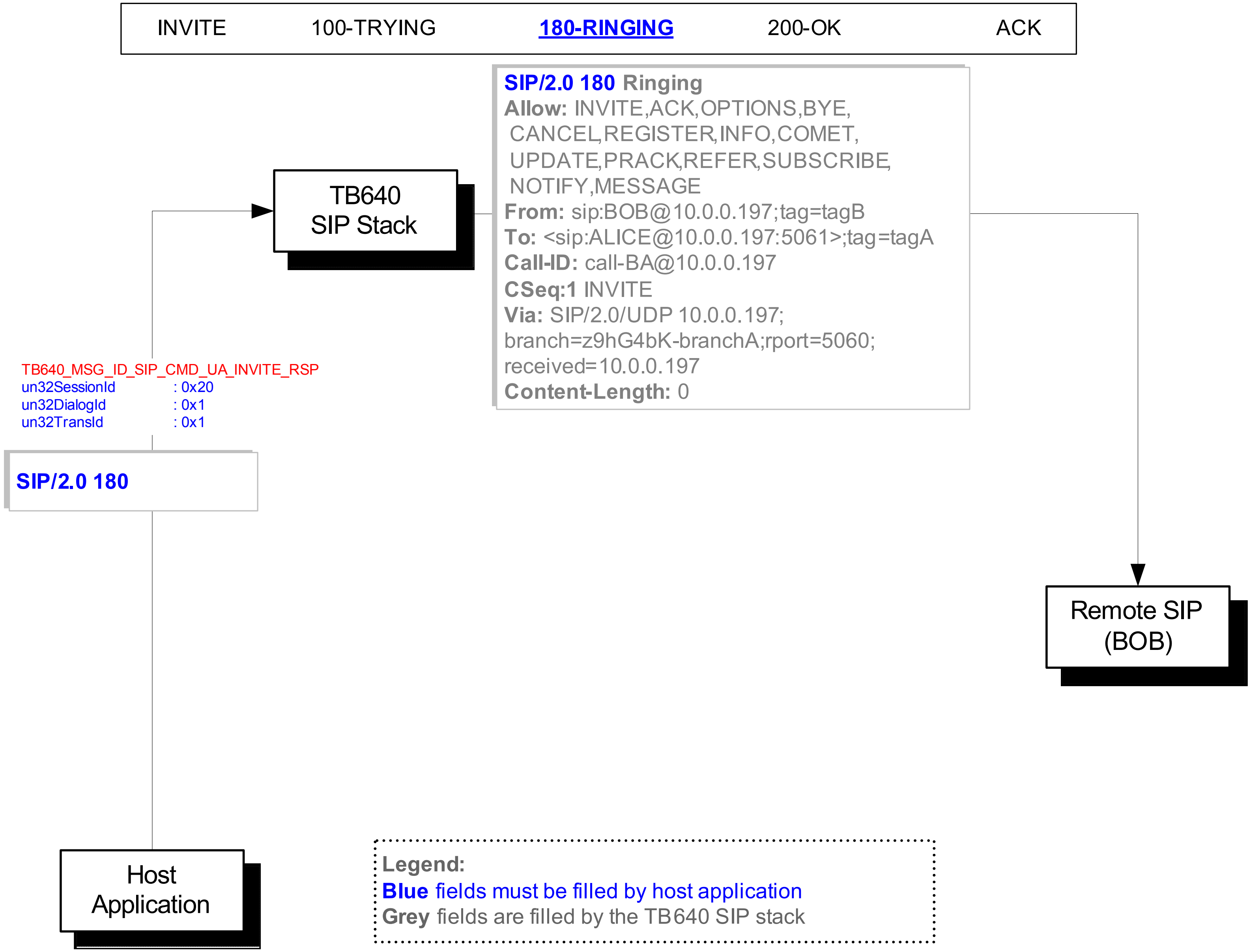


Figure 27 - PROVISIONAL Response – Status 180 Ringing

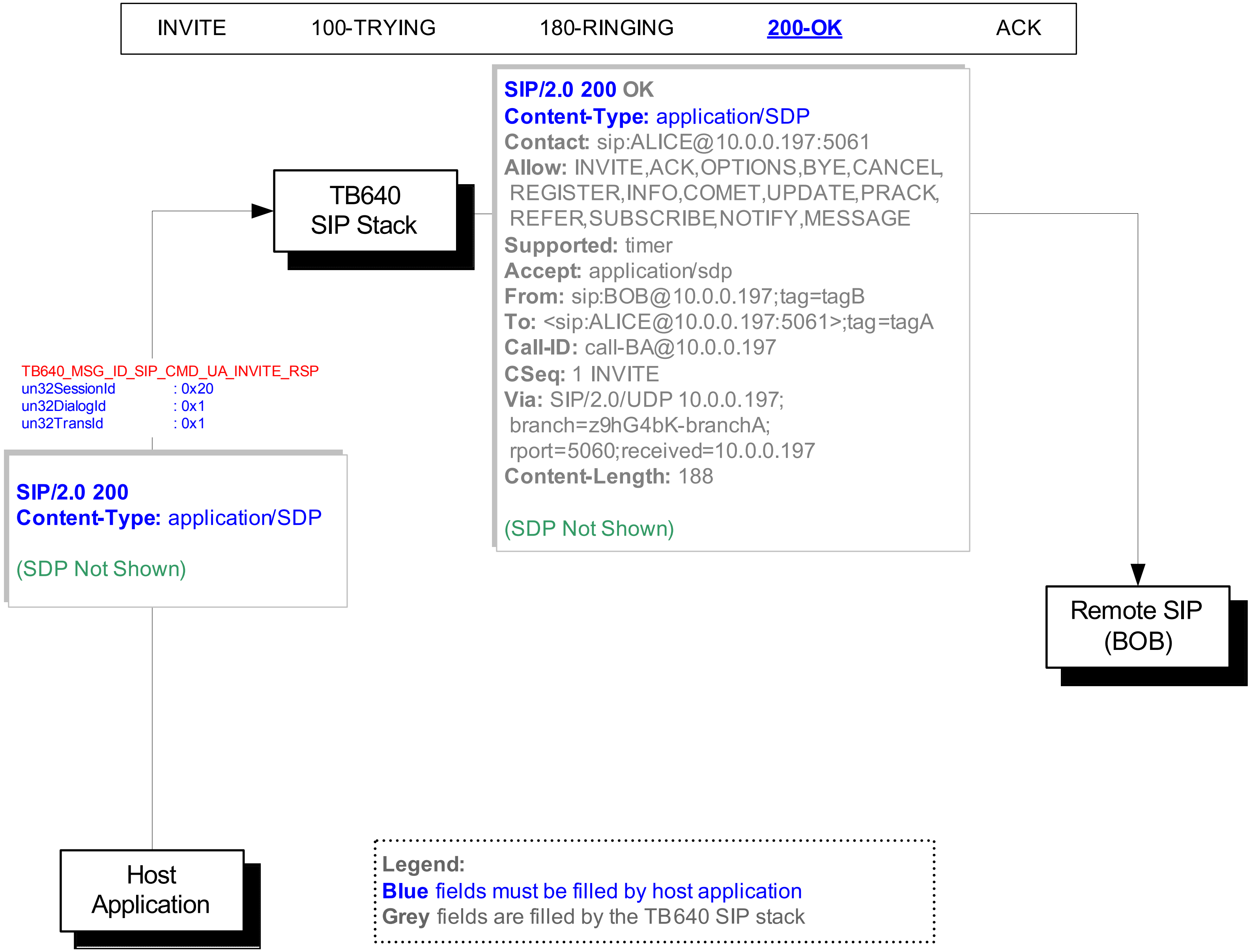


Figure 28 - INVITE Response – Status 200 Ok

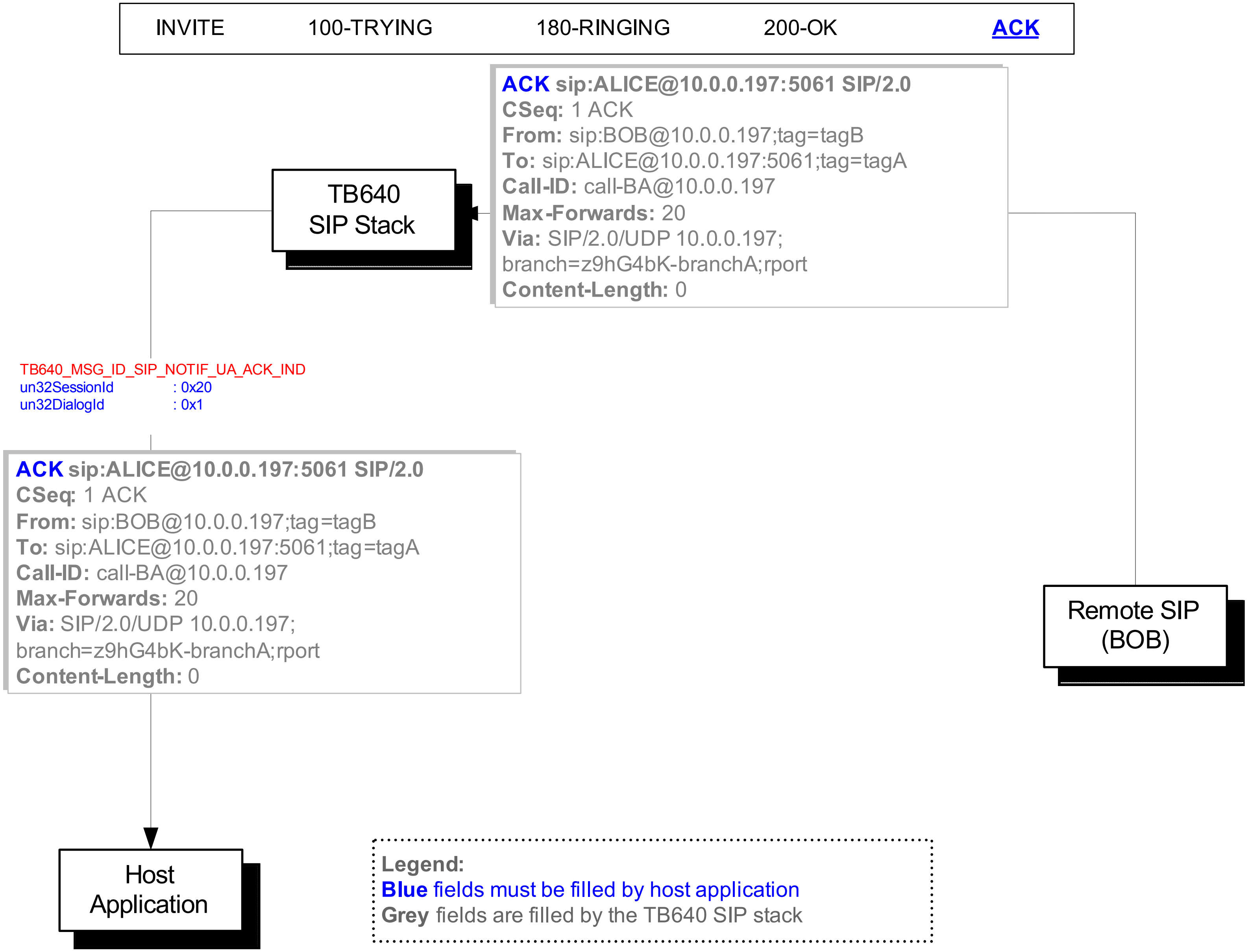


Figure 29 - ACK Indication

6.5.5 Requesting to release a call

This section will go through releasing of a call in details. Packet losses scenarios are covered in order to understand the complete call flow. Then, each messages of the call flow will be detailed in order to see the SIP Message Header offloading by the TB640 SIP stack.

6.5.5.1 Normal call flow

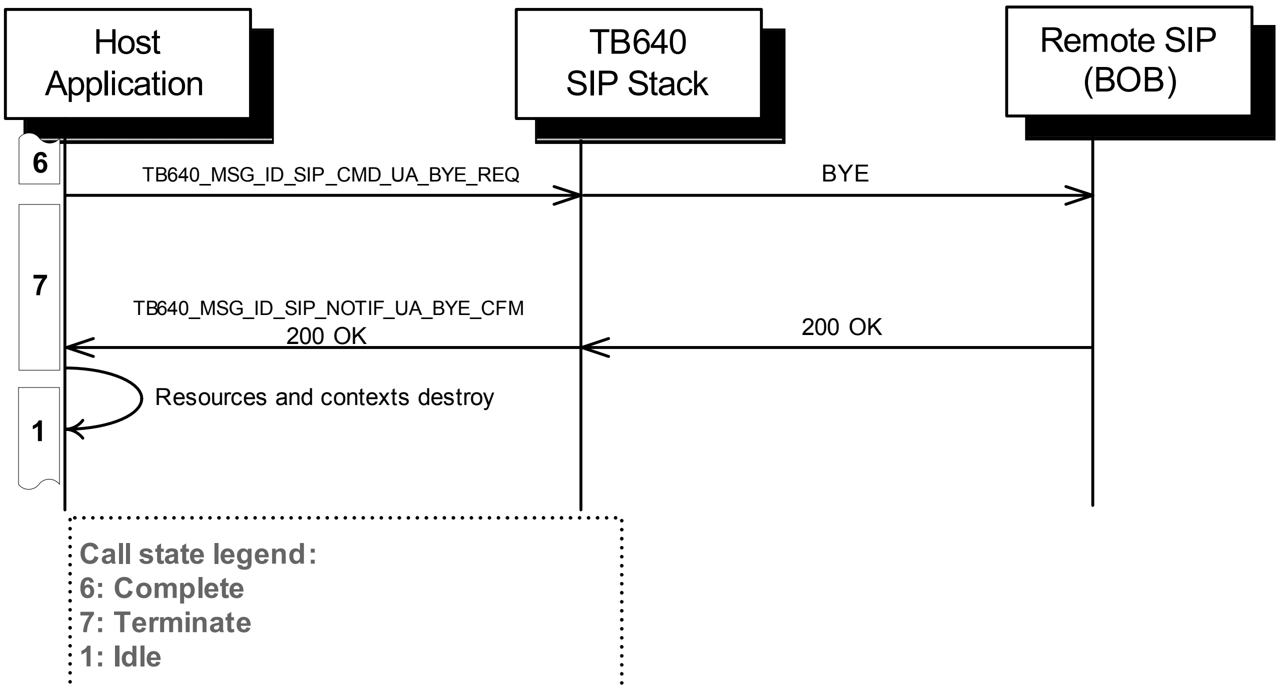


Figure 30 – Release call flow

In this call flow, the local host initiates a BYE request. Once it received the 200 OK response from the Remote SIP host (BOB), it destroy the corresponding call resources and context.

6.5.5.2 Call flow with packet losses

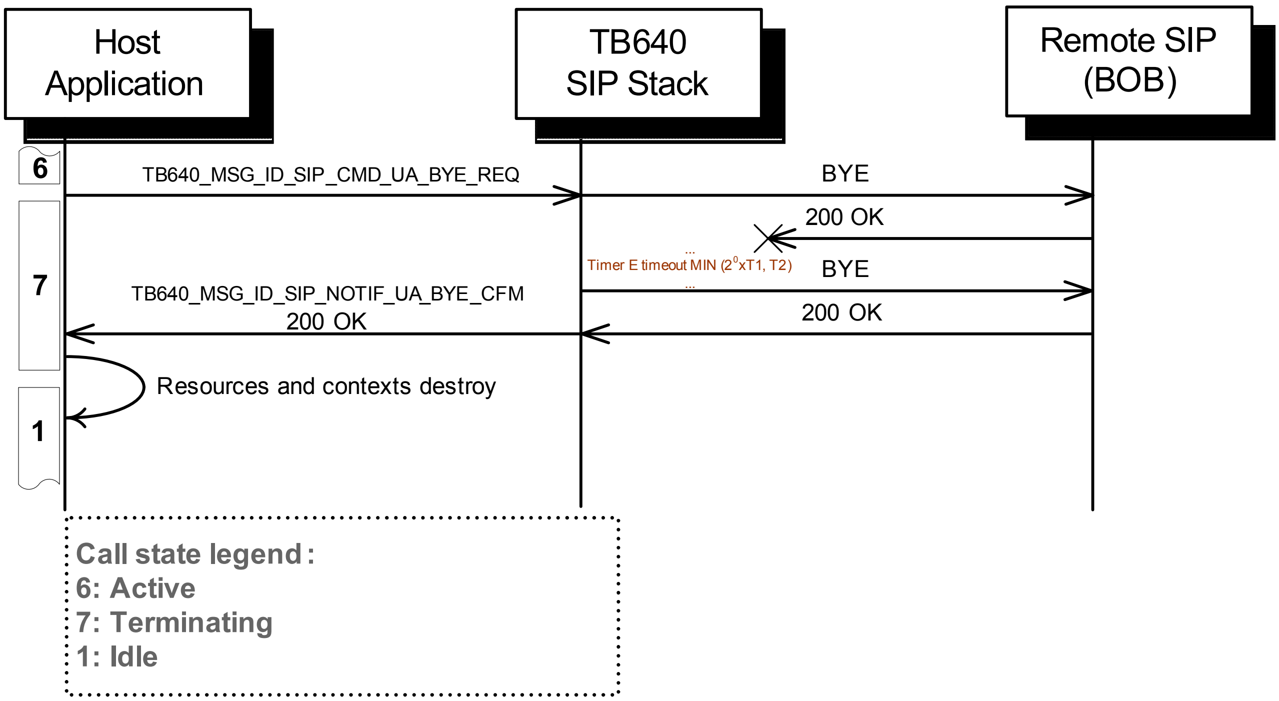


Figure 31 - Release call flow with packet losses

In this call flow, the local host initiates a BYE request. If the BYE response from the Remote SIP is lost, the TB640 retransmits the BYE request. In the current scenario, the Remote SIP did receive the BYE request, and is not aware that its response was lost. After T1 (Timer E), the BYE request is retransmitted by the TB640 SIP stack until it receives the BYE response from the remote SIP host (BOB).

6.5.5.3 Call flow with error sending

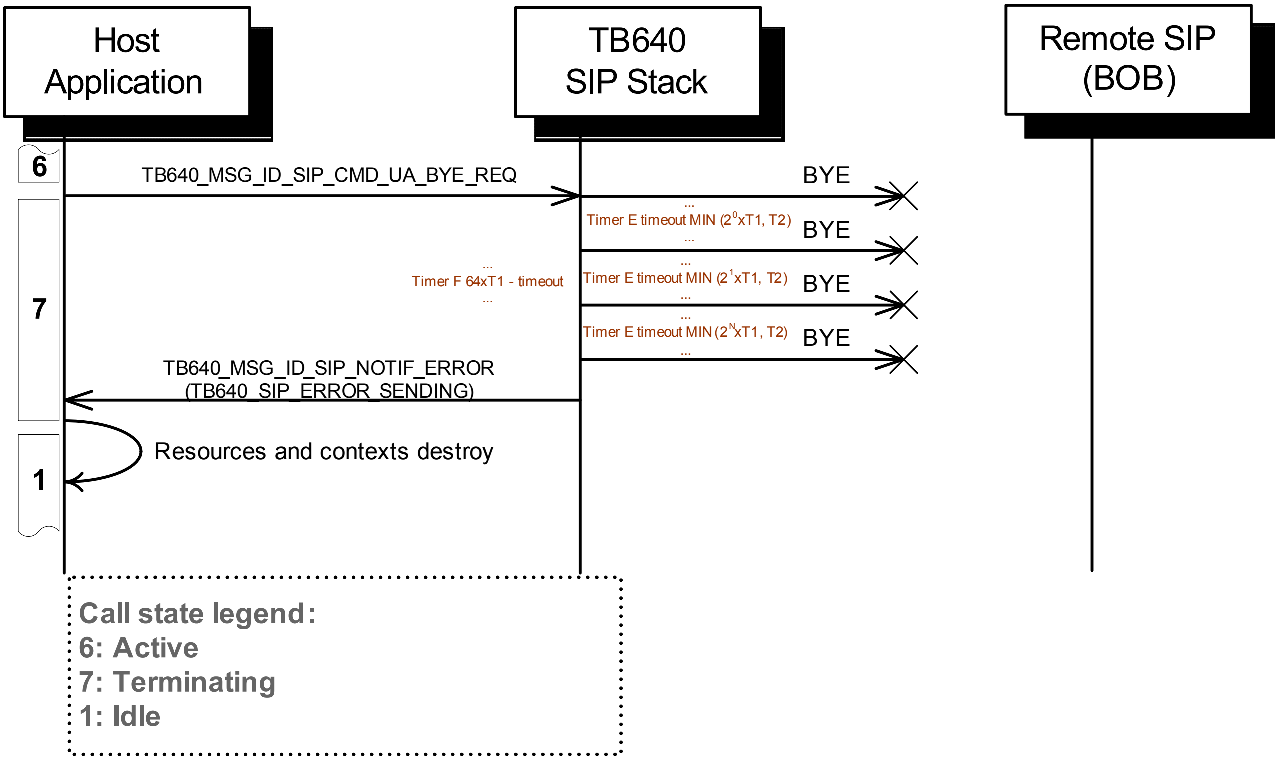


Figure 32 – Release call flow with error sending

In this call flow, the local host initiates a BYE request. After T1 milliseconds, the Timer E of the BYE transaction expire and the request gets retransmitted by the TB640 SIP stack without the host having to take any action. Each time the timer E expired, its value is multiplied by 2 until its value reach T2, then timer T2 is used. After 64T1 (timer F), if no response has been received by the stack, a TB640_MSG_ID_SIP_NOTIF_ERROR is generated by the stack with an error code TB640_SIP_ERROR_SENDING.

6.5.5.4 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

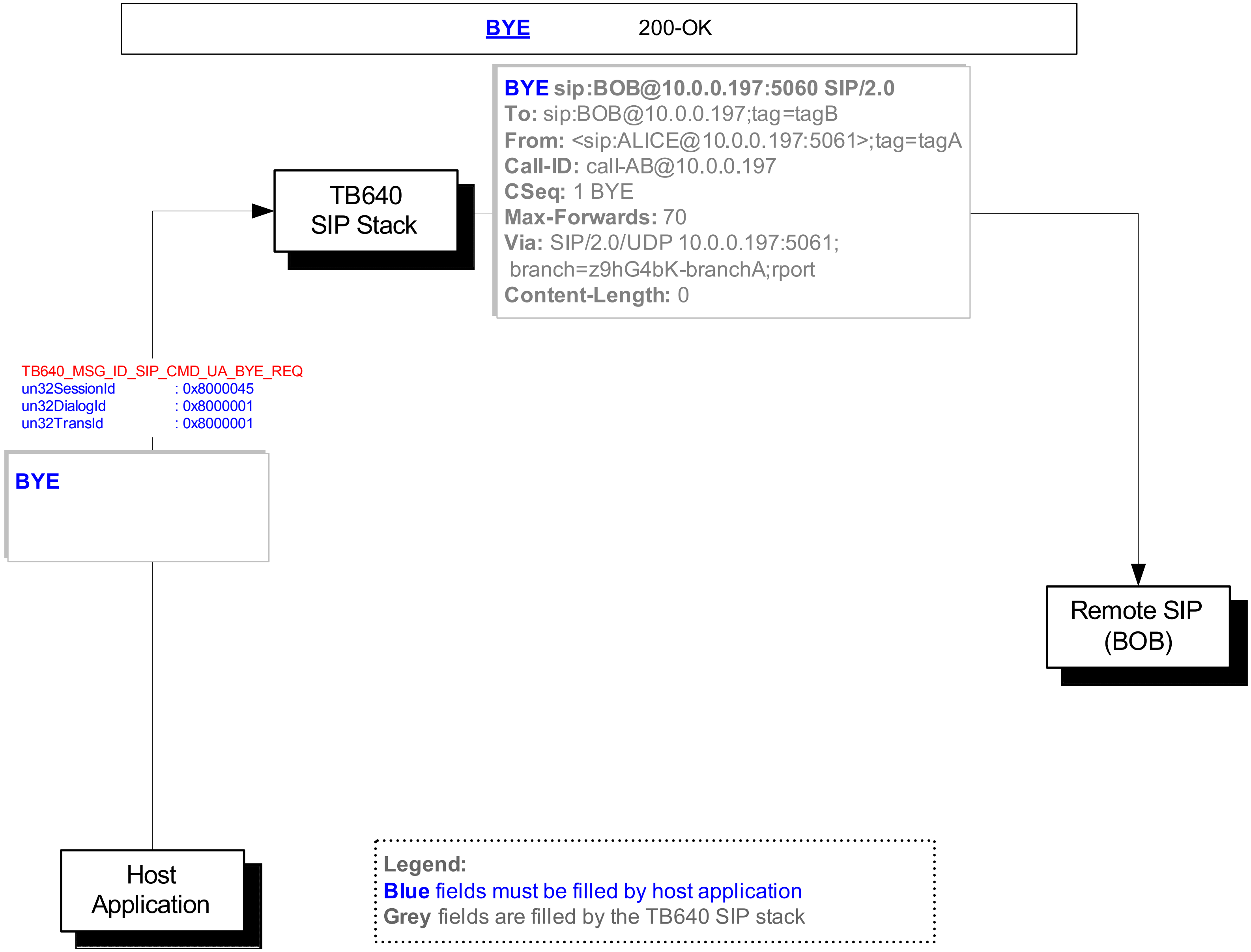


Figure 33 - BYE request

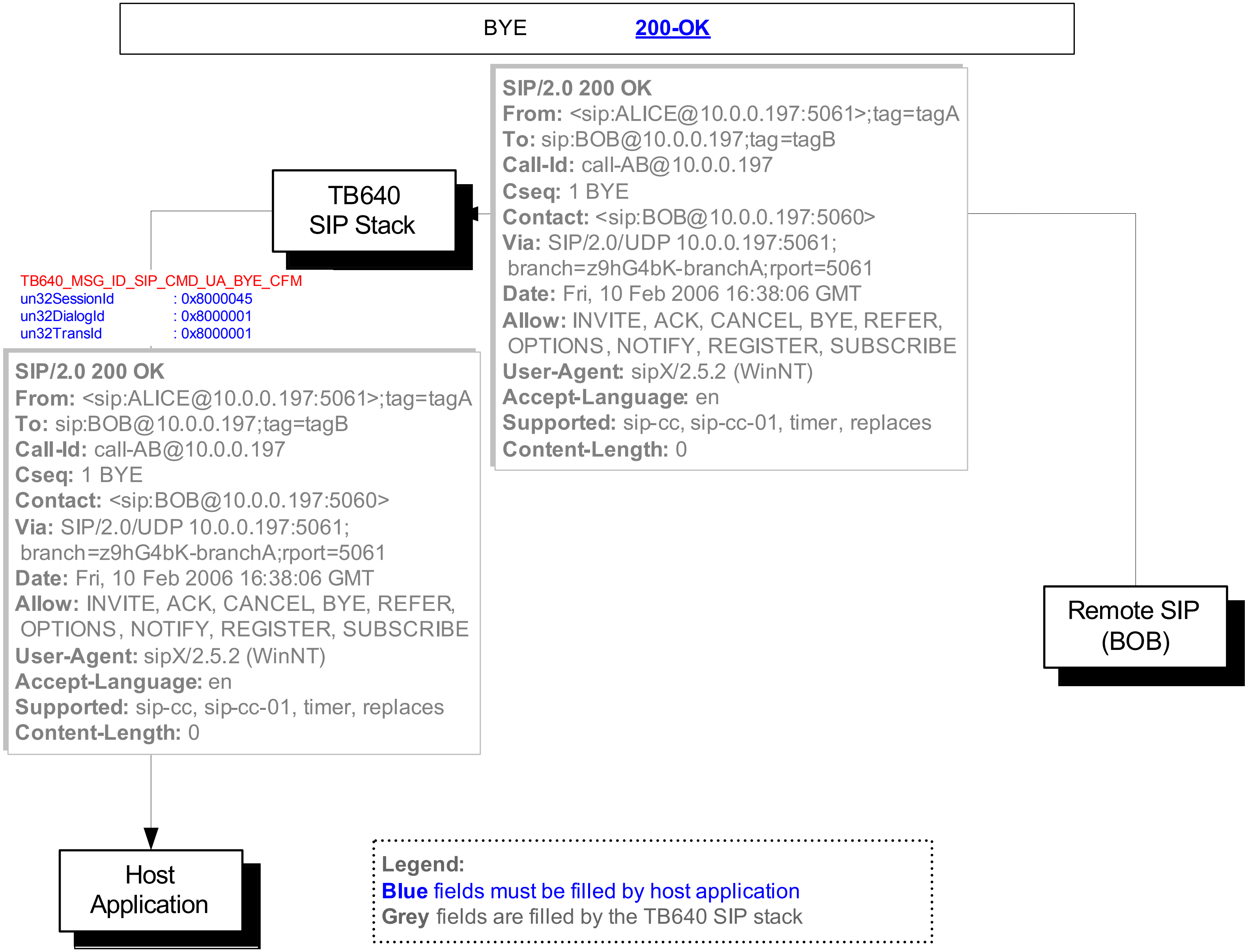


Figure 34 - BYE response – Status 200 Ok

6.5.6 Remote release of a Call

This section will go through releasing of a call in details. Packet losses scenarios are covered in order to understand the complete call flow. Then, each messages of the call flow will be detailed in order to see the SIP Message Header offloading by the TB640 SIP stack.

6.5.6.1 Call flow with packet losses

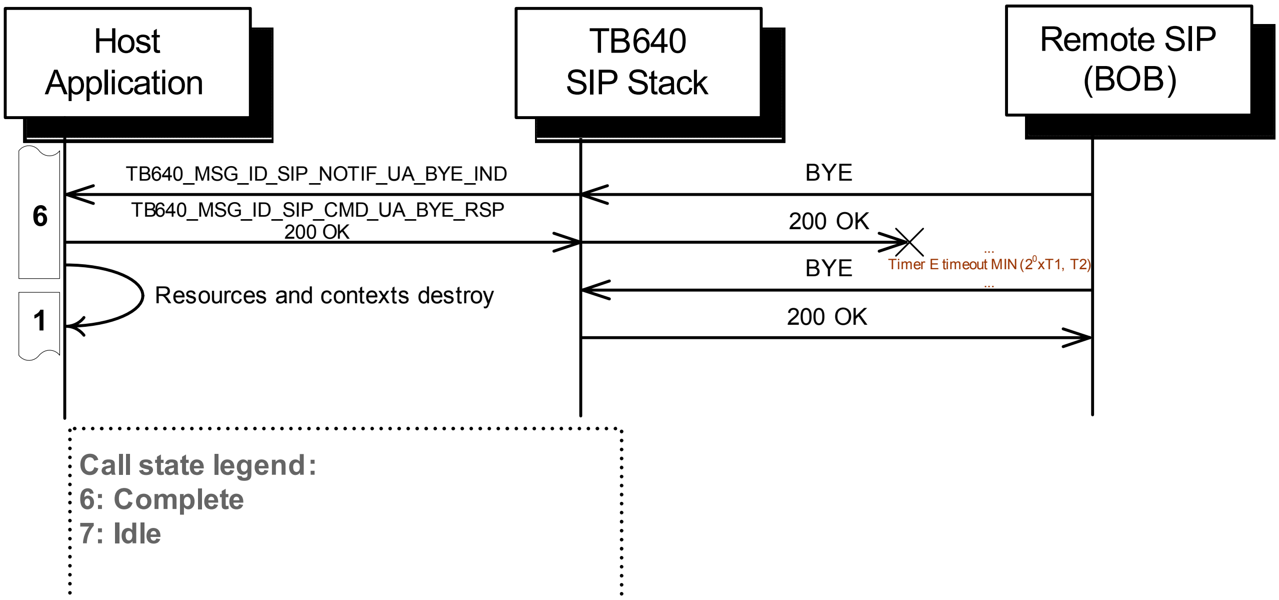


Figure 35 – Remote release call flow with packet losses

In this call flow, the remote host initiates a BYE request. The TB640 and local host did receive the BYE request, but the response is lost in between the TB640 and the Remote Sip stack. After T1 (Timer E), the BYE request is retransmitted by the remote SIP host (BOB) until it receives the BYE response from the TB640 SIP stack.

6.5.6.2 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

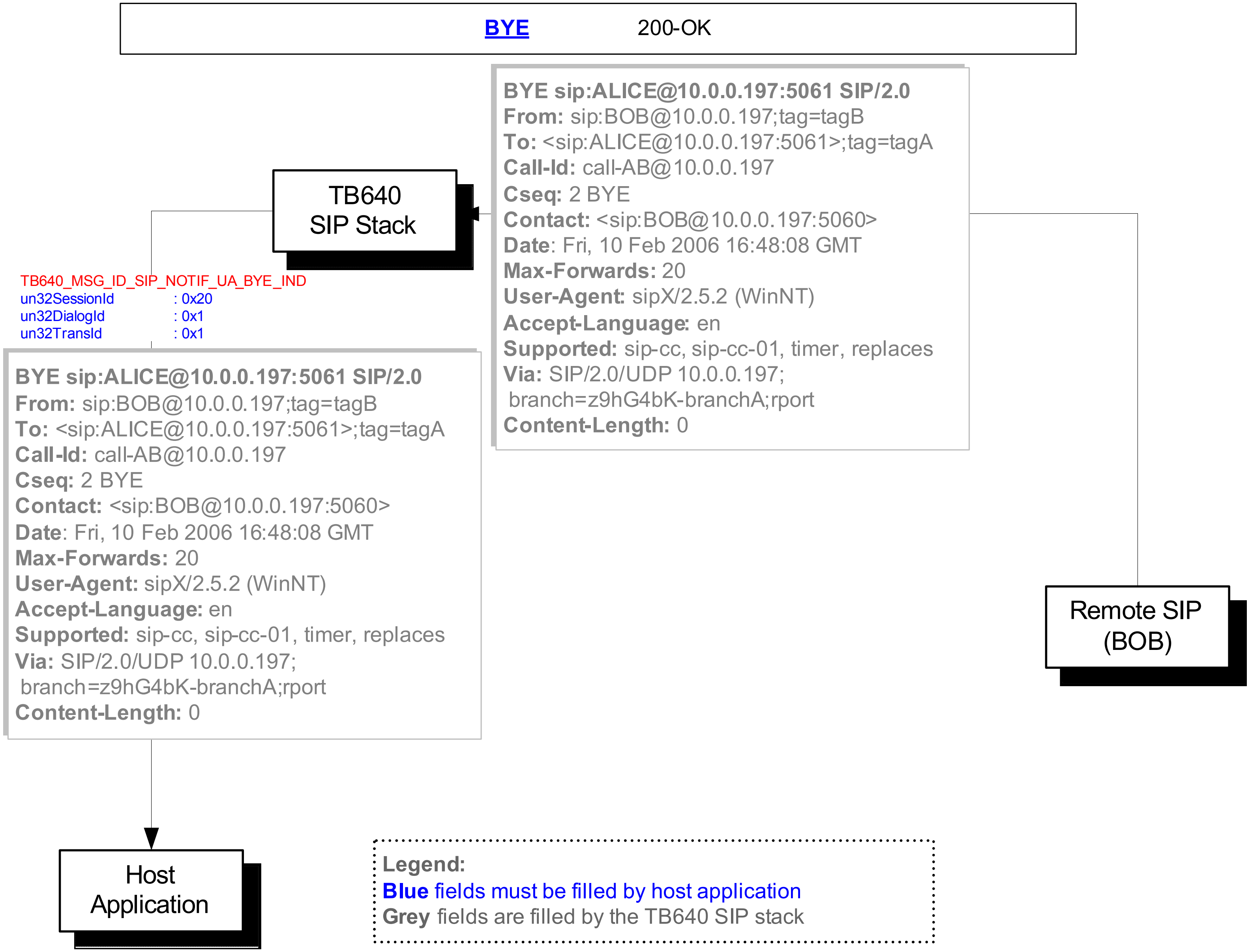


Figure 36 - BYE Indication

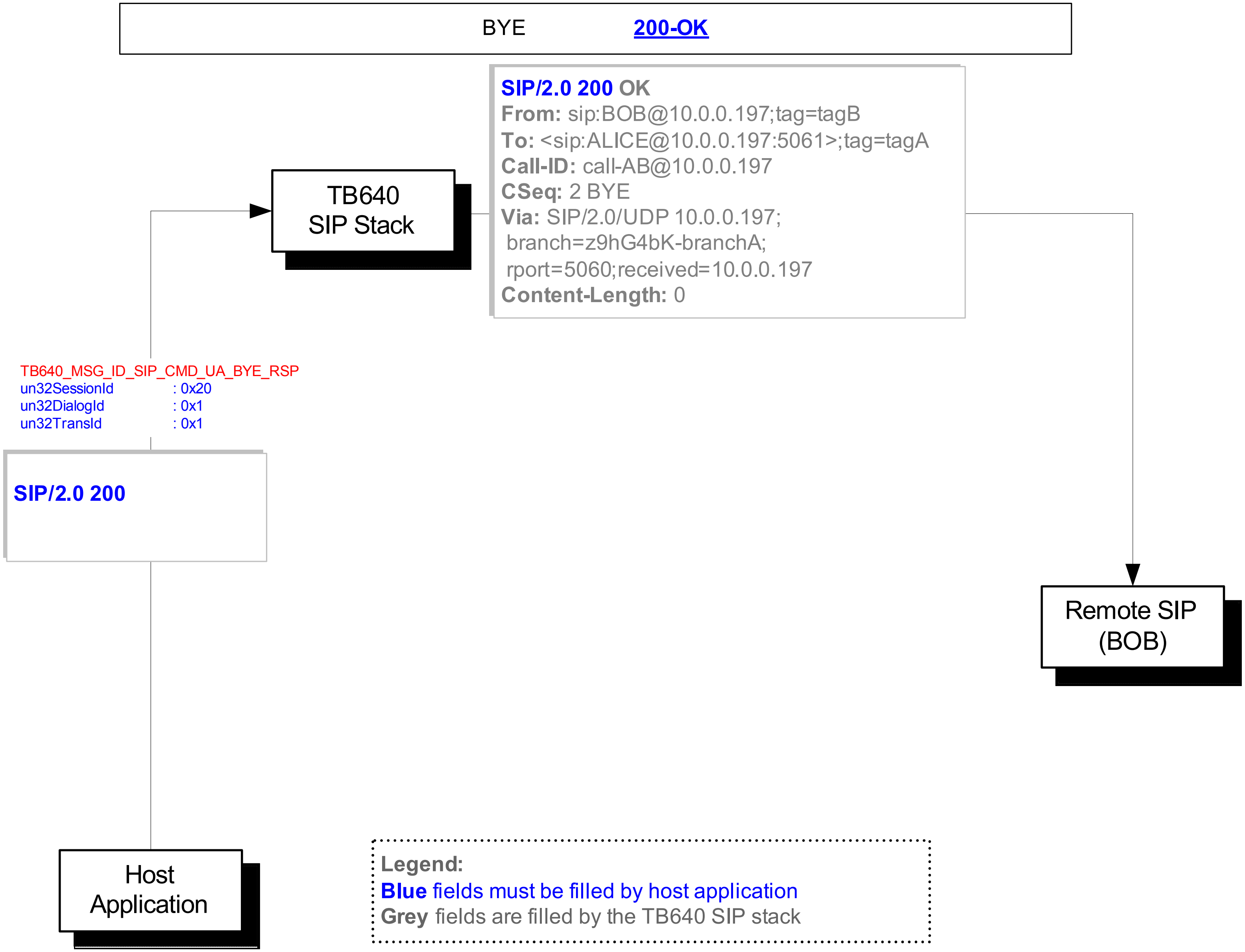


Figure 37 - BYE Response – Status 200 Ok

6.5.7 Registering

This section will go through using the Register method in details. Then, each messages of the call flow will be detailed in order to see the SIP Message Header offloading by the TB640 SIP stack. Outgoing registers must all use the same reserved session id (0x800000000). Of course they must use different dialog ids.

6.5.7.1 Call flow for register client

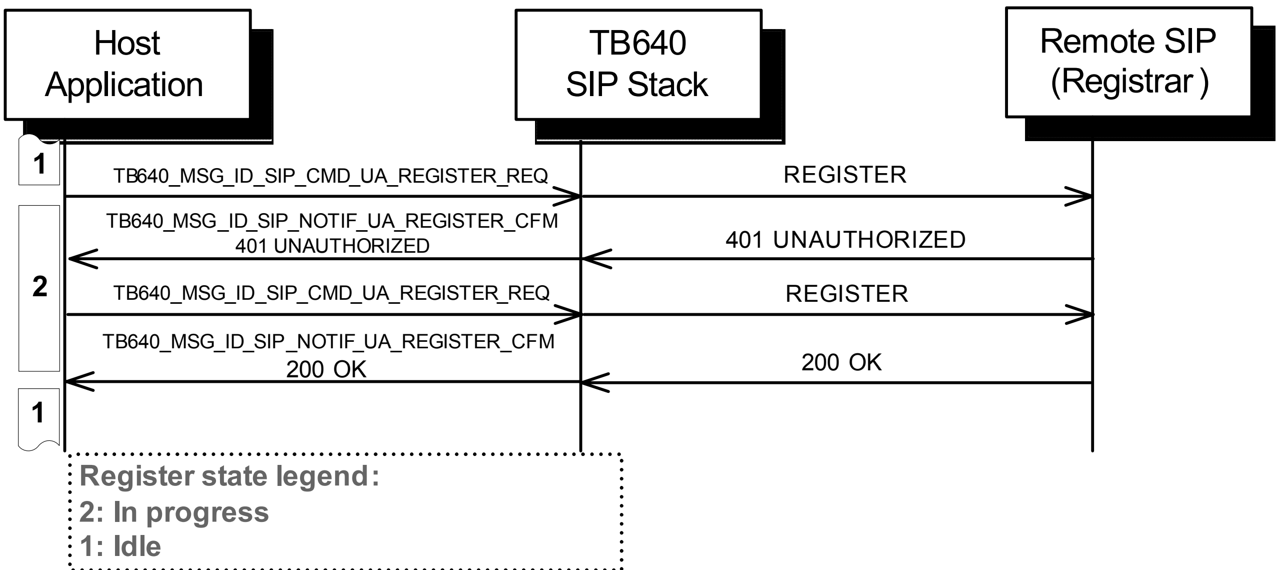


Figure 38 – Register client call flow

In this call flow, the local host initiates a REGISTER request. The remote host receives the request and responds to indicate that the registration needs authorization. The local host resends the register request with the correct authorization parameters and the remote host confirms the registration succeeded.

6.5.7.2 Call flow for register server

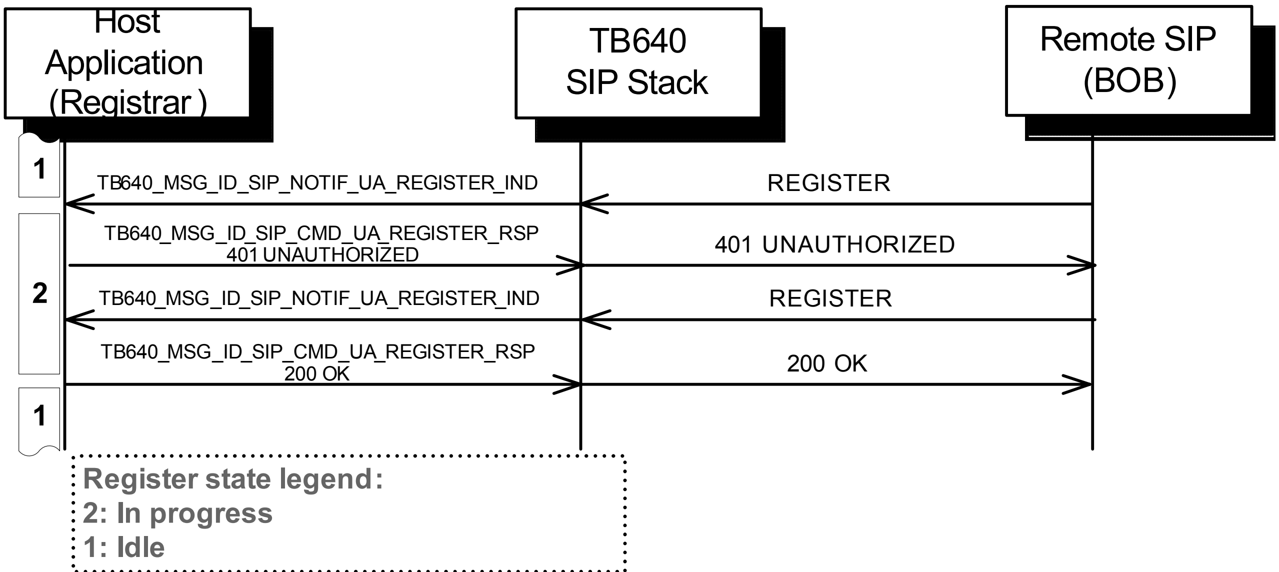


Figure 39 – Register server call flow

In this call flow, the remote host initiates a REGISTER request. The local host receives the request and responds to indicate that the registration needs authorization. The remote host resends the register request with the correct authorization parameters and the local host confirms the registration succeeded.

6.5.7.3 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

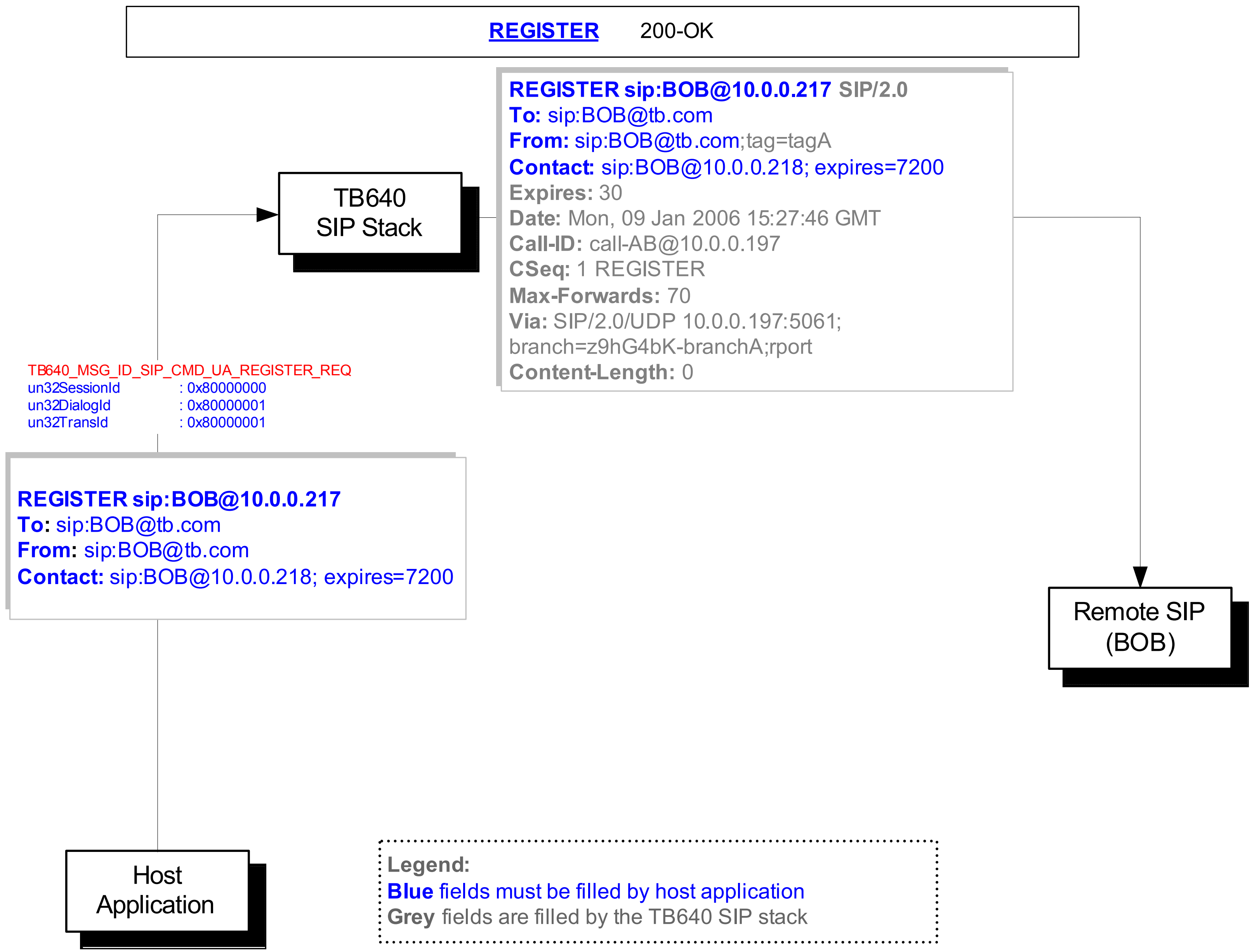


Figure 40 – Register request

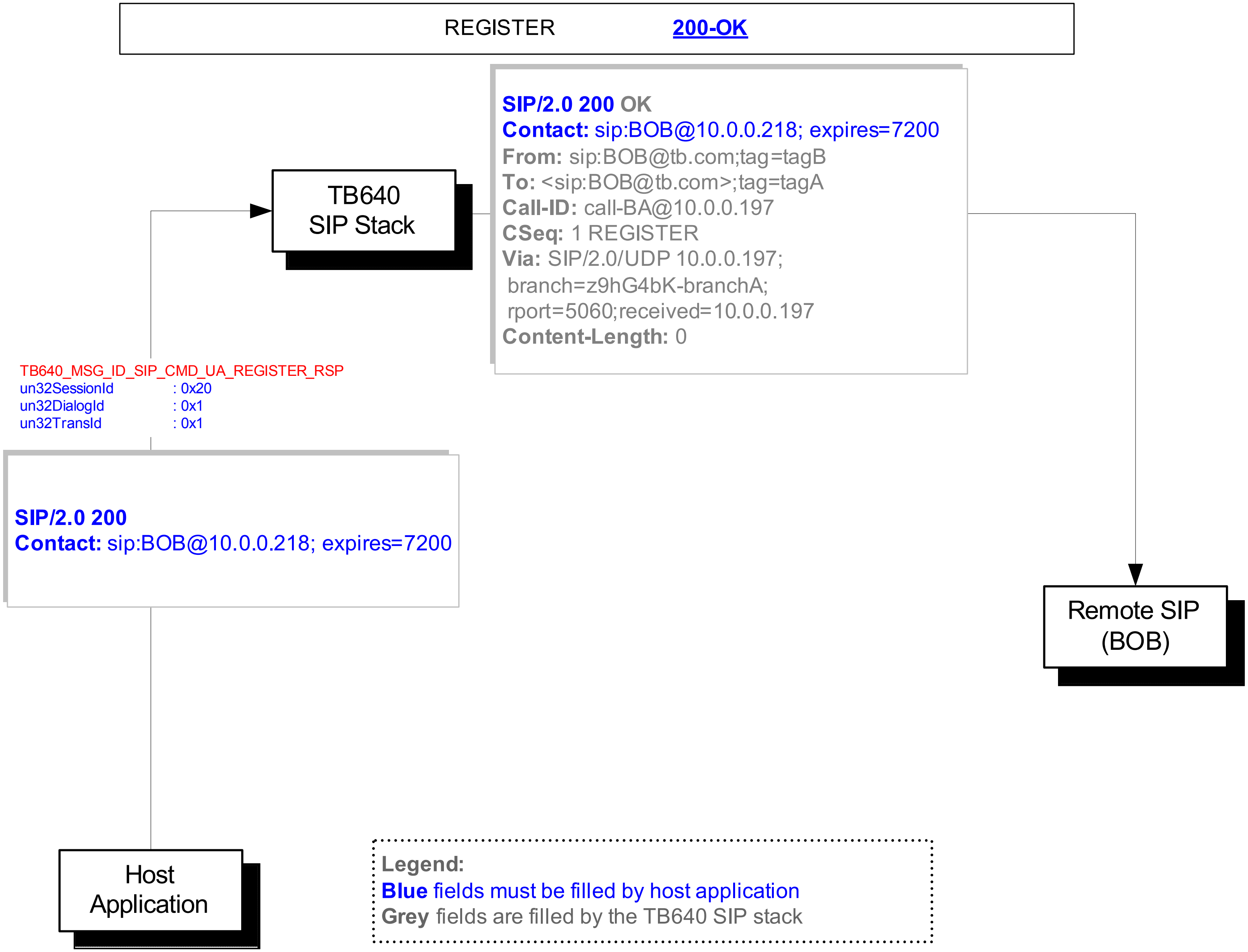


Figure 41 - Register response – Status 200 Ok

6.5.8 Options method

This section will go through using the Options method in details. Then, each messages of the call flow will be detailed in order to see the SIP Message Header offloading by the TB640 SIP stack. This method may be sent or received on an existing dialog id as defined in the specification.

6.5.8.1 Call flow for options client

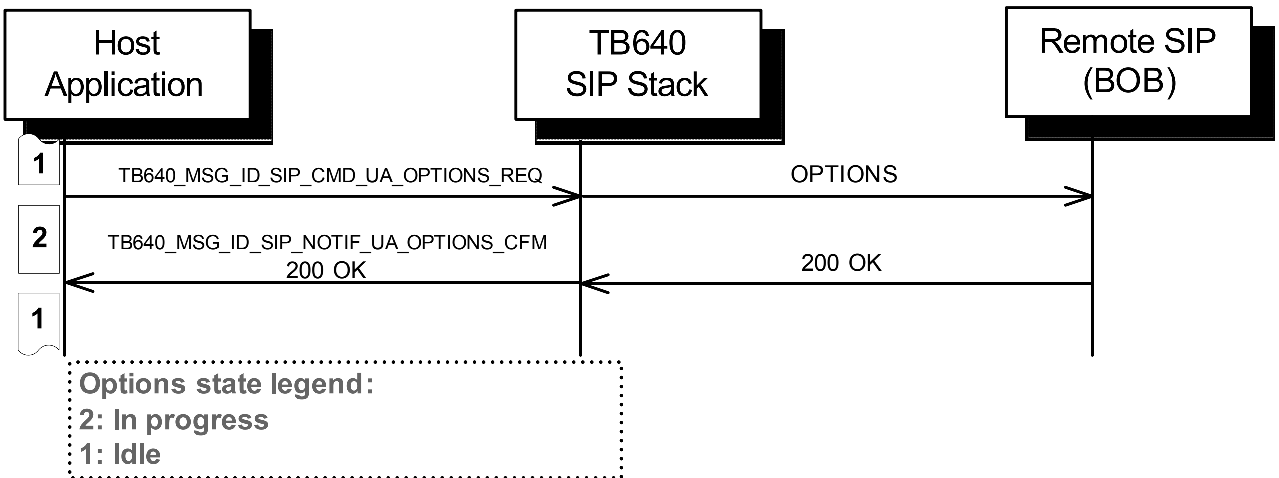


Figure 42 – Options client call flow

In this call flow, the local host initiates an OPTIONS request. The remote host receives the request and confirms the request by sending the same SDP it would use for a call.

6.5.8.2 Call flow for options server

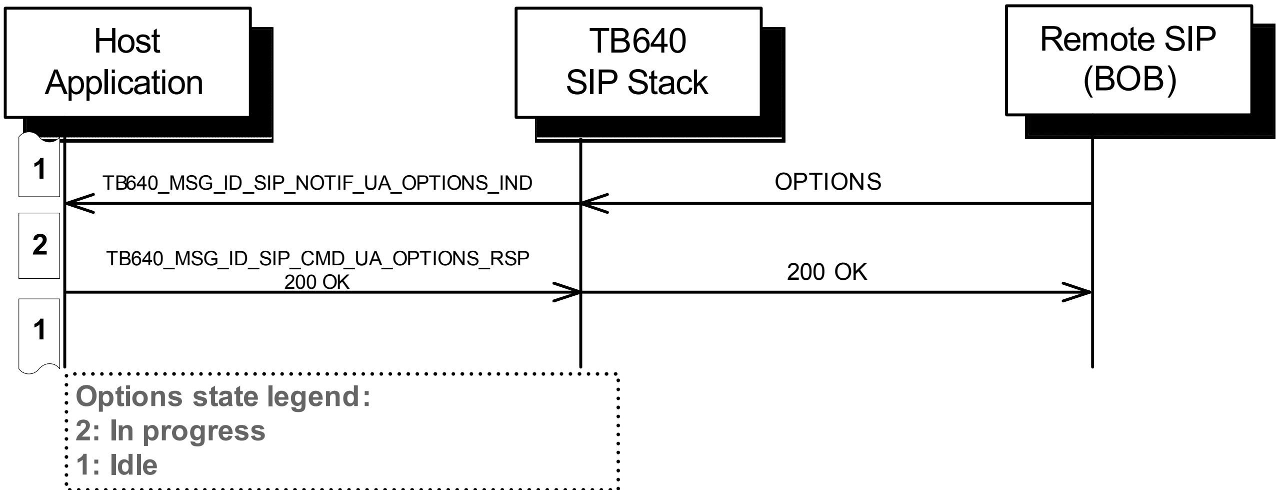


Figure 43 – Options server call flow

In this call flow, the remote host initiates an OPTIONS request. The local host receives the request and must confirm the request by sending the same SDP it would use for a call.

6.5.8.3 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

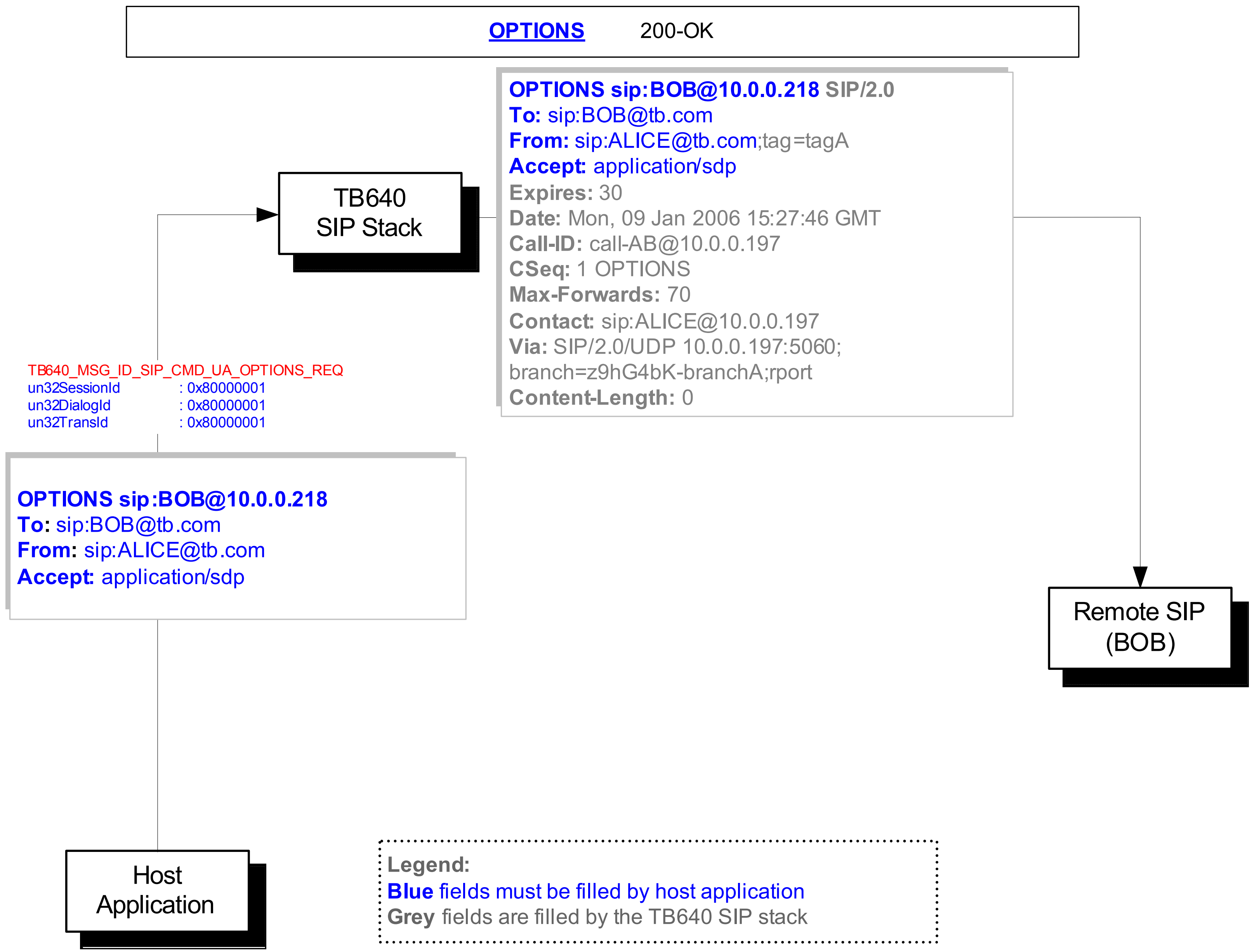


Figure 44 – Options request

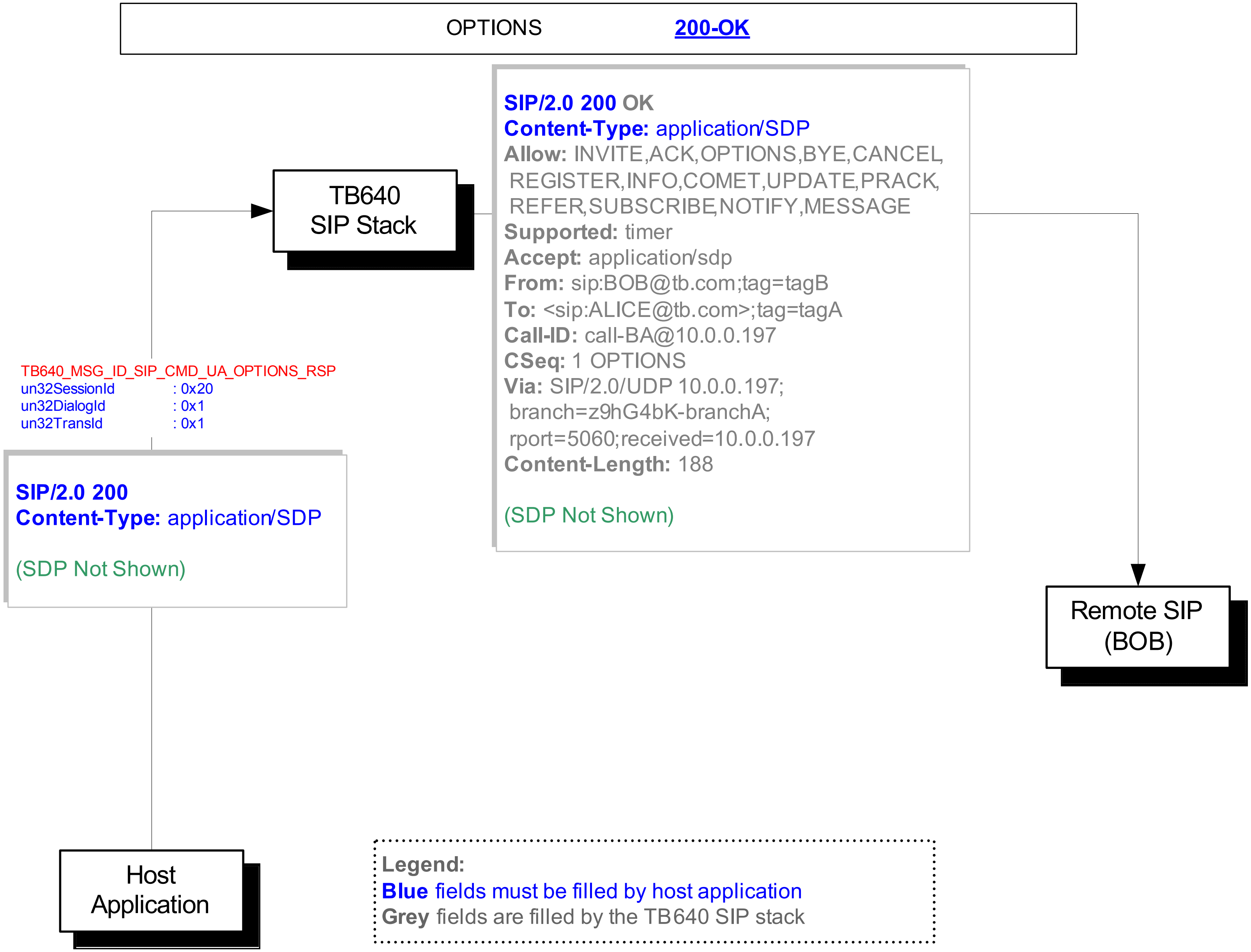


Figure 45 – Options response – Status 200 Ok

6.5.9 Refer/Notify method

This section will go through using the Refer and Notify methods in details. Then, each messages of the call flow will be detailed in order to see the SIP Message Header offloading by the TB640 SIP stack. This method may be sent or received on an existing dialog id as defined in the specification.

6.5.9.1 Call flow for refer client

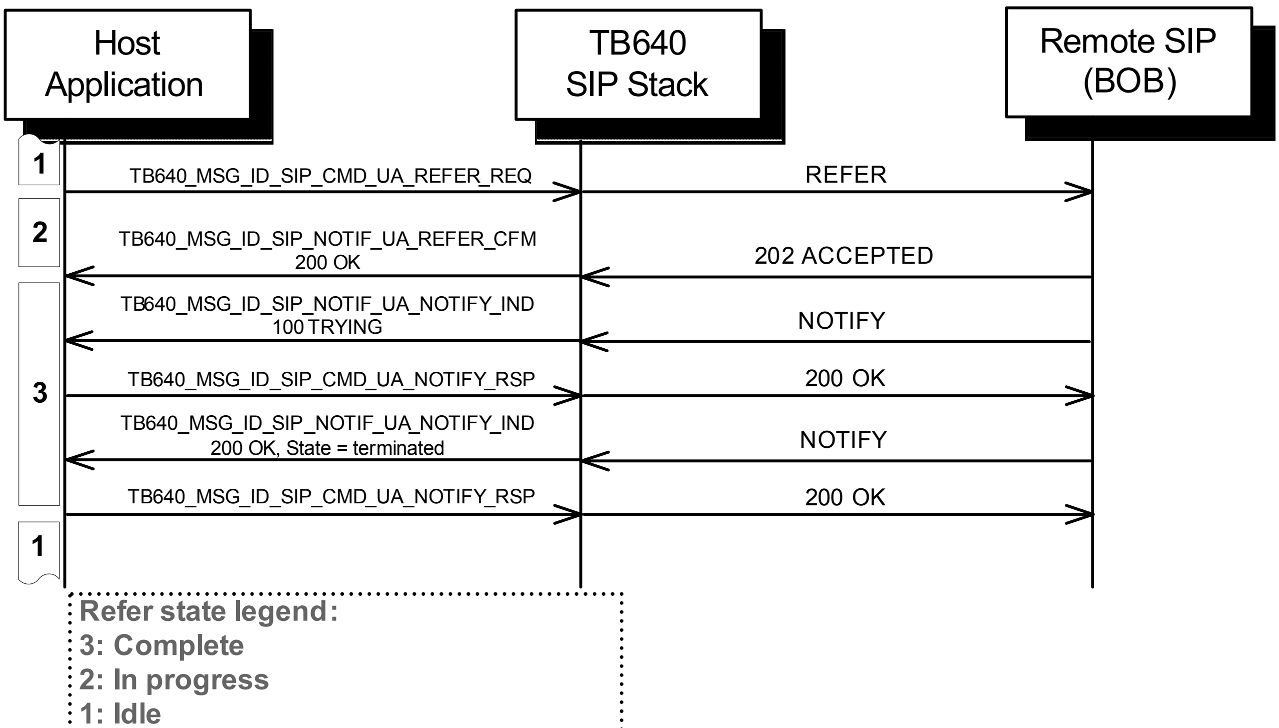


Figure 46 – Refer client call flow

In this call flow, the local host initiates an REFER request. The remote host receives the request and confirms the request, it must also send a first NOTIFY to confirm the subscription. The NOTIFY is confirmed by the local host and when the remote host has finished referring it sends another NOTIFY to terminate the subscription which is also confirmed by the local host.

6.5.9.2 Call flow for refer server

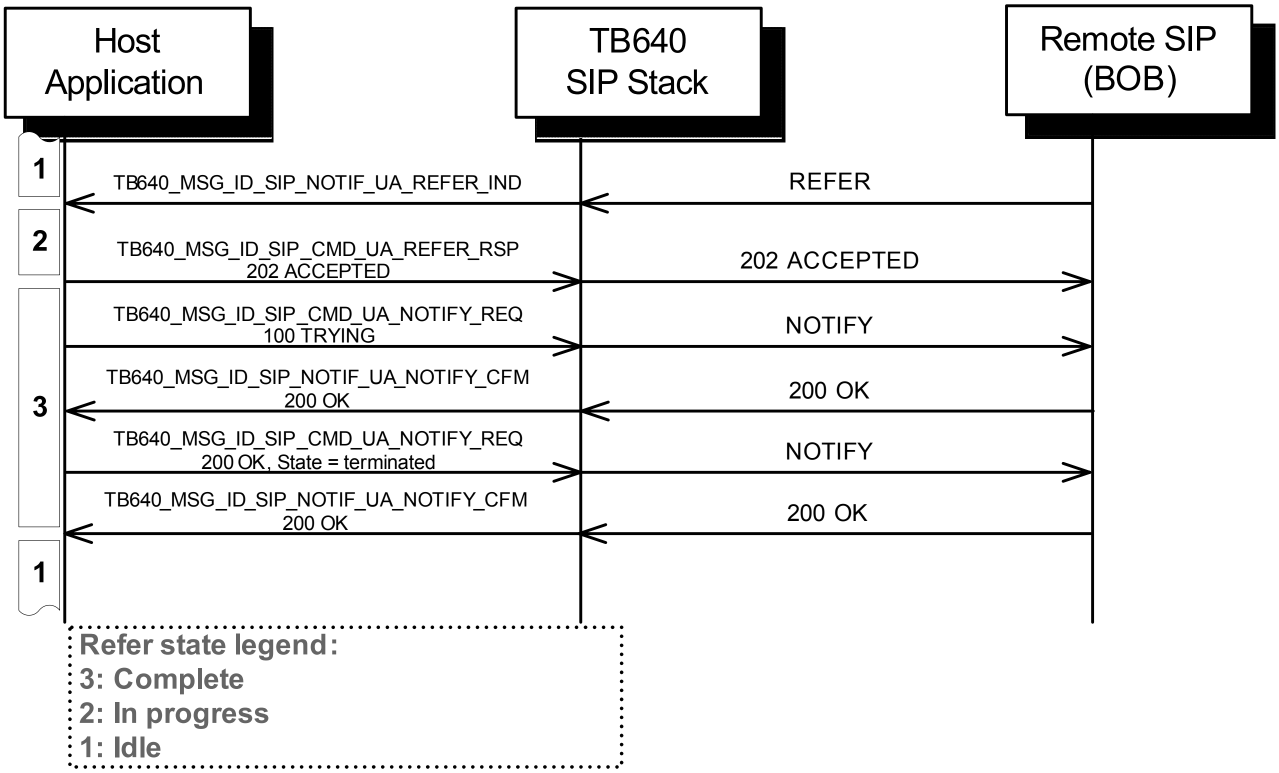


Figure 47 – Refer server call flow

In this call flow, the remote host initiates an REFER request. The local host receives the request and confirms the request, it must also send a first NOTIFY to confirm the subscription. The NOTIFY is confirmed by the remote host and when the local host has finished referring it sends another NOTIFY to terminate the subscription which is also confirmed by the remote host.

6.5.9.3 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

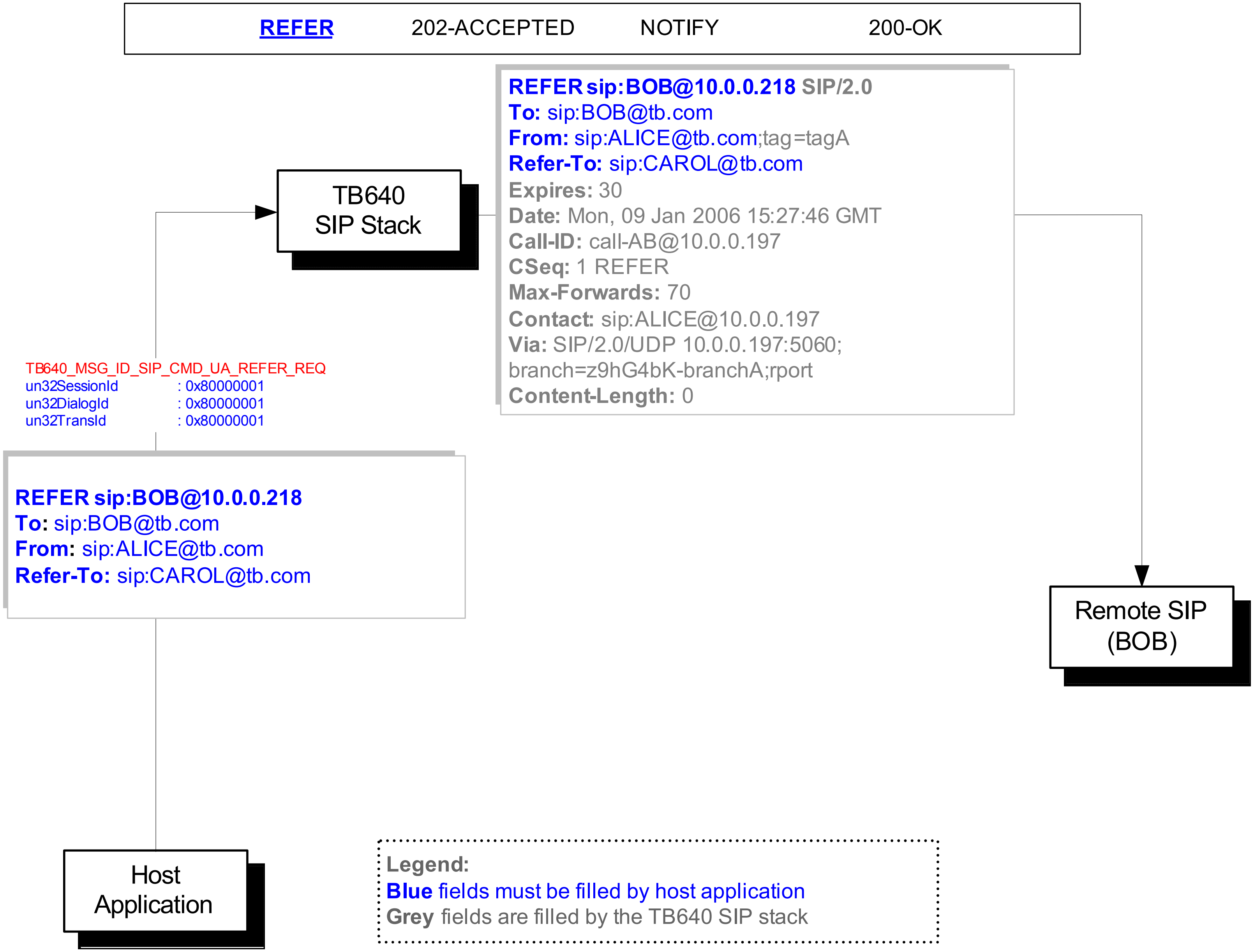


Figure 48 – Refer request

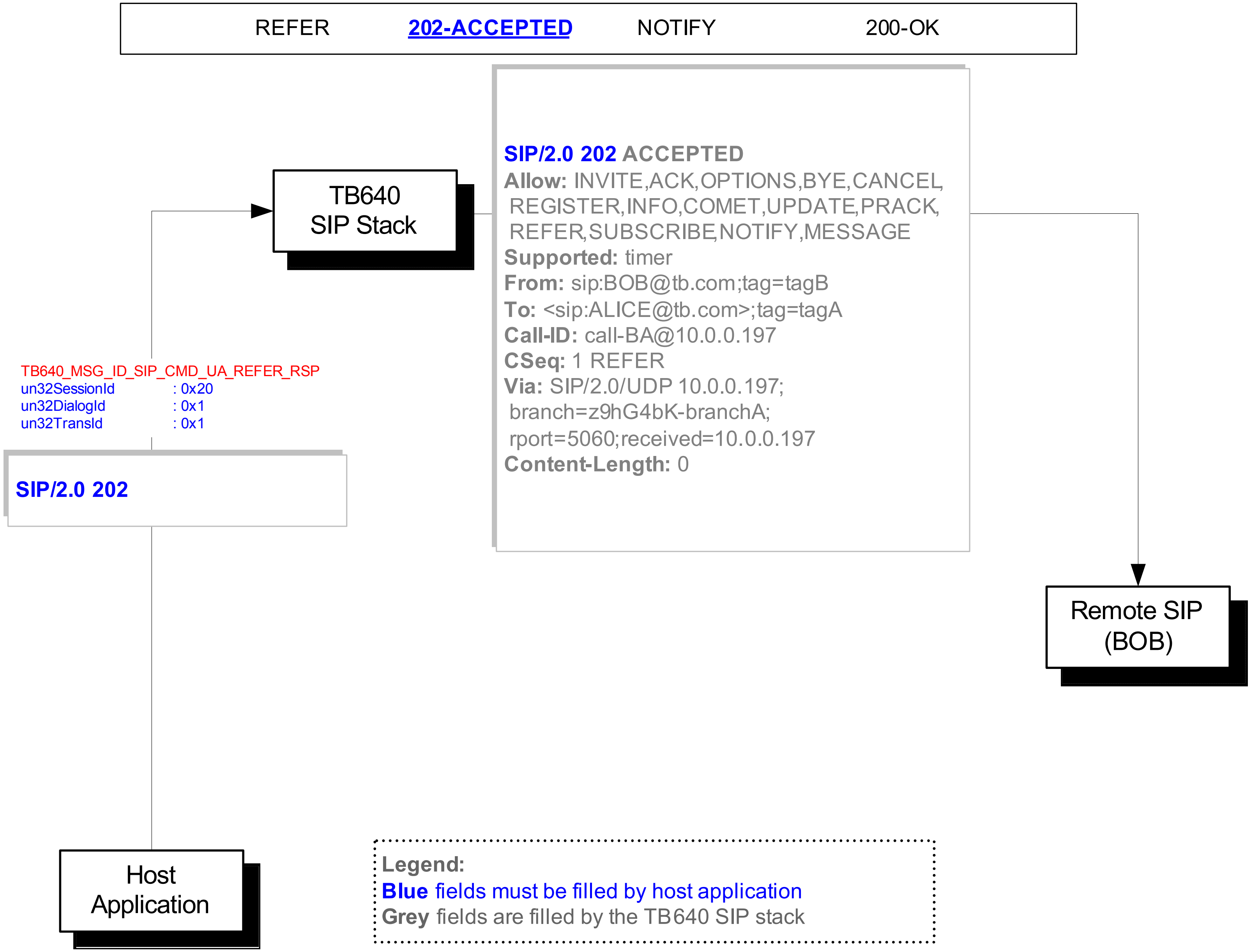


Figure 49 – Refer response – Status 202 Accepted

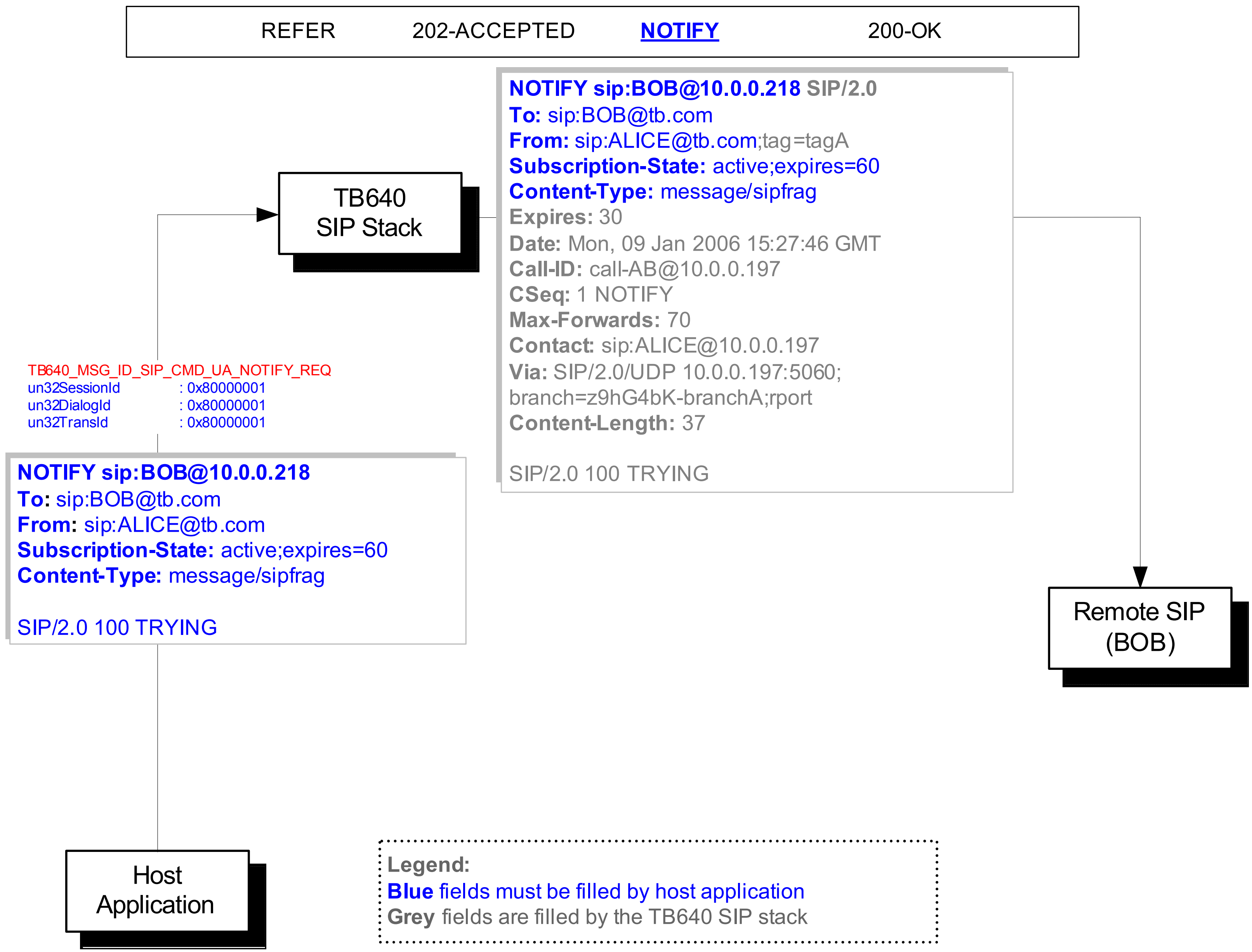


Figure 50 – Notify request

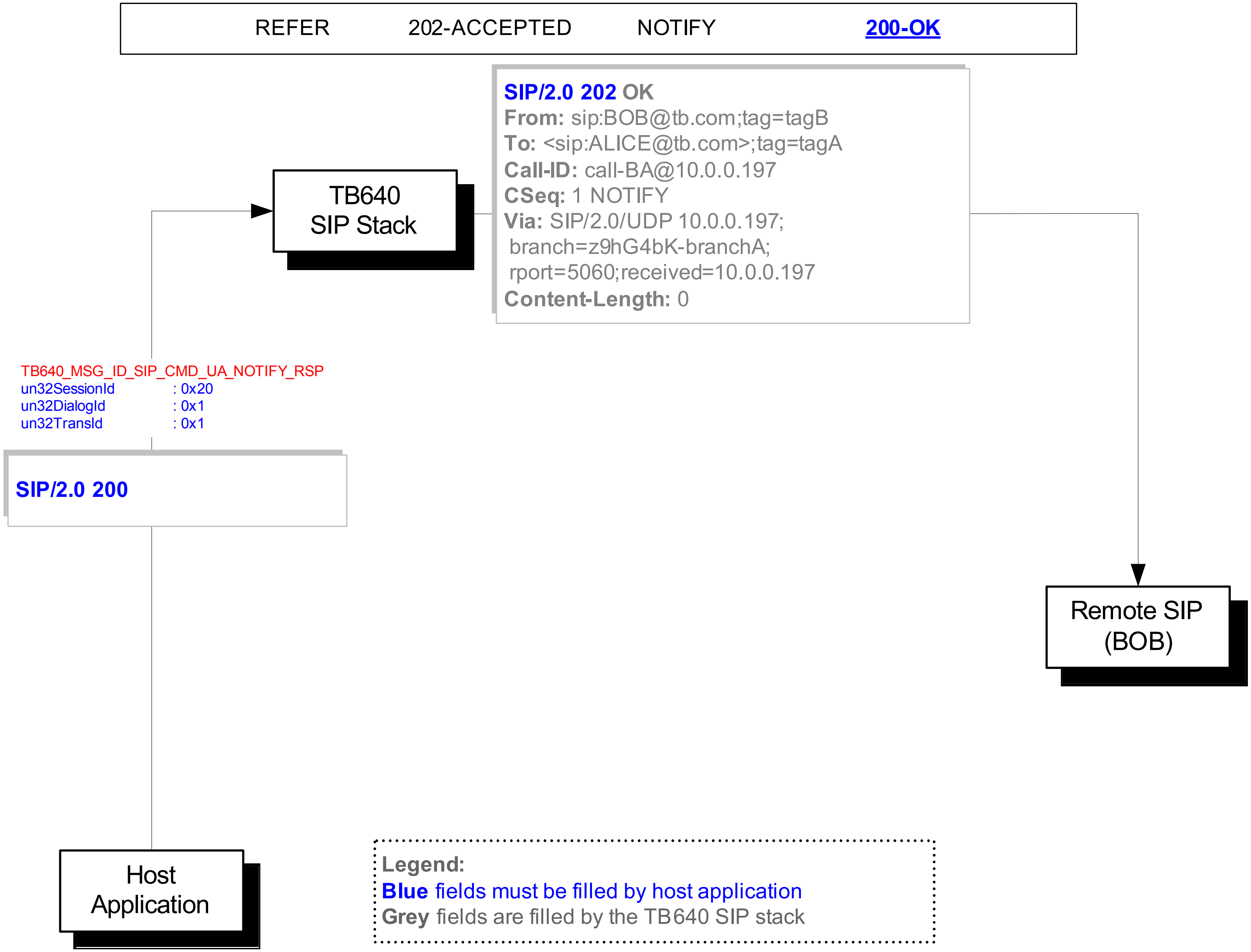


Figure 51 – Notify response – Status 200 Ok

6.5.10 Info method

This section will go through using the Info method in details. Then, each messages of the call flow will be detailed in order to see the SIP Message Header offloading by the TB640 SIP stack. This method must be sent or received on an existing dialog id as defined in the specification.

6.5.10.1 Call flow for info client

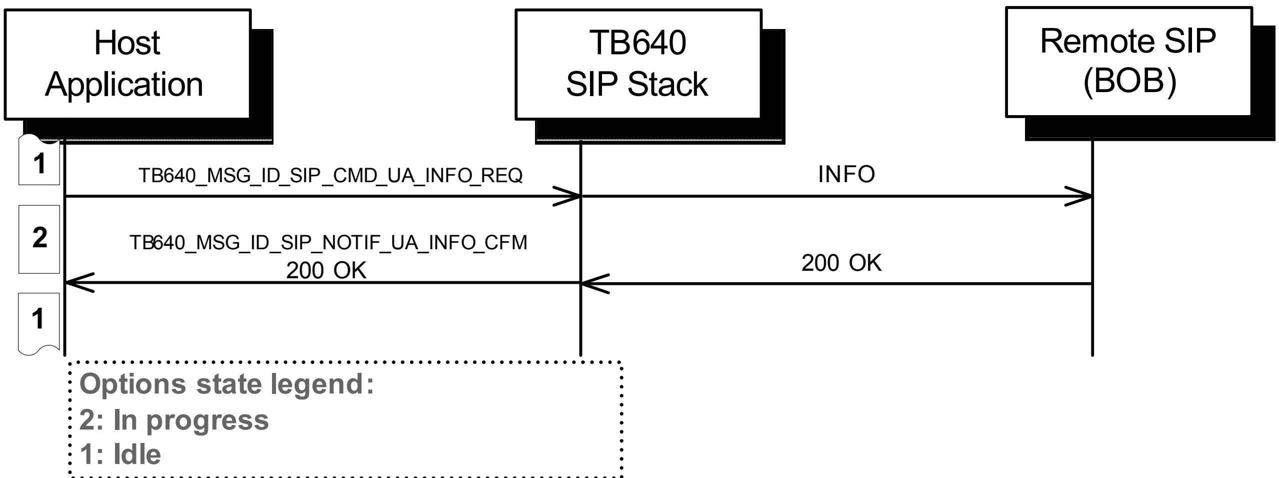


Figure 52 – Info client call flow

In this call flow, the local host initiates an INFO request. The remote host receives the request and acknowledges it.

6.5.10.2 Call flow for info server

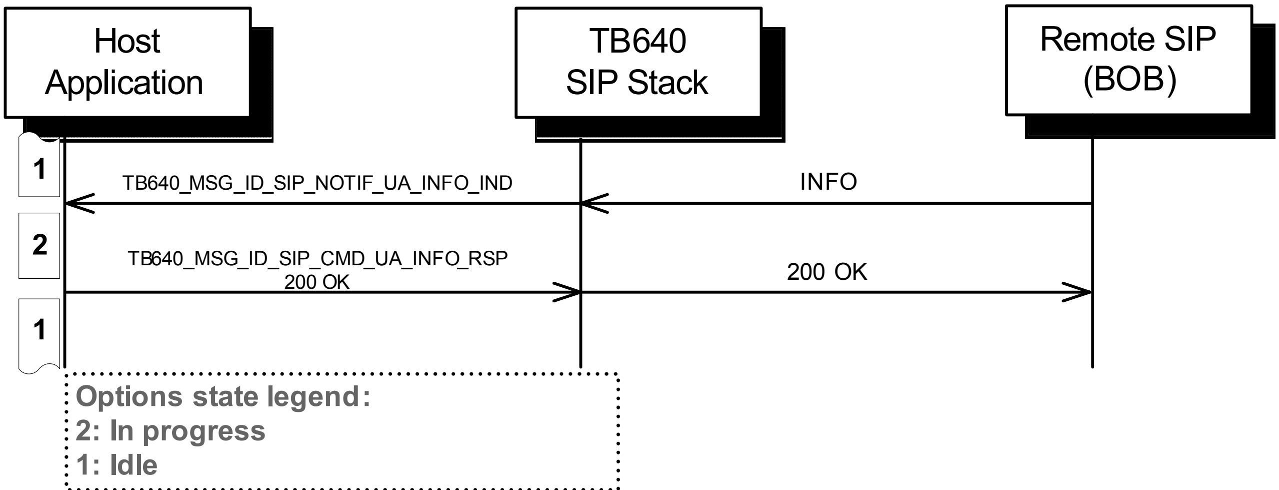


Figure 53 – Info server call flow

In this call flow, the remote host initiates an INFO request. The local host receives the request and must acknowledge it.

6.5.10.3 Detailed SIP Messages

The few next figures give an in detail view of all messages exchanged between the host and the TB640 as well as between the TB640 and the Remote SIP stack. Each SIP Header fields are given a color depending who's responsible for filling them out (either the host or the TB640).

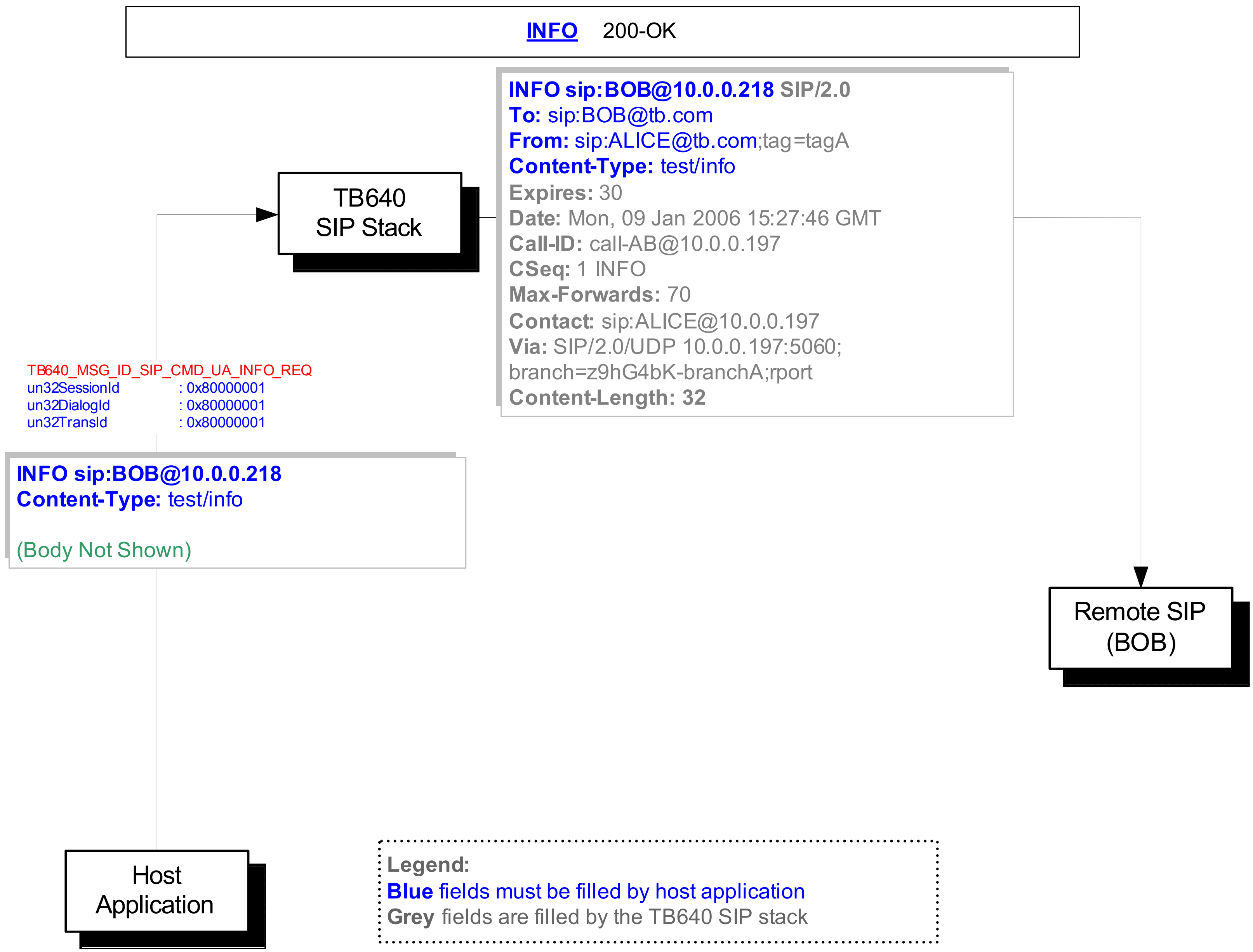


Figure 54 – Options request

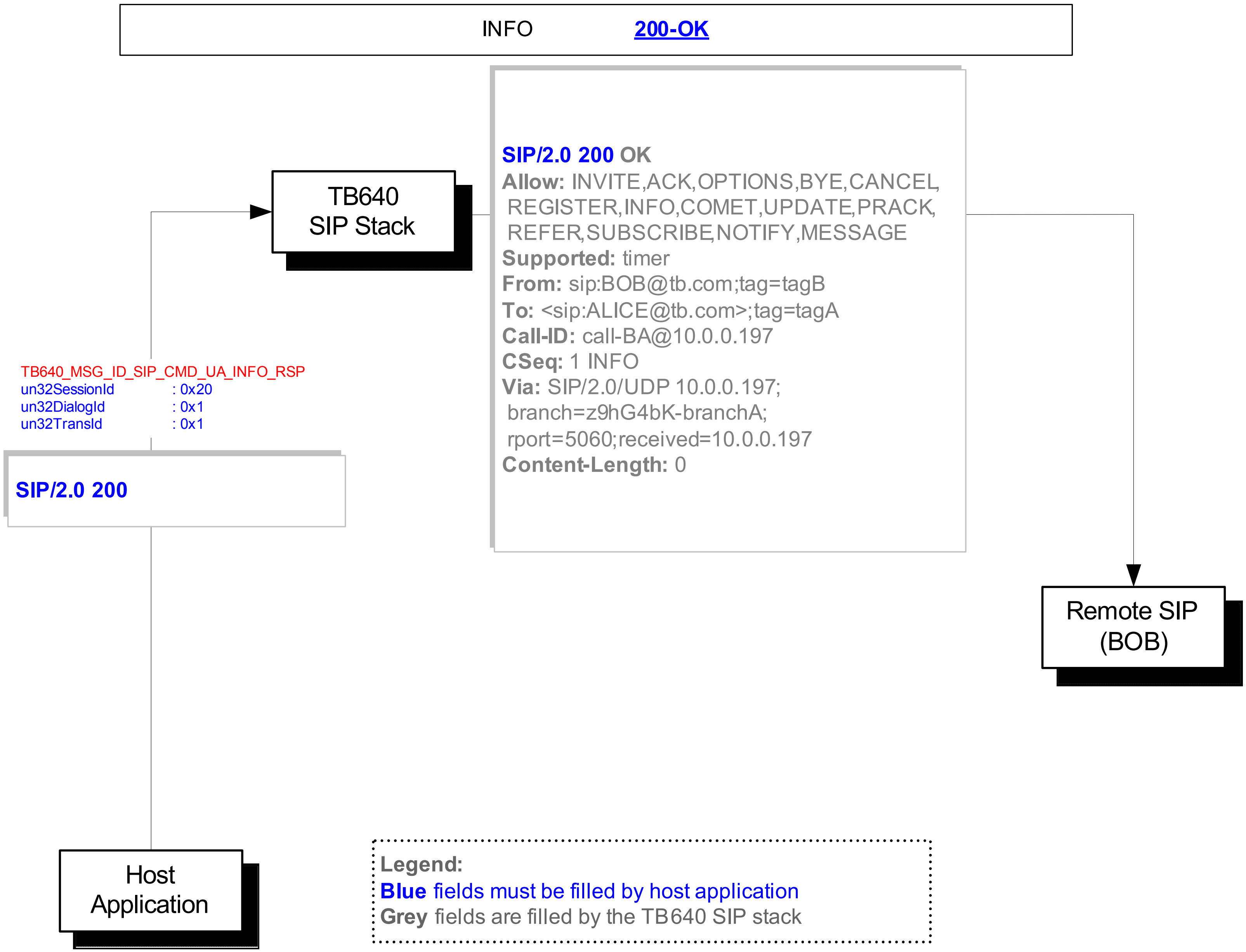


Figure 55 – Options response – Status 200 Ok

6.5.11 Replaces header

This header is useful for attended call transfer and perhaps other services. The replaces header may be filled for outgoing invites and it may also be filled for refers. For incoming invites the `TB640_MSG_ID_SIP_NOTIF_UA_REPLACES_IND` is basically a normal invite but it contains the replaced session/dialog ids. This will provide support for services that require the replaces header but it is up the client to implement the services.

6.6 Error handling

The TB640 SIP stack generates events notification on error conditions (TB640_MSG_ID_SIP_NOTIF_ ERROR). Those error condition events should be monitored to help identify run-time problems as well as configuration problems. The following structure gives information on the error condition:

```
typedef struct _TB640_EVT_SIP_NOTIF_ERROR
{
	TBX_MSG_HEADER				Header;
	TBX_UINT32				un32MsgVersion;

	TB640_SIP_SAP_HANDLE			hSipSap;
	TBX_UINT32				un32SessionId;
	TBX_BOOLEAN				fSessionCleared;
	TBX_UINT32				un32DialogId;
	TBX_UINT32				un32TranId;

	TB640_SIP_ERROR				Error;
	TB640_SIP_ERROR_MSG_TYPE		ErrorMsgType;
	TB640_SIP_ERROR_MSG_SRC			ErrorMsgSrc;

} TB640_EVT_SIP_NOTIF_ERROR, *PTB640_EVT_SIP_NOTIF_ERROR;
```

un32StructVersion: Version of the structure. Should be set to 1.
hSipSap: The handle of the user SAP.
un32SessionId: Session identifier. Null if not applicable.
fSessionCleared: Associated SessionId/DialogId have been terminated by the stack.
un32DialogId: Dialog identifier. Zero if not applicable.
un32TranId: Transaction identifier. Zero if not applicable.
Error: SIP error (TB640_SIP_ERROR).

Table 2 - TB640_SIP_ERROR description

Error	Description
TB640_SIP_ERROR_TIMEOUT	SIP stack timeout error. Possible causes: • INVITE outgoing 2xx response timeout – Ack not received. 64*T1 timer expires. • Reliable provisional responses timeout – Prack not received • Transport Error. Either Server or Client connection failure. • Incoming response with Dialog not matching existing Dialog. • Incoming response match fail; the dialog was not created by the method for which this rsp

	was received.
TB640_SIP_ERROR_ENCODING	SIP stack ABNF Encoding error. Possible causes: • Wrong SIP opaque string message in SIP request.
SOT_ERR_USER_LOC_REG_REQD	SIP user must be locally registered. Possible causes: • Outgoing SIP request or response with the From header containing a user that in not locally registered.
TB640_SIP_ERROR_NO_REG_SVR_CFG	No Registrar Address configured for this entity. Possible causes: • SIP REGISTER request with no registrar configured (see *RegistrarCfg* section 6.2.1.2).
TB640_SIP_ERROR_INVITE_REQ_FAILED	Invite Request failed. Possible causes: • Pending INVITE being processed. Invite response on that dialog haven't been received.
TB640_SIP_ERROR_REGISTER_REQ_FAILED	Register Request failed. Possible causes: • The remote registrar returned a response error (error>2xx) to a stack generated refreshed Registration. • Invalid contact in a SIP REGISTER request.
TB640_SIP_ERROR_TPTSRVR_SELECT	Transport server selection failed Possible causes: • Request sent over a SAP with no transport server configured for the required transport protocol type.
TB640_SIP_ERROR_DNS_FAILED	DNS Lookup has failed Possible causes: • Invalid host or domain name specified.
TB640_SIP_ERROR_LOCREG_INV	Local user registration is for 3rd Party Possible causes: • fAllow3rdPartyReg is disabled. To and From header must be the same.

TB640_SIP_ERROR_SENDING	Error in sending SIP message Possible causes: • Invite server transaction Ack not received for a 3xx,4xx,5xx,6xx response; timer H expires. • Non-Invite client transaction response not received in Trying or Proceeding state; timer F expires. • Invite client transaction response not received in Calling state; timer B expires.
TB640_SIP_ERROR_REG_CONTACT_STAR	Outgoing SIP REGISTER request with Contact * and no expires.
TB640_SIP_ERROR_DIALOGSTATE_INVALID	SIP message not allowed in this dialog state.
TB640_SIP_ERROR_CONTENTTYPE_NOTFOUND	Required Content-Type header not found in SIP message.
TB640_SIP_ERROR_MIMEBNDRY_NOTFOUND	MIME Boundary value not found.
TB640_SIP_ERROR_VIA_INVALID	SIP message Via header not matching with correct transport server.
TB640_SIP_ERROR_REG_IN_PROGRESS	There is already a registration in progress in the system sent to the same Registrar.
TB640_SIP_ERROR_REL_PROVRSP_NOTALLWD	Reliable provisional response is not supported by peer user agent.
TB640_SIP_ERROR_SUBSC_TMO	Subscribe timeout Possible causes: • Subscription was not refreshed by the refresher and timed out.
TB640_SIP_ERROR_REFER_TMO	Refer timeout Possible causes: • Refer subscription was not refreshed by the refresher and timed out.
TB640_SIP_ERROR_INVALID_CALLID	Invalid “Call-Id” header Possible causes: • “Call-Id” header was found by the stack in the SIP request/response message and doesn’t match the SIP request/response SessionId and DialogId.
	Unsupported method

TB640_SIP_ERROR_METHOD_NOTSUPP	Possible causes: • Outgoing UPDATE message request sent but not supported.
TB640_SIP_ERROR_MULTRSP_DIALOG	Additional call dialog for a forked call being cleared.
TB640_SIP_ERROR_EXPIRES_TIMEOUT	Expires Timer timeout. Possible causes: • INVITE request with Expires header have timed out.
TB640_SIP_ERROR_CANCEL_TIMEOUT	Cancel Timer timeout. Possible causes: • CANCEL request with have timed out.
TB640_SIP_ERROR_ANSWER_PENDING TB640_SIP_ERROR_OFFER_REQUIRED	Answer/Offer failure Possible causes: • If SDP is not present in the SIP message when there is a pending offer. • New SDP offer is sent when there is a pending offer.

ErrorMsgType: SIP error Msg Type (TB640_SIP_ERROR_MSG_TYPE). Gives the type of SIP message in witch the error occurs.

Table 3 - TB640_SIP_ERROR_MSG_TYPE description

Error message type	Description
TB640_SIP_ERROR_MSG_TYPE_INVITE	Error occurs in Invite message
TB640_SIP_ERROR_MSG_TYPE_ACK	Error occurs in Ack message
TB640_SIP_ERROR_MSG_TYPE_OPTIONS	Error occurs in Options message
TB640_SIP_ERROR_MSG_TYPE_BYE	Error occurs in Bye message
TB640_SIP_ERROR_MSG_TYPE_CANCEL	Error occurs in Cancel message
TB640_SIP_ERROR_MSG_TYPE_REGISTER	Error occurs in Register message
TB640_SIP_ERROR_MSG_TYPE_INFO	Error occurs in Info message
TB640_SIP_ERROR_MSG_TYPE_PRACK	Error occurs in Provisional Ack msg
TB640_SIP_ERROR_MSG_TYPE_REFER	Error occurs in Refer message
TB640_SIP_ERROR_MSG_TYPE_SUBSCRIBE	Error occurs in Subscribe message
TB640_SIP_ERROR_MSG_TYPE_NOTIFY	Error occurs in Notify message
TB640_SIP_ERROR_MSG_TYPE_MESSAGE	Error occurs in Message message
TB640_SIP_ERROR_MSG_TYPE_UPDATE	Error occurs in Update message
TB640_SIP_ERROR_MSG_TYPE_LOCAL_USER_ALLOC	Error occurs in Local User Alloc message
TB640_SIP_ERROR_MSG_TYPE_UNKNOWN	Unknown

ErrorMsgSrc: SIP error message source (TB640_SIP_ERROR_MSG_SRC).

Table 4 - TB640_SIP_ERROR_MSG_SRC description

Error message source	Description
TB640_SIP_ERROR_MSG_SRC_USER	Error occurs in a SIP message coming from the service user.
TB640_SIP_ERROR_MSG_SRC_NETWORK	Error occurs in a SIP message coming from the peer.

6.7 Alarms

The TB640 SIP stack generates events notification on event conditions (TB640_MSG_ID_SIP_NOTIF_ALARM). Those alarm condition events might be monitored to help identify run-time problems as well as configuration problems. The following structure gives information on the alarm condition:

✍ The Alarms interface may slightly change in the post Beta release.

```
typedef struct _TB640_EVT_SIP_NOTIF_ALARM
{
	TBX_MSG_HEADER				Header;
	TBX_UINT32				un32MsgVersion;

	TB640_SIP_ALARM_EVENT			Event;
	TB640_SIP_ALARM_INFO			AlarmInfo;

} TB640_EVT_SIP_NOTIF_ALARM, *PTB640_EVT_SIP_NOTIF_ALARM;
```

un32StructVersion: Version of the structure. Should be set to 1.
Event: Alarm event (TB640_SIP_ALARM_EVENT).
AlarmInfo: Alarm information (TB640_SIP_ALARM_INFO).

```
	typedef struct _TB640_SIP_ALARM_INFO
	{
		TB640_SIP_ALARM_INFO_TYPE		AlarmInfoType;
		union
		{
			TB640_SIP_SAP_HANDLE		hSipSap;
			TBX_BOOL			fIsMemoryCong;
		};

	} TB640_SIP_ALARM_INFO, *PTB640_SIP_ALARM_INFO;
```

Table 5 – Common SIP Alarm description

Event	Category	Cause	AlarmInfoType	Description
SAP_READY	INTERFACE	NONE	SAP	The SAP is now Ready to receive API cmd.
RES_CONG_STRT	RESOURCE	NONE	CONG	Adapter is experiencing congestion. Stop sending requests until it stops.
RES_CONG_STOP	RESOURCE	NONE	CONG	Adapter has stopped experiencing congestion. You may continue sending requests.

6.8 States

SIP uses Resource Congestion Thresholds to determine current resource threshold levels before processing incoming and outgoing requests. The congestion threshold may be triggered by memory exhaustion or by the complete call count reaching the license value. When the congestion threshold reaches a critical point, an alarm is raised (`TB640_SIP_ALARM_EVENT_RES_CONG_STRT`) and all new requests are dropped. More precisely when `fIsMemoryCong` is set to true all new transactions are dropped but existing transactions are allowed to continue. When it is set to false new transactions are accepted as long as they do not create a new call. When the resource threshold drops to a certain level another alarm is raised (`TB640_SIP_ALARM_EVENT_RES_CONG_STOP`) and all requests will be allowed again.

6.9 Accounting

When the `fAccountingIndication` configuration parameter has been enabled in user agent configuration, the SIP stack will send accounting information to the application:

- When a call is established.
- When a call is modified using a re-invite.
- When a call is terminated.

The following structure gives information on accounting indications:

```
typedef struct _TB640_EVT_SIP_NOTIF_ACCOUNTING
{
	TBX_MSG_HEADER					Header;
	TBX_UINT32					un32MsgVersion;

	TB640_SIP_ENTITY_HANDLE				hSipUa;
	TBX_UINT32					un32SessionId;
	TBX_UINT32					un32DialogId;
	TB640_SIP_ACCOUNTING_EVENT_TYPE			NotifType;
	TB640_SIP_CALL_ACCOUNTING			AccountingInfo;

} TB640_EVT_SIP_NOTIF_ACCOUNTING, *PTB640_EVT_SIP_NOTIF_ACCOUNTING;
```

un32StructVersion: Version of the structure. Should be set to 1.
hSipUa: The handle of the user agent.
un32SessionId: Session identifier.
un32DialogId: Dialog identifier.
NotifType: Type of accounting (TB640_SIP_ACCOUNTING_EVENT_TYPE).
AccountingInfo: Accounting information (TB640_SIP_CALL_ACCOUNTING).

```
	typedef struct _TB640_SIP_CALL_ACCOUNTING
	{
		TBX_BOOLEAN					fCallOriginator;
		TBX_UINT8					aun8Padding0 [3];
		TBX_UINT32					un32CallStart;
		TB640_SIP_CALL_DURATION				CallDuration;
		TB640_SIP_CALL_MEDIA_TYPE			MediaType;
		TB640_SIP_SDP_MEDIA_TYPE			DecSdpMedia;
		TB640_SIP_STRING				LocalAddr;
		TB640_SIP_STRING				RemAddr;

	} TB640_SIP_CALL_ACCOUNTING, *PTB640_SIP_CALL_ACCOUNTING;
```


End of the document

