TTC技術レポート
Technical Report

TR-1007

Session Initiation Protocol (SIP) に関する技術レポート

Technical Report on Session Initiation Protocol (SIP)

第 1 版

2003 年 3 月 14 日制定

社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TTC Telecommunication Technology Committee

目　次

<参考>

1. 概要

本技術レポートは、IETF SIP WG に検討されれている RFC3261(SIP : Session Initiation Protocol) [1] に対し作成されたものである。TTC 第一部門委員会第三専門委員会 SWG3 では、日本国内における SIP の標準化策定を実施することを目的とし、RFC3261[1] をベースに日本語訳の検討を行った。その結果を本技術レポートとして制定するものである。

また、日本語訳を行った際に抽出された、訳として紛らわしい用語(単語、熟語および文章)の対訳表ならびに要求レベルを表す語句(RFC2119)の訳文を付属資料として本技術レポートに添付する。

(1) 国際勧告等との関連

本技術レポートはIETF勧告 RFC3261[1] (2002.6 RFC として策定) に基づき日本語訳を行ったものである。

(2) 上記国際勧告等に対する追加項目等

なし。(付属資料として「用語対訳表」が追加されている)

(3) 上記国際勧告等に対する変更事項

なし。

(4) 参照した国際勧告との章立て構成の相違

なし。(RFC3261[1]の章立てと同様)

(5) 改版の履歴

版数	制定日	改版内容
第 1 版	2003 年 3 月 14 日	初版制定

(6) 工業所有権

なし。(本書の配布は無制限である)

(7) その他(参照している勧告、標準等)

[1] “Session Initiation Protocol”, RFC3261, June 2002, Internet Engineering Task Force

その他 RFC3261[1] 内で参照している勧告、標準については本文 29 章(Normative References)及び 30 章(Informative References)にて記述している。

2. 対象となる RFC の前文

Network Working Group
Request for Comments: 3261
Obsoletes: 2543
Category: Standards Track

J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia U.
G. Camarillo
Ericsson
A. Johnston
WorldCom
J. Peterson
Neustar
R. Sparks
dynamicsoft
M. Handley
ICIR
E. Schooler
AT&T
June 2002

SIP: Session Initiation Protocol

Status of this Memo

Copyright Notice

Abstract

This document describes Session Initiation Protocol (SIP), an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences.

SIP invitations used to create sessions carry session descriptions

that allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. SIP also provides a registration function that allows users to upload their current locations for use by proxy servers. SIP runs on top of several different transport protocols.

Network Working Group
Request for Comments: 3261
Obsoletes: 2543
Category: Standards Track

J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia U.
G. Camarillo
Ericsson
A. Johnston
WorldCom
J. Peterson
Neustar
R. Sparks
dynamicsoft
M. Handley
ICIR
E. Schooler
AT&T

June 2002

SIP(セッション開始プロトコル)
SIP: Session Initiation Protocol

本文書の位置付け

この文書は、インターネットコミュニティのためのインターネット標準トラックプロトコルを規定するものであり、改善のための議論や提案を依頼するものである。標準化の状態や、プロトコルの位置付けについては最新版の"Internet Official Protocol Standards" (STD 1)を参照されたい。この文書の配布は無制限である。

著作権表記

概要

この文章では、単数あるいは複数の相手とのセッションを生成、変更、切断するためのアプリケーションレイヤ制御(シグナリング)プロトコルである、セッション開始プロトコル(SIP)について述べる。ここでいうセッションとは、インターネット電話呼、マルチメディア配信、マルチメディアカンファレンスを含んだものである。

セッションを生成するために使われるSIP招待は、参加者が互換メディアタイプをマッチさせるための、セッション記述を伝える。ユーザの現在ロケーションへリクエストをルーティングする役に立ち、サービスに対してユーザを認証および認可し、プロバイダの呼ルートポリシーを実装し、かつ、ユーザに機能を提供す

る、プロキシサーバと呼ばれるエレメントを SIP は使用する。プロキシサーバが使用するユーザの現在のロケーションをユーザがアップロードすることを認める登録機能も、SIP は提供する。SIP は異なるさまざまなプロトコル上で動作する。

１．はじめに (Introduction)

セッションの生成と管理を必要とする、多くのインターネットのアプリケーションがある。ここで、セッションとは、関係する相手どおしでのデータ交換とみなされる。これらのアプリケーションの実装は、参加者の行動によって複雑化する。つまり、ユーザはエンドポイント間で移動するかもしれない、1 人のユーザを複数の名前でアドレス指定できるかもしれない、そして、ユーザはさまざまな異なるメディアで、ときには同時に通信するかもしれない。音声やビデオあるいはテキストメッセージといった、さまざまな形式のリアルタイムマルチメディアセッションデータを伝える、多くのプロトコルが立案されている。Session Initiation Protocol(SIP)は、インターネットのエンドポイント(ユーザエージェントと呼ばれる)がお互いを発見し、共有を望むセッションの特性に合意することを可能にすることで、これらのプロトコルと協調して動作する。セッションの参加者と見込まれるものの場所を特定するため、また他の機能のために、SIP は、ユーザエージェントが登録リクエストやセッションへの招待およびその他のリクエストを送ることができる、ネットワークホスト(プロキシサーバと呼ばれる)のインフラを生成することを可能にする。SIP は、下位のトランスポートプロトコルから独立し、確立されるセッションのタイプに依存せずに動作する、セッションを生成・修正・終了するための機動的で多目的なツールである。

２．SIP 機能概要 (Overview of SIP Functionality)

SIP は、インターネット電話呼のようなマルチメディアセッション(カンファレンス)を確立・修正・終了できるアプリケーションレイヤの制御プロトコルである。SIP は既存の、マルチキャストカンファレンスのようなセッションに参加者を招待することもできる。メディアを既存のセッションに追加(および削除)することができる。SIP は、パーソナルモビリティ(参考文献[27])(ユーザはネットワーク上のロケーションに関係なく、外部から見える単一の識別子を保持することができる)をサポートするネームマッピングとリダイレクションサービスを透過的にサポートする。

SIP は、マルチメディアコミュニケーションの確立と終了に関する 5 つの側面をサポートする。

ユーザロケーション(user location):
　コミュニケーションに使用されるエンドシステムの決定。

ユーザ可用性(user availability):
　着信側パーティのコミュニケーションに参加する意思の決定。

ユーザ能力(user capabilities): 使用されるメディアとメディアパラメータの決定。

セッションセットアップ(session setup):
　「呼び出し」。発信側パーティと着信側パーティ双方でのセッションパラメータの決定。

セッション管理(session management):
　セッションの転送・終了、セッションパラメータの修正、サービスの起動を含む。

SIP は垂直的に統合されたコミュニケーションシステムではない。SIP はむしろ、完全なマルチメディアアーキテクチャを築き上げるために他の IETF のプロトコルとともに使用することができるコンポーネントである。通常、これらのアーキテクチャは次のようなプロトコルを含む。リアルタイムデータのトランスポートと QoS フィードバックを提供するリアルタイムトランスポートプロトコル(RTP)(RFC1889 参考文献

[28])、ストリーミングメディアの配送を制御するためのリアルタイムストリーミングプロトコル(RTSP)(RFC2326 参考文献[29])、公衆交換電話網(PSTN)へのゲートウェイを制御するメディアゲートウェイ制御プロトコル(MEGACO)(RFC3015 参考文献[30])、およびマルチメディアセッションを記述するためのセッション記述プロトコル(SDP)(RFC2327 参考文献[1])。そのため、ユーザに完全なサービスを提供するために、SIP は他のプロトコルと共に使用されるべきである。しかしながら、SIP の基本機能とオペレーションは、これらのプロトコルのいずれにも依存しない。

SIP はサービスを提供しない。むしろ SIP は、別のサービスを実装するために使用されるプリミティブを提供する。たとえば、SIP はユーザのロケーションを特定し、その現在のロケーションに不透明オブジェクト(opaque object)を配送できる。たとえば、もしこのプリミティブが SDP で書かれたセッション記述を配送するために使用されると、エンドポイント間でセッションのパラメータを合意できる。もし同じプリミティブが、セッション記述と同様に発信者の写真を配送するために使用される場合、「caller ID」サービスが容易に実装できる。この例が示すように、ひとつのプリミティブは一般的にいくつもの異なるサービスを提供するために使用される。

SIP はフロアコントロールや投票(voting)のようなカンファレンス制御サービスを提供せず、カンファレンスがどのように管理されるか規定しない。SIP は、他のカンファレンス制御プロトコルを使用するセッションを開始するために使用することができる。SIP メッセージとそれが確立するセッションはまったく別のネットワークに渡すことができるので、SIP はいかなる種類のネットワークリソース確保能力も提供できないし、提供しない。

提供されるサービスの性質は、セキュリティを特に重要とする。このため SIP は、DoS 攻撃防御、認証(ユーザ-ユーザ間およびプロキシ-ユーザ間の両方)、完全性防御、および暗号化とプライバシーサービスを含む、セキュリティサービス一式を提供する。

SIP は IPv4 および IPv6 の両方とともに動作する。

３．述語規則 (Terminology)

このドキュメントでは、次のキーワードは BCP 14、RFC2119(参考文献[2])に記述されているとおりに解釈され、SIP に準拠した実装のための要求レベルを示す。

「MUST」、「MUST NOT」、「REQUIRED」、「SHALL」、「SHALL NOT」、
「SHOULD」、「SHOULD NOT」、「RECOMMENDED」、「NOT RECOMMENDED」、
「MAY」、および「OPTIONAL」

４．オペレーション概要 (Overview of Operation)

本節では、簡単な例を用いて SIP の基本オペレーションを紹介する。本節は本質的にチュートリアルであって、規範となる記述は含まない。

最初の例では SIP の基本機能(エンドポイントのロケーション特定、通信の要求の発信、セッションを確立するためのセッションパラメータのネゴシエーション、および確立されたセッションの解除)を示す。

図 1 は 2 人のユーザ(Alice と Bob)の間の SIP メッセージ交換の典型的な例を示している。(各メッセージ

は、文章中での参照用として「"F"+番号」としてラベル付けしてある)。この例では、インターネット上でBob の SIP フォンと通話するために、Alice は 彼女の PC 上で SIP アプリケーション(ソフトフォン)を使用する。図 1 には、セッションの確立を容易にするために Alice と Bob に成り代わって動作する、2 台の SIP プロキシサーバも示されている。この典型的な配置は、図 1 の点線の図形で示されるように、「SIP トラペゾイド(台形)」と呼ばれることが多い。

Alice は Bob の SIP アイデンティティ(SIP URI と呼ばれる URI の一種)を使用して「通話する」。SIP URI については 19.1 節で定義されている。それは E-mail アドレスと似た形式であり、通常、ユーザ名とホスト名を含む。この例では、sip:bob@biloxi.com である。ここで、biloxi.com は Bob の SIP サービスプロバイダのドメインである。Alice は sip:alice@atlanta.com という SIP URI を持っている。Alice はおそらく、Bob の SIP URI をキーボードから入力するか、ハイパーリンクあるいはアドレス帳のエントリをクリックしただろう。SIP はまた、SIPS URI と呼ばれるセキュアな URI も提供する。たとえば、sips:bob@biloxi.com である。SIPS URI にかけられた通話は、発信者から着信者のドメインまで、すべての SIP メッセージを伝えるためにセキュアな暗号化されたトランスポート(すなわち TLS)が使用されることを保証する。そこから、リクエストは着信者のドメインのポリシーに依存するセキュリティメカニズムを使用して、安全に着信者まで送られる。

SIP は HTTP に似たリクエスト/レスポンスのトランザクションモデルに基づいている。各トランザクションは、サーバ上である特定のメソッドまたは関数を呼び出すリクエストと、少なくとも一つのレスポンスからなる。この例では、Alice のソフトフォンが Bob の SIP URI に宛てて INVITE リクエストを送ることでトランザクションが始まる。INVITE は、リクエストを送る側(Alice)がサーバ(Bob)にとってほしい動作を指定する、SIP メソッドの一例である。INVITE リクエストは多くのヘッダフィールドを含む。ヘッダフィールドは、メッセージについての追加情報を提供する、名前付けされた属性である。INVITE 中に存在するヘッダフィールドは、呼のための一意な識別子、送信先アドレス、Alice のアドレス、および Alice が Bob と確立したいセッションのタイプについての情報を含む。INVITE(図 1 のメッセージ F1)は次のような形である。

図 1: SIP のセッションセットアップ例(SIP トラペゾイドあり)

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

(Alice の SDP は示されていない)

テキスト形式でエンコードされたメッセージの１行目はメソッド名(INVITE)を含む。それに続く行はヘッダフィールドのリストである。この例では必要とされる最小限のフィールドのセットを含む。ヘッダフィールドについて以下に簡単に述べる。

Via は、Alice がこのリクエストに対するレスポンスの受信を望むアドレス(pc33.atl anta.com)を含む。また、このトランザクションを識別する branch パラメータも含む。

Ｔｏは,ﾘｸｴｽﾄがもともと差し向けられた先の,ディスプレイネーム(Bob)と SIP URI または SIPS URI(sip:bob@biloxi.com)を含む。ディスプレイネームについては RFC2822(参考文献[3])で述べられている。

From もリクエストの発信元のディスプレイネーム(Alice)と SIP URI または SIPS URI(sip:alice@atlanta.com)を含む。このヘッダフィールドは、ソフトフォンが URI に追加した乱数文字列(1928301774)を含む tag パラメータも持つ。これは識別の目的で使用される。

Call-ID は、乱数文字列とソフトフォンのホスト名または IP アドレスを組み合わせて生成された、この呼のグローバルに一意な識別子を含む。To tag、From tag、および Call-ID の組み合わせは、Alice と Bob 間においてピア・トゥ・ピアの SIP リレーションシップを完全に定義し、ダイアログと呼ばれる。

CSeq(Command Sequence)は、整数値とメソッド名を含む。CSeq 番号は新しいリクエストごとにインクリメントされる。CSeq 番号は通常の連続した番号である。

Contact は、Alice にコンタクトするためのダイレクトルートを表す、通常は完全修飾ドメイン名(FQDN)からなる SIP URI または SIPS URI を含む。FQDN が望ましいとはいえ、多くのエンドシステムはドメイン名を登録していないので、IP アドレスが許可されている。Via ヘッダフィールドがレスポンスをどこに送るかを他のエレメントに伝えるのに対して、Contact ヘッダフィールドは、以降のリクエストをどこに送るかを他のエレメントに伝える。

Max-Forwards は、リクエストが送信先に向かう過程でとることができるホップ数を制限するために役立つ。Max-Forwards は、ホップ毎に１づつデクリメントされる整数からなる。

Content-Type は、メッセージボディ(例では示されていない)の説明を含む。

Content-Length は、メッセージボディのオクテット(byte)カウントを含む。

SIP ヘッダフィールドの完全なセットは 20 節で定義されている。

セッションの詳細、例えばメディアのタイプ、コーデック、サンプリングレートは、SIP を使用して記述されない。むしろ、SIP メッセージのボディが、他のプロトコルフォーマットでエンコードされたセッションの記述を含む。そのようなフォーマットの一つがセッション記述プロトコル(SDP)(RFC2327 参考文献[1])である。この SDP メッセージ(例には示されていない)は、E-mail メッセージでドキュメントのアタッチメントが運ばれるように、あるいは HTTP メッセージで Web ページが運ばれるように、SIP メッセージによって運ばれる。

ソフトフォンは Bob のロケーション、あるいは biloxi.com ドメインの SIP サーバのロケーションを知らないので、Alice のドメイン(atlanta.com)の SIP サーバに INVITE を送る。atlanta.com の SIP サーバのアドレスは、たとえば、Alice のソフトフォンに設定されているかもしれないし、あるいは DHCP によって発見されるもしれない。

atlanta.com の SIP サーバは、プロキシサーバとして知られるタイプの SIP サーバである。プロキシサーバはリクエストを送った者の代わりに SIP リクエストを受信し、それをフォワードする。この例では、プロキシサーバが INVITE リクエストを受信し、Alice のソフトフォンに 100(Trying)レスポンスを送り返す。100(Trying)レスポンスは、INVITE が受信され、プロキシが Alice のソフトフォンに成り代わって、送信先に INVITE のルーティングを試みていることを示す。SIP でのレスポンスは、数値 3 桁のコードとそれに続く説明フレーズを用いる。このレスポンスは、INVITE と同じ To、From、Call-ID、CSeq、および branch パラメータを持つ Via を含む。これは、Alice のソフトフォンが、送信した INVITE にこのレスポンスを関係付けることを可能にする。atlanta.com のプロキシサーバは、biloxi.com ドメインの SIP を見つけるために、おそらく特定のタイプの DNS(Domain Name Service)ルックアップを実行して、biloxi.com のプロキシサーバの場所を特定する。これについては参考文献[4]で述べられている。結果として、それは biloxi.com のプロキシサーバの IP アドレスを取得し、INVITE リクエストをそこにフォワード(またはプロキシ)する。リクエストをフォワードする前に、atlanta.com のプロキシサーバはそれ自身のアドレスを含む付加的な Via ヘッダフィールド値を追加する　(INVITE は最初の Via に Alice のアドレスをすでに含んでいる)。biloxi.com のプロキシサーバは INVITE を受信し、それが INVITE を受信しリクエストを処理していることを示すために、atlanta.com のプロキシサーバに 100(Trying)レスポンスで応答する。プロキシサーバは、Bob の現在の IP アドレスを含む一般的にロケーションサービスと呼ばれるデータベースを検索する。(このデータベースをどのようにして存在させることができるか、次の節でわかるだろう)。biloxi.com のプロキシサーバは、それ自身のアドレスを含むもう一つの Via ヘッダフィールド値を INVITE に追加し、それを Bob の SIP フォンにプロキシする。

Bob の SIP フォンは INVITE を受信すると、Bob が Alice からかかってきた通話を受けるかどうか決定できるように、その通話を Bob に知らせる。つまり、Bob の電話が鳴る。Bob の SIP フォンは、2 つのプロキシを反対方向にルーティングされて返される、180(Ringing)レスポンスでこのことを示す。各プロキシは、レスポンスをどこに送るか決定するために Via ヘッダフィールドを使用し、先頭から自分自身のアドレスを取り除く。結果として、最初の INVITE をルーティングするためには DNS とロケーションサービスのルックアップが必要とされたが、180(Ringing)レスポンスはルックアップやプロキシで保持されている状態を使用せずに発信者に返送することができる。このことはまた、INVITE を見る各プロキシは、その INVITE へのすべてのレスポンスをも見ることになるという望ましい特性も持つ。

Alice のソフトフォンが 180(Ringing)レスポンスを受信すると、おそらく ringback あるいは画面上にメッセージを表示して、Alice に知らせる。

この例では、Bob は通話を受けることにする。彼が受話器を取ると、彼の SIP フォンは、通話が受けられたことを示すために 200(OK)レスポンスを送信する。200(OK)は、Bob が Alice と確立することを望むセッションのタイプの SDP メディア記述を持つメッセージボディを含む。結果として、2 段階の SDP メッセージ交換があることになる。Alice が一つを Bob に送り、Bob は一つを Alice に送り返す。この 2 フェーズの交換は、基本的なネゴシエーション能力を提供する。そしてこれは、SDP 交換の単純なオファー/アンサーモデルに基づいている。もし Bob が通話を受けたくなかったり、他の通話で話中だった場合、200(OK)のかわりに

エラーレスポンスが返され、メディアセッションが確立されない結果になっただろう。SIP レスポンスコードの完全なリストは 21 節にある。Bob が送るときの 200(OK)(図 1 のメッセージ F9)は以下のようである。

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com
;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com
;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com
;branch=z9hG4bK776asdhds ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob の SDP は示されていない)
```

レスポンスの最初の行はレスポンスコード(200)とリーズンフレーズ(OK)を含む。残りの行はヘッダフィールドを含む。Via、To、From、Call-ID、および CSeq ヘッダフィールドは INVITE リクエストからコピーされた。(3 つの Via ヘッダフィールド値がある。Alice の SIP フォンが追加したもの、atlanta.com のプロキシが追加したもの、biloxi.com のプロキシが追加したものである)。Bob の SIP フォンが To ヘッダフィールドに tag パラメータを追加した。この tag は両方のエンドポイントによってダイアログに組み込まれ、この呼の以降のすべてのリクエストとレスポンスに含められる。Contact ヘッダフィールドは、Bob の SIP フォンで彼に直接到達できる URI を含む。Content-Type と Content-Length は、Bob の SDP メディア情報を含む(ここでは示されていない)メッセージボディについて言及している。

この例で示された DNS とロケーションサービスのルックアップに加えて、プロキシサーバは、リクエストをどこに送るか決定するために柔軟な「ルーティングの決定」を下せる。たとえば、もし Bob の SIP フォンが 486(Busy Here)レスポンスを返送した場合、biloxi.com のプロキシサーバは INVITE を Bob のボイスメールサーバにプロキシすることもできる。プロキシサーバは同時に多くのロケーションに INVITE を送ることもできる。このタイプのパラレルサーチはフォークとして知られている。

この場合、200(OK)は 2 つのプロキシを通してルートバックされ、Alice のソフトフォンによって受信される。それから、ringback を止め、通話が受けられたことを示す。最後に、最終レスポンス(Final response)(200(OK))の受信をコンファームするために、Alice のソフトフォンは確認メッセージ(ACK)を Bob の SIP フォンに送る。この例では、ACK は 2 つのプロキシをバイパスして Alice のソフトフォンから Bob の SIP フォンに直接送られる。これは、INVITE/200(OK)の交換を通して、2 つのエンドポイントが、最初の INVITE が送られたときには知らなかったお互いのアドレスを Contact ヘッダフィールドから知ったために起こる。2 つのプロキシが実行したルックアップはもはや必要ではないので、プロキシはコールフローから抜ける。これで、SIP セッションを確立するために使用された INVITE/200/ACK の 3 ウェイハンドシェイクが完了する。

セッションセットアップについての全詳細は13節で述べる。

Alice と Bob のメディアセッションが開始され、彼らは、SDP の交換で合意したフォーマットを使用してメディアパケットを送る。一般に、エンド・トゥ・エンドのメディアパケットはSIPのシグナリングメッセージとは別の経路を取る。

セッション中に、Alice または Bob のいずれかがメディアセッションの特性 (characteristics)の変更を決定できる。これは新しいメディア記述を含む re-INVITE を送ることで達成できる。この re-INVITE は、それが新たなセッションを確立するのではなく既存のセッションを修正するためのものであることを他のパーティが理解するように、既存のダイアログを参照する。他のパーティはその変更を受け入れるために200(OK)を送る。リクエストを送った側はその 200(OK)に対して ACK でレスポンスする。他のパーティが変更を受け入れない場合、488(Not Acceptable Here)のようなエラーレスポンスが送信される。これも ACK を受信する。しかしながら、re-INVITE の失敗は既存の呼を失敗させる原因にならない - 既にネゴシエートされた特性(characteristics)を用いてセッションは継続する。セッション修正についての全詳細は14節で述べる。

呼の終了時に、Bob が先に切断(ハングアップ)し、BYE メッセージを生成する。この BYE は、再びプロキシをバイパスして、Alice のソフトフォンにダイレクトにルーティングされる。Alice は BYE の受信を 200(OK)レスポンスで確認する。これはセッションと BYE トランザクションを終了する。ACK は送られない - ACK は、INVITE リクエストに対するレスポンスへのレスポンスとしてのみ送信される。この、INVITE の特別な対応の理由については後ほど議論するが、SIP の信頼性のメカニズム、呼び出し中の電話が受けられるまでに取ることができる時間長さ、およびフォークに関連している。この理由により、SIP におけるリクエスト対応は、INVITE または非 INVITE(INVITE 以外のすべてのメソッド)のいずれかに分類されることが多い。セッション終了についての全詳細は15節で述べる。

24.2 節では図 1 の例で示されたメッセージがすべて記載されている。

ある場合には、セッションが継続している間中、2 つのエンドポイント間のすべてのメッセージングを見ることは、SIP のシグナリングパスの中にあるプロキシにとって有用かもしれない。たとえば、biloxi.com のプロキシサーバが最初の INVITE 後も SIP のメッセージングパス中にとどまることを望む場合、それは INVITE に、解決されるとそのプロキシのホスト名または IP アドレスになる URI を含む、Record-Route として知られている、ルーティングに必要とされるヘッダフィールドを追加するだろう。この情報は、Bob の SIP フォンと(200(OK)で返されることになる Record-Route ヘッダフィールドによって)Alice のソフトフォンの双方に受信され、ダイアログが継続している間記憶される。その後 biloxy.com のプロキシサーバは、ACK、BYE、および BYE に対する 200(OK)を受信してプロキシするだろう。各プロキシは、それに続くメッセージングを受信することを独自に決定でき、メッセージングは、それを受信することを選んだすべてのプロキシを経由するだろう。この能力は、コール内機能(mid-call features)を提供するプロキシのためにたびたび使用される。

登録は、SIP におけるもう一つの通常オペレーションである。登録は biloxi.com のサーバが Bob の現在のロケーションを知ることができる一つの方法である。初期化時および定期的に、Bob の SIP フォンは SIP レジストラサーバとして知られる biloxi.com のサーバに REGISTER メッセージを送る。REGISTER メッセージは Bob の SIP URI または SIPS URI(sip:bob@biloxi.com)と彼が現在ログインしている装置を関連付ける

(Contact ヘッダフィールド中で SIP URI または SIPS URI として伝えられる)。レジストラサーバはこの関連付け(バインディングとも呼ばれる)をデータベース(ロケーションサービスと呼ばれる)に書き込む。それは biloxi.com ドメインのプロキシが利用できる。ドメインのレジストラサーバは、そのドメインのプロキシと同じ場所に配置されることが多い。SIP サーバのタイプの区別は論理的なものであって、物理的なものでないということは重要なコンセプトである。

Bob は、単一デバイスからの登録ということに制限を受けない。たとえば、自宅にある彼の SIP フォンとオフィスにある SIP フォンの両方で登録を送ることができる。この情報は共にロケーションサービスに記憶され、Bob のロケーションを特定するためにプロキシが様々なタイプの検索を実行することを可能にする。同様に、二人以上のユーザを、同時に一つのデバイス上に登録することができる。

ロケーションサービスはただの抽象的なコンセプトである。それは通常、プロキシが URI をインプットして、リクエストをどこに送るかをプロキシに教える 0 個以上の URI のセットを受け取ることを可能にする情報を含む。登録はこの情報を生成するための一つの方法であるが、唯一の方法ではない。管理者の判断で、任意のマッピング機能を設定できる。

最後に、SIP において登録は、送られてくる SIP リクエストのルーティングに使用され、送り出されるリクエストの認可には何の役割も果たさないということに注意することが重要である。SIP において認可と認証は、26 節で議論されるように、チャレンジレスポンスメカニズムでリクエストごとに、あるいは下位レイヤスキームを使用して、対応される。

この登録の例についての SIP メッセージの完全なセットの詳細は、24.1 節で述べられている。

OPTIONS を使用して SIP サーバやクライアントの能力を問い合わせたり、CANCEL を使用して保留中のリクエストをキャンセルするといった SIP の付加的なオペレーションは、後の節で紹介される。

５．プロトコルの構造 (Structure of the Protocol)

SIP はレイヤ化されたプロトコルとして構造化されている。これは、その動作が、各ステージ間のゆるい結合のみがある、ほとんど独立したプロセスステージのセットという形で説明されることを意味する。プロトコルの動作は、それを描写するために複数のレイヤとして説明される。これは、一つの節でエレメント間に共通する機能の説明を可能にする。それはどのような形であれ実装に影響しない。エレメントがレイヤを「含む」と言うとき、それはそのレイヤで定義されたルールのセットに従うことを意味する。

プロトコルで規定されたすべてのエレメントが全レイヤを含むわけではない。さらに、SIP で規定されたエレメントは論理的なエレメントであって、物理的なものではない。物理的に実現されたものは、おそらくトランザクション毎であっても別の論理エレメントとして動作することを選択できる。

SIP の最下位のレイヤは、構文と符号化である。符号化は拡張 BNF を使用して指定される。完全な BNF は 25 節で規定されている。SIP メッセージの構造の概要は、7 節で述べられている。

2 番めのレイヤはトランスポートレイヤである。このレイヤは、ネットワーク上でクライアントがどのようにリクエストを送りレスポンスを受け取るか、そしてサーバがリクエストを受け取りレスポンスを送るかを定義する。すべての SIP エレメントはトランスポートレイヤを含む。トランスポートレイヤについては 18

節で述べられている。

3番めのレイヤはトランザクションレイヤである。トランザクションはSIPの基本コンポーネントである。トランザクションは、(トランスポートレイヤを利用して)クライアントトランザクションからサーバトランザクションに送られる一つのリクエストと、サーバトランザクションからクライアントに送り返されるそのリクエストに対するすべてのレスポンスである。トランザクションレイヤは、アプリケーションレイヤの再送、レスポンスのリクエストへのマッチング、およびアプリケーションレイヤのタイムアウトに対応する。ユーザエージェントクライアント(UAC)が成し遂げるどのようなタスクも、一連のトランザクションを使って行われる。トランザクションについての議論は 17 節で述べられている。ユーザエージェントは、ステートフルプロキシでもそうであるように、トランザクションレイヤを含む。ステートレスプロキシはトランザクションレイヤを含まない。トランザクションレイヤはクライアントコンポーネント(クライアントトランザクションと呼ばれる)とサーバコンポーネント(サーバトランザクションと呼ばれる)を持ち、それらは特定のリクエストを処理するために構築される有限ステートマシーンによって表される。

トランザクションレイヤの上にあるレイヤーはトランザクションユーザ(TU)と呼ばれる。ステートレスプロキシを除く各SIPエンティティは、トランザクションユーザである。TUがリクエストを送ろうとするとき、TU はクライアントトランザクションインスタンスを生成し、リクエストを送る先の送信先 IP アドレス、ポート、およびトランスポートと共にリクエストをそれに渡す。クライアントトランザクションを生成する TU は、それをキャンセルすることもできる。クライアントがトランザクションをキャンセルするとき、クライアントはサーバが更なる処理を中止し、トランザクションが開始される前に存在していた状態に復帰し、そのトランザクションに対して特定のエラーレスポンスを生成することを要求する。これは、CANCEL リクエスト(それ自身のトランザクションを設立する)で行われるが、キャンセルされるトランザクションを参照する(9 節参照)。

SIP エレメント、すなわち、ユーザエージェントクライアントとサーバ、ステートレスおよびステートフルプロキシとレジストラサーバは、それらを互いに区別するコアを含む。ステートレスプロキシ以外のコアは、トランザクションユーザである。UAS と UAC のコアの動作はメソッドに依存するとはいえ、すべてのメソッドに対するいくつかの共通のルールがある(8 節参照)。UAC においては、これらのルールはリクエストの構築を律する。UAS においては、それらはリクエストの処理とレスポンスの生成を律する。SIP では登録が重要な役割を演じるので、REGISTER に対応する UAS は、レジストラサーバという特別な名称を与えられている。10 節では、REGISTER メソッドに対する UAC および UAS コアの動作を述べている。11 節では、UA の能力を決定するために使用される、OPTIONS メソッドに対する UAC および UAS コアの動作を述べている。

その他の特定のリクエストはダイアログ内で送られる。ダイアログとは、しばらくの間持続する 2 つのユーザエージェント間のピア・トゥ・ピアの SIP リレーションシップである。ダイアログは、ユーザエージェント間のメッセージの順序付けと、それらの間の適切なルーティングを容易にする。INVITE メソッドは、ダイアログを確立するためにこの仕様で定義された唯一の方法である。UAC がダイアログのコンテキスト内にあるリクエストを送るとき、それは 8 節で議論される UAC の一般ルールに従うが、ダイアログ内リクエスト(mid-dialog request)のルールにも従う。12 節ではダイアログ内でのリクエストの構築に加え、ダイアログについて議論し、その構築手順やメンテナンスについて示す。

SIP において最も重要なメソッドは INVITE メソッドである。INVITE メソッドは参加者間でセッションを確立するために使用される。セッションとは、コミュニケーションを目的とする、参加者の集まり、および

それらの間のメディアの流れである。13 節では、どのようにセッションが開始され、一つ以上の SIP ダイアログが生じるかを議論する。14 節では、ダイアログ内での INVITE リクエストの使用によって、そのセッションの特性がどのように修正されるかを議論する。最後に、15 節では、セッションがどのように終了されるかを議論する。

8、10、11、12、13、14、および 15 節の手順はすべて UA コアを対象とする(9 節は UA コアとプロキシコアの両方に適用する、キャンセルについて述べる)。16 節はプロキシエレメントについて議論する。プロキシエレメントはユーザエージェント間のメッセージのルーティングを容易にする。

６．定義 (Definitions)

以下の用語は SIP において特別な意味を持つ。

Address-of-Record:

Address-of-Record(AOR)とは、SIP または SIPS URI をユーザがいるかもしれない別の URI にマップすることができるロケーションサービスがあるドメインを指し示す､SIP または SIPS URI である。通常、ロケーションサービスは登録によって存在する。AOR はしばしばユーザの「パブリックアドレス」とみなされる。

バック・トゥ・バック ユーザエージェント(Back-to-Back User Agent):

バック・トゥ・バック ユーザエージェント(B2BUA)とは、リクエストを受け取り、ユーザエージェントサーバ(UAS)としてそれを処理する論理的なエンティティである。リクエストにどのように答えるべきか決定するためにユーザエージェントクライアント(UAC)として動作し、リクエストを生成する。プロキシサーバとは違い、ダイアログの状態を保持し、それが確立したダイアログ上で送られるすべてのリクエストに関与しなければならない。UAC と UAS が結合されたものなので、その動作に関する明示的な定義は必要とされない。

呼(Call):

呼とは、通常、マルチメディア会話をするためにセットアップされるピア間のコミュニケーションについて言及する略式の用語である。

コールレグ(Call Leg):

ダイアログの別名(参考文献[31])。本仕様では、もう使用されない。

コールステートフル(Call Stateful):

プロキシは、ダイアログを開始する INVITE から終了する BYE リクエストまでの間ダイアログのステートを保持するとき、コールステートフルである。コールステートフルなプロキシは常にトランザクションステートフルであるが、その逆は必ずしも真ではない。

クライアント(Client):

クライアントとは、SIP リクエストを送り SIP レスポンスを受け取るあらゆるネットワークエレメントのことである。クライアントはユーザ(人間)と直接対話することも、しないこともある。ユーザエージェントクライアントとプロキシはクライアントである。

カンファレンス(Conference):

複数の参加者を含むマルチメディアセッション(下記参照)。

コア(Core):

コアは個々のタイプのSIPエンティティに特有な機能を指定する。すなわち、ステートフルまたはステートレスプロキシに特有な機能、ユーザエージェントまたはレジストラサーバに特有な機能。ステートレスプロキシのコア以外のすべてのコアはトランザクションユーザである。

ダイアログ(Dialog):

ダイアログとは、しばらくの間持続する2つのUA間のピア・トゥ・ピアのSIPリレーションシップである。ダイアログは、INVITEリクエストに対する2xxレスポンスのようなSIPメッセージによって確立される。ダイアログは呼識別子(call identifier)、ローカルtag、およびリモートtagによって識別される。ダイアログは、以前はRFC2543においてコールレグとして知られていた。

ダウンストリーム(Downstream):

ユーザエージェントクライアントからユーザエージェントサーバへのリクエストの流れの向きについて言及するときの、トランザクション内をフォワードされるメッセージの向き。

最終レスポンス(Final Response):

SIPトランザクションを終了しない暫定レスポンス(Provisional Response)とは逆に、SIPトランザクションを終了するレスポンス。2xx、3xx、4xx、5xx、および6xxはすべて最終レスポンス(Final response)である。

ヘッダ(Header):

ヘッダとは、SIPメッセージについての情報を伝えるSIPメッセージの構成要素である。ヘッダはヘッダフィールドのつながりとして構築される。

ヘッダフィールド(Header Field):

ヘッダフィールドとは、SIPメッセージヘッダの構成要素である。ヘッダフィールドは1個以上のフィールド列として出現しうる。ヘッダフィールド列はヘッダフィールド名と、0個以上のヘッダフィールド値で構成される。特定のヘッダフィールド列上の複数のヘッダフィールド値はカンマで区切られる。いくつかのヘッダフィールドは1つのヘッダフィールド値しか持つことができないので、結果として、1つのヘッダフィールド列として現れる。

ヘッダフィールド値(Header Field Value):

ヘッダフィールド値とは、一つの値である。ヘッダフィールドは0個以上のヘッダフィールド値で構成される。

ホームドメイン(Home Domain):

SIPユーザにサービスを提供するドメイン。通常これは、登録のAddress-of-Record(AOR)内のURIに存在するドメインである。

通知レスポンス(Informational Response):

レスポンスと同じ。

起動者(Initiator)、発信側パーティ(Calling Party)、発信者(Caller):

INVITE リクエストでセッション(およびダイアログ)を開始するパーティ。発信者は、ダイアログを確立した最初の INVITE を送ってからそのダイアログの終了まで、この役割を維持する。

招待:

INVITE リクエストのことである。

招待された側、招待されたユーザ、着信側パーティ(Called Party)、着信者(Callee):

新たなセッションを確立するための INVITE リクエストを受け取るパーティ。着信者は、INVITE を受け取ってからその INVITE で確立されたダイアログの終了まで、この役割を維持する。

ロケーションサービス(Location Service):

ロケーションサービスは、着信者のいる可能性のあるロケーション(一つまたは複数)を取得するために、SIP リダイレクトサーバあるいはプロキシサーバによって使用される。ロケーションサービスは Address-of-Record(AOR)キーとゼロ個以上のコンタクトアドレスのバインディングリストを含む。バインディングは様々な方法によって生成、削除できる。この仕様では、バインディングをアップデートする REGISTER メソッドを定義する。

ループ(Loop):

プロキシに到達し、フォワードされ、その後しばらくしてから同じプロキシに戻ってくるリクエスト。それがプロキシに 2 度目に到達するとき、それの Request-URI は初回時と同じであり、プロキシのオペレーションに影響を及ぼすその他のヘッダフィールドは変更されていないため、プロキシはそのリクエストに対して初回時と同じ処理決定を下すことになる。ループしたリクエストはエラーであり、それを検出しハンドリングする手順はプロトコルによって記述される。

ルースルーティング(Loose Routing):

プロキシが Route ヘッダフィールドの処理のために、この仕様で定義されている手順に従う場合、そのプロキシはルースルーティングであるといわれる。この手順は、送信先に向かう途中で訪れる必要があるプロキシのセット(Route ヘッダフィールド中に存在する)から、リクエストの送信先(Request-URI 中に存在する)を分離する。これらのメカニズムに準拠するプロキシはルースルータ(Loose Router)としても知られている。

メッセージ(Message):

プロトコルの一部として SIP エレメント間で送られるデータ。SIP メッセージはリクエストかレスポンスのいずれかである。

メソッド(Method):

メソッドとは、サーバ上でリクエストが呼び出さることを意味する、主要な機能である。メソッドはリクエストメッセージ自身によって伝えられる。メソッドの例として、INVITE と BYE がある。

アウトバウンドプロキシ(Outbound Proxy):

クライアントからのリクエストを受け取るプロキシ(たとえそれが Request-URI で解決されたサーバでなくても)。通常、UA にはマニュアルでアウトバウンドプロキシが設定される。あるいは自動設定プロトコルによってアウトバウンドプロキシを知ることができる。

パラレルサーチ(Parallel Search):

パラレルサーチでは、送られてくるリクエストを受け取ると、プロキシはユーザのいると思われる複数のロケーションにいくつかのリクエストを発行する。シーケンシャルサーチのように一つのリクエストを発行して、最終レスポンス(Final response)を待ってから次のリクエストを発行するのではなく、パラレルサーチでは、すでに発行したリクエストの結果を待たずに複数のリクエストを発行する。

暫定レスポンス(Provisional Response):

進行状況を示すためにサーバが使用するレスポンス。このレスポンスは SIP トランザクションを終了しない。1xx レスポンスは暫定レスポンス(provisional response)であり、その他のレスポンスは最終レスポンス(Final response)とみなされる。

プロキシ(Proxy)、プロキシサーバ(Proxy Server):

他のクライアントに替わりリクエストを生成するために、サーバやクライアントの両方として動作する中間エンティティ。プロキシサーバは、主としてルーティングの役割を演じる。これは、リクエストが、ターゲットユーザに「より近い」他のエンティティに送られることを保証することが、プロキシサーバの役目であることを意味する。プロキシは、ポリシーを強制するためにも有用である(たとえば、ユーザが電話をかけることを許可されていることを確認すること)。プロキシは、リクエストメッセージをフォワードする前に、それを解釈し、必要であればその特定部分を書き換える。

再帰(Recursion):

クライアントが 3xx レスポンスの Contact ヘッダフィールドに含まれる一つ以上の URI に対して新規リクエストを生成するとき、クライアントは 3xx レスポンスで再帰するという。

リダイレクトサーバ:

リダイレクトサーバとは、受け取ったリクエストに対して 3xx レスポンスを生成するユーザエージェントサーバである。そうすることで、別の URI セットにコンタクトすることをクライアントに指示する。

レジストラサーバ(Registrar):

レジストラサーバとは、REGISTER リクエストを受け入れ、それがハンドリングするドメインのロケーションサービスに、そのリクエストで受け取った情報を置くサーバである。

レギュラートランザクション(Regular Transaction):

レギュラートランザクションとは、INVITE、ACK、または CANCEL 以外のメソッドを含むすべてのトランザクションである。

リクエスト(Request):

特定のオペレーションを実行するために、クライアントからサーバに送られたSIPメッセージのことである。

レスポンス(Response):

クライアントからサーバに送られたリクエストのステータスを示すために、サーバからクライアントに送られたSIPメッセージのことである。

Ringback:

ringbackとは、発信側パーティのアプリケーションが作り出す、着信側パーティが呼び出されていることを示す呼び出し音である。

ルートセット(Route Set):

ルートセットとは、特定のリクエストを送るときにトラバースされなければならないプロキシのリストを表す、SIPまたはSIPS URIの並びの集まりである。ルートセットはRecord-Routeのようなヘッダから知ることができるし、あるいは、設定することもできる。

サーバ(Server):

サーバとは、リクエストを受信処理し、そのリクエストにレスポンスを送り返すためのネットワークエレメントである。サーバの例として、プロキシ、ユーザエージェントサーバ、リダイレクトサーバ、およびレジストラサーバがある。

シーケンシャルサーチ(Sequential Search):

シーケンシャルサーチでは、プロキシサーバは各コンタクトアドレスを順番に試す。すでに試したコンタクトアドレスが最終レスポンス(Final response)を生成した後にのみ次のコンタクトアドレスに進む。2xxまたは6xxクラスの最終レスポンス(Final response)は常にシーケンシャルサーチを終了する。

セッション(Session):

SDP仕様によれば「マルチメディアセッションとは、マルチメディアを送る側と受ける側および送り手から受け手に流れるデータストリームのセットのことである。マルチメディアカンファレンスは、マルチメディアセッションの一例である。」(RFC2327参考文献[1])(SDPで定義されたセッションは、一つ以上のRTPセッションから成る)。この定義のように、着信者を(異なる呼によって)同じセッションに何回も招待可能である。SDPが使用される場合、セッションは、SDPユーザ名、セッションID、ネットワークタイプ、アドレスタイプ、およびoriginフィールドのアドレスエレメントを結合したもので定義される。

SIPトランザクション(SIP Transaction):

SIPトランザクションはクライアントとサーバの間で発生し、クライアントからサーバに送られた最初のリクエストから、サーバからクライアントに送られた最終レスポンス(Final response)(非1xx)までのすべてのメッセージからなる。リクエストがINVITEで最終レスポンス(Final response)が非2xxの場合、トランザクションはそのレスポンスに対するACKも含む。INVITEリクエストに対する2xxレスポンスへのACKは、別のトランザクションである。

スパイラル(Spiral):

スパイラルとは、プロキシにルーティングされてからフォワードされ、再び同じプロキシに戻ってくるSIPリクエストのことである。しかし今度は、オリジナルリクエストとは違う処理決定を下されることになるという違いがある。通常これは、そのリクエストが前回の到着時とは異なるRequest-URIを持つことを意味する。ループとは違い、スパイラルはエラー状態ではない。スパイラルの典型的な原因は、着信転送(call forwarding)である。ユーザがjoe@example.comを呼び出す。example.comのプロキシはそれをJoeのPCにフォワードし、今度はそのPCがそれをbob@example.comにフォワードする。このリクエストはexample.comのプロキシにプロキシされて戻される。しかしながら、これはループではない。リクエストが別のユーザに向けられているので、スパイラルとみなされ、有効な状態である。

ステートフルプロキシ(Stateful Proxy):

リクエストを処理する間、この仕様で定義されているクライアントトランザクションおよびサーバトランザクションのステートマシーンを維持する、論理的なエンティティであり、トランザクションステートフルプロキシとしても知られている。ステートフルプロキシの動作は、16節でさらに詳しく定義されている。(トランザクション)ステートフルプロキシは、コールステートフルプロキシと同じではない。

ステートレスプロキシ(Stateless Proxy):

リクエストを処理するときに、この仕様で定義されているクライアントトランザクションあるいはサーバトランザクションのステートマシーンを維持しない、論理的なエンティティ。ステートレスプロキシは、受け取るすべてのリクエストをダウンストリームに、および受け取るすべてのレスポンスをアップストリームにフォワードする。

ストリクトルーティング(Strict Routing):

プロキシがRFC2543および本RFC策定中のルーティング処理ルールに従う場合、そのプロキシはストリクトルーティングであるといわれる。そのルールは、Routeヘッダフィールドが存在したときに、プロキシにRequest-URIのコンテンツを破棄させていた。ルースルーティング動作を優先して、この仕様ではストリクトルーティング動作は使用されない。ストリクトルーティングを実行するプロキシはストリクトルータ(Strict Router)としても知られている。

ターゲットリフレッシュリクエスト(Target Refresh Request):

ダイアログ内で送られたターゲットリフレッシュリクエストは、ダイアログのリモートターゲットを修正することができるリクエスト、と定義される。

トランザクションユーザ(Transaction User、TU):

トランザクションレイヤの上にあるプロトコル処理のレイヤ。トランザクションユーザは、UACコア、UASコア、およびプロキシコアを含む。

アップストリーム(Upstream):

ユーザエージェントサーバからユーザエージェントクライアントに戻るレスポンスの流れの向きについて言及するときの、トランザクション内をフォワードされるメッセージの向きである。

URL エンコードされた(URL-encoded):

RFC2396 の 2.4 節(参考文献[5])に則ってエンコードされた文字列。

ユーザエージェントクライアント(User Agent Client、UAC):

ユーザエージェントクライアントとは、新しいリクエストを生成し、それを送るためにクライアントトランザクションのステートマシーンを使用する、論理的なエンティティ。UAC の役目は、そのトランザクションが継続する間だけ存続する。言い換えるなら、あるソフトウェアがリクエストを開始した場合、それは、そのトランザクションが継続する間 UAC として動作する。その後それがリクエストを受け取った場合、それはそのトランザクション処理のためにユーザエージェントサーバの役目を果たす。

UAC コア(UAC Core):

トランザクションレイヤとトランスポートレイヤの上にある、UAC に必要とされる処理機能のセット。

ユーザエージェントサーバ(User Agent Server、UAS):

ユーザエージェントサーバとは、SIP リクエストに対してレスポンスを生成する論理的なエンティティである。レスポンスは、リクエストを受け入れる、拒否する、あるいはリダイレクトする。この役目は、そのトランザクションが継続する間だけ存続する。言い換えるなら、あるソフトウェアがリクエストにレスポンスする場合、それは、そのトランザクションが継続する間 UAS として動作する。その後それがリクエストを生成した場合、それはそのトランザクション処理のためにユーザエージェントクライアントの役目を果たす。

UAS コア(UAS Core):

トランザクションレイヤとトランスポートレイヤの上にある、UAS に必要とされる処理機能のセット。

ユーザエージェント(User Agent、UA):

ユーザエージェントクライアントとしてもユーザエージェントサーバとしても動作できる、論理的なエンティティ。

プロキシサーバおよびリダイレクトサーバと同様に、UAC と UAS の役目はトランザクションごとに定義される。たとえば、呼を開始するユーザエージェントは、最初の INVITE リクエストを送るときは UAC として動作し、着信者から BYE リクエストを受け取るときは UAS として動作する。同様に、同一のソフトウェアが、あるリクエストにはプロキシサーバとして動作し、その次のリクエストにはリダイレクトサーバとして動作することも可能である。

上で定義されたプロキシサーバ、ロケーションサーバ、およびレジストラサーバは、論理的なエンティティである。実装ではそれらを一つのアプリケーションに結合してもよい[MAY]。

７．SIP メッセージ (SIP Messages)

SIP はテキストベースのプロトコルであり、UTF-8 文字セット(RFC2279 参考文献[7])を使用する。

SIP メッセージは、クライアントからサーバへのリクエストまたはサーバからクライアントへのレスポンスのいずれかである。

リクエストメッセージ(7.1 節)とレスポンスメッセージ(7.2 節)は共に、文字セットと構文詳細において構文は異なるが、RFC2822(参考文献[3])の基本フォーマットを使用する。(たとえば、SIP は RFC2822 のヘッダフィールドでは有効でないかもしれないヘッダフィールドを許可する) 双方のメッセージタイプは、開始行(start-line)、一つ以上のヘッダフィールド、ヘッダフィールドの終了を示す空行、およびオプションの message-body から成る。

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

開始行、各メッセージヘッダ行、および空行は、キャリッジリターン・ラインフィードの並び(CRLF)で終わらなくてはならない[MUST]。message-body が存在しないときでも、空行がなくてはならない[MUST]ことに注意すること。

上記の文字セットの違いを除いて、SIP のメッセージ構文とヘッダフィールド構文のほとんどは、HTTP/1.1 と同一である。ここで構文と意味合い(semantics)を繰り返すかわりに、現在の HTTP/1.1 仕様(RFC2616 参考文献[8])の X.Y 節を参照するために[HX.Y]という表記を使用する。

しかしながら、SIP は HTTP の拡張ではない。

7.1 リクエスト (Requests)

SIP リクエストは開始行に Request-Line を持つことで、他と区別される。Request-Line は一つの空白文字(SP)で区切られた、メソッド名、Request-URI、プロトコルバージョンを含む。

Request-Line は CRLF で終わる。行末の CRLF の並びを除いて、CR や LF は認められていない。どのエレメントにもリニアホワイトスペース(LWS)は認められていない。

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

メソッド:

この仕様では 6 つのメソッドを定義する。コンタクトインフォメーションを登録するための REGISTER、セッションをセットアップするための INVITE、ACK、CANCEL、セッションを終了するための BYE、およびサーバにその能力を問い合わせるための OPTIONS である。standards track RFC でドキュメント化されている SIP 拡張は、付加的なメソッドを定義するかもしれない。

Request-URI:

Request-URI は、19.1 節で述べられている SIP URI または SIPS URI、あるいは通常の URI(RFC2396 参考文献[5])である。Request-URI は、このリクエストが宛てられたユーザやサービスを示す。

Request-URI はエスケープされていないスペースやコントロール文字を含んではいけなく[MUST NOT]、「<>」で囲んでもいけない[MUST NOT]。

SIP エレメントは、たとえば RFC2806(参考文献[9])の「tel」URI スキームのような「sip」および「sips」以外のスキームの Request-URI をサポートしてもよい[MAY]。SIP エレメントは、結果的に SIP URI、SIPS URI またはその他のスキームのどれかになるどのようなメカニズムでも自由に使用して、非 SIP URI を解釈してもよい[MAY]。

SIP-Version:

リクエストメッセージとレスポンスメッセージは共に使用中の SIP のバージョンを含み、バージョンの順序付け、承諾条件、およびバージョン番号のアップグレードに関して[H3.1]に従う(HTTP を SIP に、HTTP/1.1 を SIP/2.0 に置き換えて)。この仕様に準拠するために、SIP メッセージを送るアプリケーションは、SIP-Version として「SIP/2.0」を含まなくてはならない[MUST]。SIP-Version の文字列は大文字小文字を区別しないが、実装では大文字で送らなくてはならない[MUST]。

HTTP/1.1 とは違い、SIP はバージョン番号をリテラル文字列として扱う。実際には、この点はたいして影響を与えないと思われる。

7.2 レスポンス (Responses)

SIP のレスポンスは、開始行に Status-Line を持つことで、リクエストと区別される。 Status-Line は、プロトコルバージョンとそれに続く Status-Code(数値)とそれに関連付けられたテキストフレーズから成る。各エレメントは SP 文字で区切られる。

行末の CRLF の並びを除いて、CR や LF は認められていない。

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

Status-Code は、リクエストを理解し満たそうとした結果を示す 3 桁の整数からなる結果コードである。リーズンフレーズは、Status-Code にテキストによる短い説明を与えることを意図している。Status-Code がオートマトンによる使用を意図しているのに対して、リーズンフレーズは人間のユーザによる使用を目的としている。クライアントはリーズンフレーズを検査したり表示したりすることを要求されない。

この仕様はリーズンフレーズのための特定の文章を提示するが、実装では、たとえばリクエストの Accept-Language ヘッダフィールドで示されている言語で、その他の文章を選択してもよい[MAY]。

Status-Code の最初の桁はレスポンスのクラスを定義する。最後の 2 桁にはカテゴリ分けの役目はない。このため、100 から 199 の間のどのステータスコードをもつレスポンスも「1xx レスポンス]、200 から 299 の間のどのステータスコードを持つレスポンスも「2xx レスポンス」(以下同様)、と呼ばれる。SIP/2.0 では最初の桁に 6 つの値が許可されている。

1xx: 暫定 -- リクエストが受信され、そのリクエストの処理を継続している。

2xx: 成功 -- アクションの受信が成功し、理解され、受け入れられた。

3xx: リダイレクト -- リクエストを完了するために、更なるアクションを取る必要がある。

4xx: クライアントエラー -- リクエストが誤りのある構文を含む、またはこのサーバではそのリクエストを実行できない。

5xx: サーバエラー -- サーバが、明らかに有効なリクエストの実行に失敗した。

6xx: グローバルな失敗 -- リクエストはどのサーバでも実行できなかった。

21 節で、これらのクラスを定義し、それぞれのコードについて述べる。

7.3 ヘッダフィールド (Header Fields)

SIP のヘッダフィールドは、構文と意味合い(semantics)の双方において HTTP のヘッダフィールドとよく似ている。特に、SIP のヘッダフィールドは[H4.2]の、message-header の構文定義、および複数行に渡るヘッダフィールドのルールに従う。しかしながら、後者は HTTP では、暗黙的なホワイトスペースと折りたたみ(implicit whitespace and folding)とともに規定されている。この仕様では、RFC2234(参考文献[10])に準拠し、記述法に不可欠な部分として、明示的なホワイトスペースと折りたたみ (explicit whitespace and folding)のみを使用する。

[H4.2]では、同じヘッダフィールド名でカンマ区切りリストの値を持つ複数のヘッダフィールドが、一つのヘッダフィールドに結合できることも規定している。それは SIP にも適用されるが、記述法が異なるので細かいルールは異なる。具体的には、以下の形式の記述法の SIP ヘッダはすべて、

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

同じ名称のヘッダフィールドをカンマ区切りリストで結合することが認められている。Contact ヘッダフィールドは、そのヘッダフィールド値が「*」でない限り、カンマ区切りのリストが認められている。

7.3.1 ヘッダフィールドのフォーマット (Header Fields Format)

ヘッダフィールドは、RFC2822(参考文献[3])の 2.2 節で与えられているものと同じ一般的なヘッダのフォーマットに従う。各ヘッダフィールドは、フィールド名とそれに続くコロン(:)およびフィールド値から成る。

```
field-name: field-value
```

25 節で規定されている message-header の正式な記述法では、コロンの前後に任意の数のホワイトスペースを認めている。しかしながら、実装では、フィールド名とコロンの間にスペースを入れず、コロンとフィールド値の間にスペースを 1 個(SP)使用するべきである。

```
Subject:           lunch
Subject     :      lunch
Subject           :lunch
```

```
Subject: lunch
```

すなわち、上記はすべて有効で等価であるが、最後のものが好ましい形式である。

ヘッダフィールドは、2 行目以降の先頭に少なくとも 1 つの SP または水平タブ(HT)を置くことで、複数行に渡って記述できる。行が中断される個所と次の行の先頭にあるホワイトスペースは 1 つの SP 文字として扱われる。そのため、次の 2 つは等価である。

```
Subject: I know you're there, pick up the phone and talk to me!
Subject: I know you're there,
         pick up the phone
         and talk to me!
```

異なるフィールド名のヘッダフィールドどうしの相対的な並び順は重要ではない。しかしながら、プロキシ処理に必要とされるヘッダフィールド(たとえば、Via、Route、Record-Route、Proxy-Require、Max-Forwards、および Proxy-Authorization)は、高速なパーシングを容易にするためにメッセージの先頭のほうに現れるようにすることが推奨される[RECOMMENDED]。同じフィールド名のヘッダフィールド列どうしの相対的な並び順は重要である。同一フィールド名の複数のヘッダフィールド列は、それらのヘッダフィールドのフィールド値全体がカンマ区切りリストとして(すなわち、7.3 節で定義されている記述法に従う場合)定義されているときに限り、一つのメッセージ中に存在してもよい[MAY]。最初のフィールド値にそのあとの各フィールド値をカンマで区切ってアペンドし、メッセージの意味合い(semantics)を変えずに、複数のヘッダフィールド列を一つの「field-name: field-value」のペアに結合することが可能でなくてはならない[MUST]。このルールの例外は、WWW-Authenticate、Authorization、Proxy-Authenticate、および Proxy-Authorization ヘッダフィールドである。これらのフィールド名を持つ複数のヘッダフィールド列は、メッセージ中に存在してもよい[MAY]が、それらの記述法は 7.3 節にリストされている一般的な形式に従わないので、それらは一つのヘッダフィールド列に結合されてはいけない[MUST NOT]。

実装では、1 行に 1 つの値を持つ形式あるいはカンマ区切り値形式のどのような組み合わせによるものであれ、同じフィールド名を持つ複数のヘッダフィールド列を処理できなくてはならない[MUST]。

以下のヘッダフィールド列のグループは有効であり、かつ等価である。

```
Route: <sip:alice@atlanta.com>
Subject: Lunch
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>
Subject: Lunch

Subject: Lunch
Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>, <sip:carol@chicago.com>
```

以下の各ブロックは有効であるが、等価ではない。

Route: <sip:alice@atlanta.com>
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>

Route: <sip:bob@biloxi.com>
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>,<sip:carol@chicago.com>,<sip:bob@biloxi.com> ヘッダフィールド値のフォーマットは、ヘッダ名ごとに定義される。それは常に、TEXT-UTF8 オクテットの意味不明なシーケンスか、あるいはホワイトスペース、トークン、分離記号、引用符で囲まれた文字列の組み合わせのいずれかである。多くの既存のヘッダフィールドは、値の一般的な形式とセミコロンで区切られてそれに続くパラメータ名(parameter-name)とパラメータ値(parameter-value)のペアのシーケンスである。

field-name: field-value *(;parameter-name=parameter-value)

ヘッダフィールド値には任意の個数のパラメータペアを付けることができるが、与えられるどのパラメータ名も 2 回以上現れてはいけない[MUST NOT]。

ヘッダフィールドを比較するときは、フィールド名は常に大文字小文字を区別しない。特定のヘッダフィールドの定義で特に指定されない限り、フィールド値、パラメータ名、パラメータ値は大文字小文字を区別しない。トークンは常に大文字小文字を区別しない。特に規定がなければ、引用符で囲まれた文字列として表現された値は大文字小文字を区別する。たとえば、

Contact: <sip:alice@atlanta.com>;expires=3600

は以下と等価である。

CONTACT: <sip:alice@atlanta.com>;ExPiReS=3600

また

Content-Disposition: session;handling=optional

は以下と等価である。

content-disposition: Session;HANDLING=OPTIONAL

以下の 2 つのヘッダフィールドは等価ではない。

```
Warning: 370 devnull "Choose a bigger pipe"
Warning: 370 devnull "CHOOSE A BIGGER PIPE"
```

7.3.2 **ヘッダフィールドの分類** (Header Field Classification)

いくつかのヘッダフィールドはリクエストの中あるいはレスポンスの中でのみ意味をなす。これらはそれぞれ、リクエストヘッダフィールド、レスポンスヘッダフィールドと呼ばれる。ヘッダフィールドがそのカテゴリーにマッチしないメッセージ中に現れた場合(たとえば、リクエストヘッダフィールドがレスポンス中に)、それは無視されなくてはならない[MUST]。20 節で、各ヘッダの分類を定義している。

7.3.3 **コンパクトフォーム** (Compact Form)

SIP は、一般的なヘッダフィールド名を省略された形で表すメカニズムを提供する。これは、メッセージが利用可能なトランスポート上で配送するには大きくなりすぎる可能性があるときに有用である(たとえば、UDP を使用するときに MTU を超える)。これらのコンパクトフォームは 20 節で定義されている。メッセージの意味合い(semantics)を変えることなく、いつでも長い形式のヘッダ名をコンパクトフォームで置き換えてもよい[MAY]。ヘッダフィールド名は同一のメッセージ中に、長い形式と短い形式の両方で現れてもよい[MAY]。実装では、各ヘッダ名の長短の形式を双方共に受け入れなくてはならない[MUST]。

7.4 **ボディ** (Bodies)

この仕様の拡張で定義される新規リクエストを含むすべてのリクエストは、特に断りのない限り、メッセージボディを含んでもよい[MAY]。ボディの解釈はリクエストのメソッドに依存する。

レスポンスメッセージでは、リクエストのメソッドとレスポンスのステータスコードが、あらゆるメッセージボディのタイプと解釈を決定する。すべてのレスポンスはボディを含んでもよい[MAY]。

7.4.1 **メッセージボディのタイプ** (Message Body Type)

メッセージボディのインターネットメディアタイプは、Content-Type ヘッダフィールドで与えられなくてはならない[MUST]。もしボディが何らかのエンコーディングを施されていた場合(たとえば圧縮など)、このことは Content-Encoding ヘッダフィールドで示されなくてはならない[MUST]。そうでない場合、Content-Encoding は省略されなくてはならない[MUST]。妥当な場合には、メッセージボディの文字セットが Content-Type のヘッダフィールド値の一部として示される。

RFC2046(参考文献[11])で定義されている「multipart」という MIME タイプは、メッセージのボディ中で使用してもよい[MAY]。マルチパートのメッセージボディを含むリクエストを送る実装は、セッション記述を非マルチパートのメッセージボディで送らなくてはならない[MUST](リモートの実装が、マルチパートを含まない Accept ヘッダフィールドでこれを要求する場合)。

SIP のメッセージはバイナリのボディまたはボディの一部を含んでもよい[MAY]。明示的な文字セットのパラメータが送信側から提示されない場合、「text」タイプのメディア・サブタイプは UTF-8」の文字セットを初期値として保持する。

7.4.2 **メッセージボディの長さ** (Message Body Length)

ボディ長(バイト)は Content-Length ヘッダフィールドで提供される。20.14 節では、このヘッダに必要なコンテンツについて詳細に述べている。

HTTP/1.1 の「chunked」トランスファーエンコーディングは、SIP で使用してはいけない[MUST NOT]。(注意:chunked エンコーディングは、連続した chunk としてボディをトランスファーするために(各 chunk はそれ自身のサイズインジケーターを持つ)、メッセージのボディを修正する。)

7.5 SIP メッセージのフレーム化 (Framing SIP Messages)

HTTP とは違い、SIP の実装は UDP あるいはその他の信頼性の低いデータグラムプロトコルを使用してもよい[MAY]。そのようなデータグラムはそれぞれ一つのリクエストまたはレスポンスを伝える。信頼性の低いトランスポートの使用時の制約については 18 節参照のこと。

ストリーム指向のトランスポート上で SIP メッセージを処理する実装は、開始行[H4.1]の前に現れるいかなる CRLF も無視しなくてはならない[MUST]。

Content-Length ヘッダフィールド値は、ストリーム中で各 SIP メッセージの終わりの位置を特定するために使用される。それは、SIP メッセージがストリーム指向のトランスポート上で送られるときにはいつでも存在する。

８．ユーザエージェントの一般的な動作 (General User Agent Behavior)

ユーザエージェントはエンドシステムに相当する。それは、リクエストを生成するユーザエージェントクライアント(UAC)およびそれらのリクエストにレスポンスするユーザエージェントサーバ(UAS)を含む。UAC は、外部からの刺激(ユーザのボタンクリック、または PSTN ライン上の信号)に基づいてリクエストを生成したり、レスポンスを処理することができる。UAS は、リクエストを受け取り、ユーザ入力、外部からの刺激、プログラム実行結果、あるいはその他のメカニズムに基づいて、レスポンスを生成することができる。

UAC がリクエストを送るとき、それはいくつかのプロキシサーバを通過する。プロキシサーバは UAS にそのリクエストをフォワードする。UAS がレスポンスを生成するとき、そのレスポンスは UAC にフォワードされる。

UAC および UAS の処理手順は二つのファクターに強く依存している。一つは、リクエストまたはレスポンスがダイアログの内部にあるか外部にあるかということ。二つ目は、リクエストのメソッドに基づくということ。ダイアログについては 12 節で十分に論じる。それらは、ユーザエージェント間のピア・トゥ・ピア リレーションシップに相当し、INVITE のような特定の SIP メソッドによって確立される。

この節では、ダイアログの外部のリクエストを処理するときの UAC および UAS の動作のための、メソッドに依存しないルールについて論じる。これは当然、それ自身でダイアログを確立するリクエストを含む。

ダイアログ外のリクエストとレスポンスのためのセキュリティ処理手順は 26 節で述べられている。特に、UAS と UAC が相互に認証するために存在するメカニズムについてである。プライバシー機能の限定されたセットも S/MIME を用いてボディを暗号化することでサポートされている。

8.1 UAC の動作 (UAC Behavior)

この節では、ダイアログ外の UAC の動作を取り上げる。

8.1.1 **リクエストの生成** (Generating the Request)

UAC によって作成された有効な SIP リクエストは、少なくとも次のヘッダフィールドを含まなくてはならない[MUST]。To、From、CSeq、Call-ID、Max-Forwards、および Via。これらすべてのヘッダフィールドは、全 SIP リクエストに必須である。これらの 6 つのヘッダフィールドは、SIP メッセージの基本的な構成単位である。その理由は、それらが連帯して、メッセージのアドレッシング、レスポンスのルーティング、メッセージの伝達(propagation)の制限、メッセージの順番付け、およびトランザクションの一意な識別を含む、重要なメッセージルーティングサービスの大半を提供するからである。これらのヘッダフィールドは、メソッド、Request-URI、および SIP バージョンを含む必須のリクエスト行(request line)に加える。

ダイアログの外部で送られるリクエストの例には、セッションを確立する INVITE(13 節)および能力を問い合わせる OPTIONS(11 節)が含まれる。

8.1.1.1 Request-URI

メッセージの最初の Request-URI には、To フィールドの URI の値が設定されるべきである[SHOULD]。注目すべき一つの例外は、REGISTER メソッドである。REGISTER の Request-URI を設定する動作は 10 節で示される。これらのフィールドに同じ値を設定することは、プライバシーのためあるいは利便性のために望ましくないかもしれない(特に発信元の UA の Request-URI が、処理中に変更されることを期待する場合)。

ある特別な状況下では、既存のルートセットがメッセージの Request-URI に影響を及ぼすことがある。pre-existing ルートセットとは、UAC が外に向けて送り出すダイアログ外リクエストの宛先となる、一連のサーバを識別する順番に並べられた URI セットである。一般に、それらは UA 上でユーザまたはサービスプロバイダによって、マニュアルあるいは他の非 SIP メカニズムで設定される。プロバイダーが UA に アウトバウンドプロキシを設定することを望む場合、それに一つの URI(アウトバウンドプロキシのもの)を持つ既存のルートセットを提供することで、それを行うことが推奨される[RECOMMENDED]。

既存のルートセットが存在する場合、(たとえダイアログがなくても、)リモートのターゲット URI として望ましい Request-URI を使用して、12.2.1.1 節に詳述されている Request-URI と Route ヘッダフィールドを組み込むための手順に従わなくてはならない[MUST]。

8.1.1.2 To 24

To ヘッダフィールドは、まず、希望するリクエストの「論理的」な受信者、またはこのリクエストのターゲットであるユーザあるいはリソースの Address-of-Record(AOR)を指定する。これは、リクエストの最終的な受信者であるかもしれないし、そうでないかもしれない。To ヘッダフィールドは、SIP URI または SIPS URI を含んでもよい[MAY]が、妥当なときには、その他の URI スキーム(たとえば、tel URL(RFC2806 参考文献[9]))を使用することもできる。すべての SIP 実装は、SIP URI スキームをサポートしなくてはならない[MUST]。TLS をサポートするいかなる実装も、SIPS URI スキームをサポートしなくてはならない[MUST]。To ヘッダフィールドは、ディスプレイネームを考慮に入れている。

UAC は、ある特定のリクエストのための To ヘッダフィールドをどのように組み込むか、様々な方法で知ることができる。通常、ユーザがヒューマンインターフェースを介して(おそらく、URI をマニュアルで入力するか、ある種のアドレス帳から選択して)、To ヘッダフィールドを提示する。高い頻度で、ユーザは完全な URI ではなく数字や文字の列を入力するだろう(たとえば、「bob」)。この入力をどのように解釈するかは UA の判断による。その文字列を SIP URI のユーザ部分を形成するために使用するということは、UA が名前

をドメイン内で解決して SIP URI のアットマークの右側を得ることを望むことを暗に示す(たとえば、sip:bob@example.com)。その文字列を SIPS URI のユーザ部分を形成するために使用するということは、UA が、通信を安全に行い、名前がドメイン内で解決されてアットマークの右側が得られることを望むことを暗に示す。アットマークの右側は高い頻度でリクエスト送信者のホームドメインであるだろう。このことは、そのホームドメインが外に送られるリクエストを処理することを認める。これは、ホームドメイン内でユーザ部分の解釈が求められる、「スピードダイアル」のような機能に対して有用である。　ユーザによって入力された電話番号を解釈すべきドメインを指定することを UA が望まないとき、tel URL が使用されるかもしれない。そうすることで、リクエストが通過する各ドメインがその機会を与えられる。たとえば、空港にいるユーザは、空港のアウトバウンドプロキシにログインして、それを通してリクエストを送るかもしれない。そのユーザが「411」(これはアメリカのローカル電話番号案内の番号である)を入力する場合、それは、ユーザのホームドメインではなく、空港のアウトバウンドプロキシによって解釈され処理される必要がある。この場合、「tel:411」が正しい選択である。

ダイアログ外のリクエストは、To タグを含んではいけない[MUST NOT](リクエストの To フィールドのタグは、ダイアログの相手を識別する)。ダイアログが確立されていないので、タグは存在しない。

To ヘッダフィールドについての詳細は 20.39 節を参照のこと。以下は、有効な To ヘッダフィールドの例である。

```
To: Carol <sip:carol@chicago.com>
```

8.1.1.3 From

From ヘッダフィールドは、リクエストの起動者の論理的なアイデンティティを示す。これはおそらく、ユーザの address-of-record(AOR)であろう。To ヘッダフィールドと同じように、URI およびオプションとしてディスプレイネームを含む。それは、リクエストに適用するための処理ルールを決定するために、SIP エレメントによって使用される(たとえば、自動通話拒否)。このような場合、IP アドレスやホストの FQDN は論理的な名前ではないので、From の URI がこれらを含まないことが非常に重要である。

From ヘッダフィールドはディスプレイネームを考慮している。クライアントのアイデンティティを隠したままにする場合、UAC は「Anonymous」というディスプレイネームを、構文的に正しいが意味を持たない(sip:thisis@anonymous.invalid のような)URI と共に使用するべきである[SHOULD]。

通常、ある特定の UA によって生成されたリクエスト中の From ヘッダフィールドに存在する値は、ユーザあるいはユーザのローカルドメインの管理者によって、あらかじめ用意されている。ある特定の UA が複数のユーザに使用される場合、それは、ユーザのアイデンティティに対応した URI を含む、切り替え可能なプロファイルを持っているかもしれない。リクエストの受信者は、リクエストの発信元が、その From ヘッダフィールドで主張している者であるということを確かめるために、リクエストの発信元を認証することができる(認証に関する詳細は、22 節参照のこと)。

From フィールドは、UAC が選んだ新しいタグを含まなくてはならない[MUST]。タグの選択について詳しくは、19.3 節参照のこと。

From ヘッダフィールドに関しての詳細は、20.20 節参照のこと。

例:

From: "Bob" <sips:bob@biloxi.com> ;tag=a48s

From: sip:+12125551212@phone2net.com;tag=887s

From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

8.1.14 Call-ID

Call-ID ヘッダフィールドは、連続するメッセージをグループ化するための一意の識別子として作用する。それは、ダイアログ中のどの UA が送ったものであれ、すべてのリクエストおよびレスポンスにおいて同じ値でなくてはならない[MUST]。それは、UA からの各登録においても同じ値であるべきである[SHOULD]。

すべてのダイアログの外部にある UAC によって生成された新規リクエストでは、Call-ID ヘッダフィールドは、メソッド特有の動作で優先されなければ、あらゆる時と場所においてグローバルに一意な値となるよう、UAC により選択されなくてはならない[MUST]。すべての SIP UA は、それが作り出す Call-ID ヘッダフィールドが、他のいかなる UA によっても偶然に生成されないことを保証する手段を持たねばならない。リクエストの修正を要請する特定の失敗レスポンスの後にリクエストを再試行するときは(たとえば、認証に対するチャレンジ)、再試行されるリクエストが新規リクエストとみなされることはなく、そのため新規 Call-ID ヘッダフィールドは、必要でないということに注意すること。8.1.3.5 節参照。

Call-ID の生成において、暗号的にランダムな識別子(RFC1750 参考文献[12])の使用が推奨される[RECOMMENDED]。実装では、「localid@host」形式を使用してもよい[MAY]。Call-ID は、大文字小文字を区別し、単純に各バイトごとに比較される。

暗号的にランダムな識別子の使用は、セッションの乗っ取りに対するある種の防御を提供し、起こりうる意図しない Call-ID の衝突を減少させる。

リクエストの Call-ID ヘッダフィールド値を選択するための準備やヒューマン インターフェースは必要とされない。

Call-ID ヘッダフィールドに関しての更なる情報は、20.8 節参照のこと。

例:

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com

8.1.1.5 CSeq

CSeq ヘッダフィールドは、トランザクションを識別し順番付けするための手段としての役目を果たす。CSeq は、シーケンス番号とメソッドから成る。メソッドは、リクエストのメソッドとマッチしなくてはならない[MUST]。ダイアログ外の非 REGISTER リクエストでは、シーケンス番号値は任意である。シーケンス番号値は、32 ビットの符号なし整数で表現可能でなくてはならず[MUST]、また、2 の 31 乗未満でなくてはならない[MUST]。上記のガイドラインに従う限り、クライアントは、CSeq ヘッダフィールド値を選択するために、それが望むいかなるメカニズムでも使用できる。

12.2.1.1 節では、ダイアログ内のリクエストに対する CSeq の構築について論じる。

例:

CSeq: 4711 INVITE

8.1.1.6 Max-Forwards

Max-Forwards ヘッダフィールドは、リクエストが、送信先に向かう過程で通過することができるホップ数を制限するための役目を果たす。Max-Forwards は、各ホップで 1 ずつ減少する整数からなる。リクエストが、送信先に到達する前に Max-Forwards 値が 0 になった場合、リクエストは 483(Too Many Hops)エラーレスポンスで拒否される。

UAC はそれが発信する各リクエストに、値が 70 であるべき[SHOULD]Max-Forwards ヘッダフィールドを挿入しなくてはならない[MUST]。この数は、ループがないときにどのような SIP ネットワークにおいてもリクエストが却下されないだろうことを保証するが、ループが起こったときにプロキシのリソースを消費するほど大きくはない十分に大きな数として選択された。これよりも小さな値は注意して、また UA がトポロジーを知っているネットワークにおいてのみ使用されるべきである。

8.1.1.7 Via

Via ヘッダフィールドは、トランザクションのために使用されるトランスポートを示し、またレスポンスが送られることになるロケーションを識別する。Via ヘッダフィールド値は、ネクストホップに到達するために使用されるトランスポートが選択された後に追加される(これには参考文献[4]の手順の利用が関係するかもしれない)。

UAC がリクエストを生成するとき、それはそのリクエストに Via を挿入しなくてはならない[MUST]。ヘッダフィールド中のプロトコル名とプロトコルバージョンは、それぞれ SIP および 2.0 でなくてはならない[MUST]。Via ヘッダフィールド値は branch パラメータを含まなくてはならない[MUST]。このパラメータは、そのリクエストによって生成されたトランザクションを識別するために使用される。このパラメータはクライアントとサーバの両方に利用される。

branch パラメータ値は、その UA によって送られるすべてのリクエストに対して、あらゆる時と場所を通して一意でなくてはならない[MUST]。このルールの例外は、CANCEL と非 2xx レスポンスに対する ACK である。以下で議論されるように、CANCEL リクエストは、それがキャンセルするリクエストと同じ branch パラメータ値を持つ。17.1.1.3 節で論じるように、非 2xx レスポンスに対する ACK は、INVITE と同じ branch ID を持つ(この INVITE に対するレスポンスを ACK は了承する)。

> トランザクション ID として branch ID パラメータを使用することを容易にするための、branch ID パラメータの一意性の特性は、RFC2543 の一部ではなかった。

この仕様に準拠するエレメントによって挿入された branch ID は、常に文字列「z9hG4bK」から始まらなくてはならない[MUST]。これら 7 つの文字は、リクエストを受け取るサーバで branch ID が、この仕様で述べられる方法で構築されたこと(すなわち、グローバルに一意であること)を確定できるように、マジッククッキーとして使用される(古い RFC2543 の実装がこのような値を選択しないことを確実にするために、7 つで十分だと思われる)。この要求以上の、branch トークンの細かいフォーマットは、実装で定義される。

Via ヘッダの maddr、ttl、および sent-by コンポーネントは、リクエストがトランスポートレイヤで処理されるときに設定される(18 節参照)。

プロキシの Via 処理については、16.6 節の項目 8 および 16.7 節の項目 3 で記述する。

8.1.1.8 Contact

Contact ヘッダフィールドは、これに続くリクエストでその UA の特定のインスタンスにコンタクトするために使用できる SIP URI または SIPS URI を提供する。Contact ヘッダフィールドは、結果としてダイアログを確立することができるすべてのリクエストの中に、正確に一つの SIP または SIPS URI を含んで存在しなくてはならない[MUST]。本仕様で定義されるメソッドでは、INVITE リクエストのみが含まれる。これらのリクエストでは、Contact のスコープはグローバルである。すなわち、Contact ヘッダフィールド値は、UA がリクエストを受け取りたいと望む URI を含み、それ以降のいかなるダイアログ外のリクエストで使用された場合でも、この URI は有効でなくてはならない[MUST]。

Request-URI または最初の Route ヘッダ値が SIPS URI を含む場合、Contact ヘッダフィールドも同様に SIPS URI を含まなくてはならない[MUST]。

Contact ヘッダフィールドに関する更なる情報は、20.10 節参照のこと。

8.1.1.9 Supported と Require (Supported and Require)

UAC が、サーバがレスポンスに適用できる SIP の拡張をサポートする場合、その UAC は、それらの拡張のためのオプションタグ(19.2節)をリストした Supported ヘッダフィールドをリクエストに含めるべきである[SHOULD]。

リストされたオプションタグは、標準化過程にある RFC(standards track RFC)で定義された拡張のみを参照しなくてはならない[MUST]。これは、サーバが、サービスを受けるためにベンダーが定義した非標準の機能を実装するクライアントからの強要を防止するためである。experimental RFC や informational RFC で定義された拡張は、それらもベンダーが定義した拡張をドキュメント化するために使用されることが多いので、リクエストの Supported ヘッダフィールドで使用することから明確に除外される。

UAC が、リクエストを処理するためにそのリクエストに適用する拡張を、UAS が理解することを強要したいと望む場合、UAC はその拡張のためのオプションタグをリストした Require ヘッダフィールドをリクエストに挿入しなくてはならない[MUST]。UAC が、リクエストに拡張を適用することを望み、通過するすべてのプロキシがその拡張を理解することを強要したい場合、UAC はその拡張のためのオプションタグをリストした Proxy-Require ヘッダフィールドをリクエストに挿入しなくてはならない[MUST]。

Supported ヘッダフィールドと同様に、Require および Proxy-Require ヘッダーフィールドのオプションタグは、standards-track RFC で定義されている拡張のみを参照しなくてはならない[MUST]。

8.1.1.10 付加的なメッセージコンポーネント (Additional Message Components)

新しいリクエストが生成され、上に述べられたヘッダフィールドが適切に構成された後に、メソッド固有のすべてのヘッダフィールドを追加すると同時に、すべての付加的なオプションのヘッダフィールドが追加

される。

SIP リクエストは、MIME エンコードされた message-body を含んでもよい[MAY]。リクエストが含んでいるボディの種類に関わらず、ボディの内容を特徴付けるために、特定のヘッダフィールドが明記されなければならない。これらのヘッダフィールドに関する更なる情報は、20.11 節から 20.15 を参照のこと。

8.1.2 **リクエストの送信** (Sending the Request)

その後、リクエストの送信先が算出される。特にローカルポリシーが規定されていなければ、以下に示すように参考文献[4]で述べられている手順で DNS を適用することにより送信先が、確定されなくてはならない[MUST]。ルートセットの最初のエレメントがストリクトルータを示す場合(12.2.1.1 節で述べられるようにリクエストを形成することになる)、リクエストの Request-URI にその手順が適用されなくてはならない[MUST]。そうでなければ、手順は、リクエストの最初の Route ヘッダフィールド値(存在すれば)に適用されるか、または Route ヘッダフィールドが存在しない場合は、リクエストの Request-URI に適用される。これらの手順は、順番に並べられた、試行するアドレス、ポート、およびトランスポートのセットを与える。参考文献[4]の手順に対する入力としてどの URI が使用されるかに依らず、Request-URI が SIPS リソースを指定する場合には、UAC は入力された URI が SIPS URI であったかのように参考文献[4]の手順に従わなくてはならない[MUST]。

ローカルポリシーは、試行するための別の送信先のセットを規定してもよい[MAY]。Request-URI が SIPS URI を含む場合、別のいかなる送信先も TLS でコンタクトされなくてはならない[MUST]。リクエストが Route ヘッダフィールドを含まなければ、別のデスティネーションにおけるそれ以上の制約はない。これは、アウトバウンドプロキシを指定する方法として、既存のルートセットに代わる単純な選択肢を提供する。しかしながら、アウトバウンドプロキシを設定するためのそのアプローチは推奨されない[NOT RECOMMENDED]。一つの URI を持つ既存のルートセットが、その代わりに使用されるべきである[SHOULD]。リクエストが Route ヘッダフィールドを含む場合、その先頭の値から得られるロケーションにリクエストが、送られるべきである[SHOULD]が、(RFC2543 中のそれらとは対象的に) UA が、このドキュメントで規定される Route と Request-URI のポリシーを守ることが確かだと思うどのようなサーバにも送ってもよい[MAY]。特に、アウトバウンドプロキシを設定された UAC は、アウトバウンドプロキシにすべてのメッセージを送るというポリシーを採用する代わりに、最初の Route ヘッダフィールド値で示されるロケーションにリクエストを送ることを試みるべきである[SHOULD]。

> これは、Record-Route ヘッダフィールド値を追加しないアウトバウンドプロキシが、それ以降のリクエストのパスから除外されることを保証する。それは、最初の経路(ルート)の URI を解決できないエンドポイントが、そのタスクをアウトバウンドプロキシに委任することを許容する。

UAC は、サーバにコンタクトするまで各アドレスを試行するという、ステートフルエレメントのために参考文献[4]で定義されている手順に従うべきである[SHOULD]。それぞれの試行は、新規トランザクションを構成し、それにより新規 branch パラメータを持った異なる先頭の Via ヘッダフィールド値を伝える。さらに、Via ヘッダフィールドの transport 値には、ターゲットサーバに対して決定されたどのようなトランスポートでも設定される。

8.1.3 **レスポンスの処理** (Processing Responses)

レスポンスは、最初にトランスポートレイヤで処理され、その後トランザクションレイヤに上げられる(渡

される)。トランザクションレイヤは、その処理を行い、その後それを TU に上げる(渡す)。TU でのレスポンス処理の大半はメソッド固有である。しかしながら、メソッドに依存しないいくつかの一般的な動作がある。

8.1.3.1 トランザクションレイヤのエラー (Transaction Layer Errors)

場合によっては、トランザクションレイヤから返送されるレスポンスは SIP メッセージではなく、トランザクションレイヤのエラーであろう。トランザクションレイヤからタイムアウトエラーを受け取った場合、それは 408(Request Timeout)ステータスコードを受け取ったかのように扱われなくてはならない[MUST]。トランスポートレイヤから致命的なトランスポートエラーが報告された場合(一般的に、UDP における致命的な ICMP エラーまたは TCP におけるコネクション失敗のため)、その状態は 503(Service Unavailable)ステータスコードとして扱われなくてはならない[MUST]。

8.1.3.2 認識できないレスポンス (Unrecognized Responses)

UAC は、それが認識できないどのような最終レスポンス(Final response)でも、そのクラスの x00 レスポンスコードに等価なものとして扱わなくてはならず[MUST]、すべてのクラスの x00 レスポンスコードを処理できなくてはならない[MUST]。たとえば、UAC が認識できないレスポンスコード 431 を受け取った場合、UAC はそれのリクエストに何か問題があったと間違いなく仮定し、そのレスポンスを 400(Bad Request)レスポンスコードを受け取ったかのように処理できる。UAC は、それが認識できない 100 以外のすべての暫定レスポンス(provisional response)を 183(Session Progress)レスポンスコードとして扱わなくてはならない[MUST]。UAC は 100 と 183 レスポンスを処理できなくてはならない[MUST]。

8.1.3.3 複数の Via (Vias)

レスポンスの中に 2 つ以上の Via ヘッダフィールド値が存在する場合、UAC はそのメッセージを破棄するべきである[SHOULD]。

> リクエストの発信元の前にある付加的な Via ヘッダフィールド値の存在は、メッセージがミスルーティングされたか、あるいは壊れていることを示唆している。

8.1.3.4 3xx レスポンスの処理 (Processing 3xx Responses)

リダイレクトレスポンスの受信時には(たとえば、301 レスポンスステータスコード)、クライアントは、リダイレクトされたリクエストに基づいて一つ以上の新しいリクエストを作るために、Contact ヘッダフィールドの URI を使用するべきである[SHOULD]。この処理は、16.5 節と 16.6 で述べられているように、3xx クラスレスポンスで再帰するプロキシの処理と似ている。クライアントは、正確に一つの URI(オリジナルリクエストの Request-URI)を含む最初のターゲットセットで処理を開始する。クライアントが、そのリクエストに対する 3xx クラスレスポンスに基づく新規リクエストを作ることを望む場合、クライアントはターゲットセットに、試行する URI を配置する。この仕様の制限にしたがい、クライアントは、ターゲットセットにどの Contact の URI を配置するか選択できる。プロキシの再帰と同様に、3xx クラスレスポンスを処理するクライアントは、与えられたいかなる URI もターゲットセットに 1 回より多く追加してはいけない[MUST NOT]。オリジナルリクエストが Request-URI に SIPS URI を持っていた場合、クライアントは非 SIPS URI に再帰することを選択してもよい[MAY]が、不確かな URI にリダイレクトすることをユーザに通知するべきである[SHOULD]。

> 新規のどのようなリクエストでも、contact としてオリジナル URI を含む 3xx レスポンスを受け取るかもしれない。2 つのロケーションをお互いにリダイレクトするように設定することができる。与えられ

たいかなるURIでもターゲットセットに1回だけ置くことで、リダイレクションの無限ループを防げる。

ターゲットセットが大きくなると、クライアントはそのURIに対してどのような順番でも新規リクエストを生成してもよい[MAY]。一般的なメカニズムは、Contactヘッダフィールド値からのqパラメータ値でターゲットセットを並べ替えることである。それらのURIに対するリクエストは順次に(serially)あるいは並行して(parallel)生成してもよい[MAY]。大きいものから小さいものへqの値のグループを順次に、そしてそれぞれのq値のグループのURIを並行して処理することが一つのアプローチである。等しいq値のcontactの間では任意に選択を行い、qの値の大きいものから小さいものの順番で順次処理だけを実行することが、もう一つの別のアプローチである。

リストのアドレスへのコンタクトが失敗することになった場合、次のパラグラフで定義されるように、エレメントは、リストを使い切るまで、リストの次のアドレスに移る。リストを使いきった場合、リクエストは失敗したことになる。

失敗は、失敗レスポンスコード(399よりも大きいコード)を介して検出されるべきである[SHOULD]。ネットワークエラーに対してクライアントトランザクションは、どのようなトランスポートレイヤのエラーでもトランザクションユーザに報告する。いくつかのレスポンスコード(8.1.3.5節で詳述する)は、リクエストが再試行できることを示すことに注意すること。再試行されるリクエストは失敗とみなされるべきではない。

個々のcontactアドレスに対する失敗を受け取る場合、クライアントは次のcontactアドレスを試すべきである[SHOULD]。これは、新規リクエストを配送するために新規クライアントトランザクションの生成を伴う。

3xxレスポンス中のcontactアドレスに基づいてリクエストを生成するために、UACは、ターゲットセットからRequest-URIに、URIのmethod-paramとheaderパラメータ(これらのパラメータの定義は19.1.1節参照のこと)を除き、すべてのURIをコピーしなくてはならない[MUST]。それは、19.1.5節のガイドラインに従って、新規リクエストのヘッダフィールド値を生成するためにheaderパラメータを、リダイレクトされたリクエストに関連付けられたヘッダフィールド値を上書きして使用する。

ある場合には、contactアドレスで伝えられたヘッダフィールドが、リダイレクトされたオリジナルリクエストに存在するリクエストヘッダフィールドの代わりに付加されるかも知れないということに注意すること。一般的なルールとして、ヘッダフィールドが、カンマ区切りの値のリストを受け入れることができる場合、新しいヘッダフィールド値は、リダイレクトされたオリジナルリクエストに存在するいかなる値にも付加してもよい[MAY]。ヘッダフィールドが複数の値を受け入れない場合、リダイレクトされたオリジナルリクエストの値は、contactアドレスで伝えられたヘッダフィールド値で上書きしてもよい[MAY]。たとえば、contactアドレスが以下の値で返送される場合、

sip:user@host?Subject=foo&Call-Info=<http://www.foo.com>

リダイレクトされたオリジナルリクエストのどのようなSubjectヘッダフィールドも上書きされるが、HTTP URLは既存のどのようなCall-Infoヘッダフィールド値にも単に付加されるだけである。

UACはリダイレクトされたオリジナルリクエストで使われていたのと同じTo、From、およびCall-IDを再

利用することが推奨される[RECOMMENDED]が、UAC は、たとえば新規リクエストのために Call-ID ヘッダフィールド値をアップデートすることも選択してもよい[MAY]。

最後に、新規リクエストが構築されるとすぐに、それは新しいクライアントトランザクション上で送られる。したがって、8.1.1.7 節で議論されるように、最初の Via フィールドに新しい branch ID を持たなくてはならない[MUST]。

その他のすべての点において、リダイレクトレスポンスの受信によって送られるリクエストは、オリジナルリクエストのヘッダフィールドとボディを再利用するべきである[SHOULD]。

ある場合には、Contact ヘッダフィールド値は、受け取ったステータスコードおよび有効期限間隔の存在に依存して、UAC において一時的あるいは恒久的にキャッシュされるかもしれない。21.3.2 と 21.3.3 節参照のこと。

8.1.3.5 4xx レスポンスの処理 (Processing 4xx Responses)

ある種の 4xx レスポンスコードは、メソッドに依存しない特定の UA 処理を要求する。

401(Unauthorized)または 407(Proxy Authentication Required)レスポンスを受け取った場合、UAC は信用証明書(credentials)を持ったリクエストで再試行するために、22.2 および 22.3 節の認可手順に従うべきである[SHOULD]。

413(Request Entity Too Large)レスポンスを受け取った場合(21.4.11 節)、リクエストは、UAS が受け入れようとしたものよりも長いボディを含んでいる。可能であれば、UAC は、ボディを省略するかより短いボディを使用して、リクエストを再試行するべきである[SHOULD]。

415(Unsupported Media Type)レスポンスを受け取った場合(21.4.13 節)、リクエストは、UAS のサポートしないメディアタイプを含んでいた。UAC は、今度はレスポンス中の Accept ヘッダフィールドにリストされたタイプ、レスポンス中の Accept-Encoding ヘッダフィールドにリストされたエンコーディング、およびレスポンス中の Accept-Language にリストされた言語を含む内容だけを使用して、リクエストの送信を再試行するべきである[SHOULD]。

416(Unsupported URI Scheme)レスポンスを受け取った場合(21.4.14 節)、Request-URI がサーバのサポートしない URI スキームを使用している。クライアントは、今度は SIP URI を使用して、リクエストを再試行するべきである[SHOULD]。

420(Bad Extension)レスポンスを受け取った場合(21.4.15 節)、リクエストはプロキシまたは UAS がサポートしない機能のオプションタグをリストしている Proxy-Require または Require（※）ヘッダフィールドを含んでいた。UAC は、今度はレスポンス中の Unsupported ヘッダフィールドにリストされたすべての拡張を省略して、リクエストを再試行するべきである[SHOULD]。

> *[※訳注：原文では"Require or Proxy-Require"となっているが、順序逆転の 誤記と考えられる。本件については、SIP WG においてもバグと認識されている]*

上記のすべての場合において、適切な修正を加えた新規リクエストを生成することによって、リクエストは再試行される。この新規リクエストは、新規のトランザクションを構成する。また、この新規リクエストは、それ以前に送られたリクエストと同じ Call-ID、To、および From の値を持つべきである[SHOULD]が、CSeq は前回よりも一つ大きなシーケンス番号を含むべきである。

まだ定義されていないものも含めその他の 4xx レスポンスでは、メソッドと使用場面によって、リトライが可能かもしれないし可能でないかもしれない。

8.2 UAS の動作 (UAS Behavior)

ダイアログの外部のリクエストが UAS によって処理されるとき、メソッドとは無関係に従うべき処理ルールのセットがある。12 節で、リクエストが、ダイアログの内部にあるか外部にあるかを UAS がどのように判断するかについての手引きを与える。

リクエスト処理は、非分割(atomic)であることに注意すること。リクエストが、受け入れられるとき、それに関連付けられるすべての状態の変更が行われなくてはならない[MUST]。それが拒否される場合は、すべての状態変更は実行されてはいけない[MUST NOT]。

UAS はこの節のこれ以降に書かれているステップの順番でリクエストを処理するべきである[SHOULD](すなわち、認証から始めて、次にメソッド、ヘッダフィールドなどこの節の残りの部分全体を通して書かれていることを検査する)。

8.2.1 メソッド検査 (Method Inspection)

いったんリクエストが認証されると(または認証がスキップされると)、UAS はリクエストのメソッドを検査しなくてはならない[MUST]。UAS がリクエストのメソッドを認識したがサポートしていない場合は、405(Method Not Allowed)レスポンスを生成しなくてはならない[MUST]。レスポンスの生成のための手順は 8.2.6 節で述べられている。UAS は 405(Method Not Allowed)レスポンスに Allow ヘッダフィールドも加えなくてはならない[MUST]。Allow ヘッダフィールドは、メッセージを生成する UAS がサポートしているメソッドのセットをリストしなくてはならない[MUST]。Allow ヘッダフィールドについては 20.5　節で述べられている。

そのメソッドが、サーバによってサポートされているものであれば、処理は継続する。

8.2.2 ヘッダ検査 (Header Inspection)

UAS がリクエスト中のヘッダフィールドを理解しない(すなわち、そのヘッダフィールドがこの仕様またはサポートされるいかなる拡張でも定義されていない)場合、サーバは、そのヘッダフィールドを無視し、メッセージの処理を継続しなくてはならない[MUST]。UAS は、リクエストの処理に必要ではないすべての不正な形式のヘッダを無視するべきである[SHOULD]。

8.2.2.1 To と Request-URI (To and Request-URI)

To ヘッダフィールドは、From フィールドで識別されるユーザが指定した、リクエストのオリジナル受信者を識別する。着信転送または他のプロキシによるオペレーションがあるため、オリジナル受信者がリクエストを処理する UAS かどうかわからない。UAS は、To フィールドが、その UAS と同一でないときにリクエストを受け取るかどうか決定するために、それが望むどのようなポリシーでも適用してもよい[MAY]。しかし

ながら、UAS は、To ヘッダフィールド中の URI スキーム(たとえば tel: URI)を認識できない場合、あるいは To ヘッダフィールドが、その UAS の既知または現在のユーザに宛てられたものでない場合でも、リクエストを受け取ることが推奨される[RECOMMENDED]。その一方で、UAS がリクエストを拒否することにした場合、403 ステータスコードレスポンスを生成し、それをトランスミッションのためにサーバトランザクションに渡すべきである[SHOULD]。

一方、Request-URI は、リクエストを処理する UAS を識別する。Request-URI が、UAS のサポートしないスキームを使用する場合、UAS は、そのリクエストを 416(Unsupported URI Scheme)レスポンスで拒否するべきである[SHOULD]。UAS は、あるアドレスに対してのリクエストを受け入れることを望むが、Request-URI がそのアドレスと同じでない場合、UAS は 404(Not Found)レスポンスでリクエストを拒否するべきである[SHOULD]。一般的に、特定のコンタクトアドレスに自身の address-of-record (AOR)をバインドするために REGISTER メソッドを使用する UA は、そのコンタクトアドレスに等しい Request-URI を持つリクエストを受ける。受け取る Request-URI の、可能性のあるその他のソースは、ダイアログを確立またはリフレッシュする、その UA が送るリクエストとレスポンスの Contact ヘッダフィールドが含まれる。

8.2.2.2 マージされたリクエスト (Merged Requests)

リクエストが、To ヘッダフィールドにタグを持たない場合、UAS コアは、進行中のトランザクションに対してそのリクエストを確認しなくてはならない[MUST]。From タグ、Call-ID、CSeq が進行中のトランザクションに関連付けられているものと正確に一致するが、リクエストはトランザクションにマッチしない(17.2.3 節のマッチングルールに基づいて)場合、UAS コアは、482(Loop Detected)レスポンスを生成し、それをサーバトランザクションに渡すべきである[SHOULD]。

> 同じリクエストが異なるパスを通って(最も可能性が高いのはフォークによって)UAS に 1 回より多く到着するとき、UAS は受け取ったそのようなリクエストの最初のものを処理し、残りのものに対しては 482(Loop Detected)で レスポンスする。

8.2.2.3 Require

UAS は、リクエストを処理する適切なエレメントであると判断した場合、それは Require ヘッダフィールド(もし存在すれば)を検査する。

Require ヘッダフィールドは、リクエストを適切に処理するために UAS がサポートすることを UAC が期待する SIP 拡張について、UAS に知らせるために UAC が使用する。Require ヘッダフィールドのフォーマットは 20.32 節で述べられている。Require ヘッダフィールドにリストされている option-tag を UAS が理解しない場合、それはステータスコード 420(Bad Extension)レスポンスを生成して応答しなくてはならない[MUST]。UAS は Unsupported ヘッダフィールドを追加し、リクエストの Require ヘッダフィールドの中で理解できなかったオプションをその中にリストしなくてはならない[MUST]。

Require と Proxy-Require は、SIP の CANCEL リクエスト、または非 2xx レスポンスに対する ACK リクエストで使用してはいけない[MUST NOT]ことに注意すること。これらのヘッダフィールドは、これらのリクエスト中に存在した場合には無視されなくてはならない[MUST]。

2xx レスポンスに対する ACK リクエストは、最初のリクエストに存在した Require と Proxy-Require の値のみを含まなくてはならない[MUST]。

例:

```
UAC->UAS:   INVITE sip:watson@bell-telephone.com SIP/2.0
              Require: 100rel

UAS->UAC:   SIP/2.0 420 Bad Extension
              Unsupported: 100rel
```

この動作は、クライアントとサーバのやりとり(interaction)が、双方にすべてのオプションが理解されたときは遅滞なく進行し、(上記の例のように)オプションが理解されなかったときにのみスローダウンするということを確実にする。うまく適合したクライアントとサーバのペアでは、ネゴシエーションメカニズムによく必要とされるラウンドトリップを節約し、やりとりはすばやく進行する。さらにそれは、サーバが理解できない機能をクライアントが要求したときのあいまいさも除去する。呼ハンドリングフィールド(call handling field)のようないくつかの機能は、エンドシステムにのみ関係がある。

8.2.3 **コンテンツの処理** (Content Processing)

UAS が、クライアントに要求されたすべての拡張を理解できると仮定すると、UAS はメッセージのボディとそれを記述するヘッダフィールドを検査する。そのタイプ(Content-Type で示される)、言語(Content-Language で示される)、あるいはエンコーディング(Content-Encoding で示される)を理解できないボディがあり、そのボディ部分が optional(Content-Disposition で示される)でなければ、UAS は、415(Unsupported Media Type)レスポンスでそのリクエストを拒否しなくてはならない[MUST]。リクエストが、UAS がサポートしないタイプのボディを含んだ場合、レスポンスは、それが理解できるすべてのタイプのボディをリストした Accept ヘッダフィールドを含まなくてはならない[MUST]。リクエストが、UAS で理解できないコンテンツのエンコーディングを含んだ場合、レスポンスは、UAS で理解できるエンコーディングをリストした Accept-Encoding ヘッダフィールドを含まなくてはならない[MUST]。リクエストが、UAS が理解できない言語のコンテンツを含んだ場合、レスポンスは、UAS で理解できる言語を示す Accept-Language ヘッダフィールドを含まなくてはならない[MUST]。これらのチェック以上のボディのハンドリングはメソッドとタイプ固有である。コンテンツ固有のヘッダフィールド処理に関する更なる情報は、7.4 節および 20.11 節から 20.15 節を参照のこと。

8.2.4 **拡張の適用** (Applying Extensions)

レスポンスを生成する際にある種の拡張の適用を望む UAS は、その拡張のサポートが、リクエスト中の Supported ヘッダフィールドで示されない場合は、そうしてはいけない[MUST NOT]。希望する拡張がサポートされていない場合、サーバは、ベースライン SIP およびクライアントがサポートするその他の拡張だけに頼るべきである[SHOULD]。拡張なしではサーバがリクエストを処理できないという稀な状況において、サーバは 421(Extension Required)レスポンスを送ってもよい[MAY]。このレスポンスは、特定の拡張のサポートがないと適切なレスポンスを生成できないことを示す。必要とされる拡張は、レスポンスの Require ヘッダフィールドに含まれなくてはならない[MUST]。この動作は、通常、相互運用性を取れなくするので推奨されていない[NOT RECOMMENDED]。

非 421 レスポンスに適用されるすべての拡張は、そのレスポンスに含まれる Require ヘッダフィールドにリストされなくてはならない[MUST]。当然、サーバはリクエストの Supported ヘッダフィールドにリストされていない拡張を適用してはいけない[MUST NOT]。この結果として、レスポンス中の Require ヘッダフィー

ルドは、standards track RFC で定義されているオプションタグのみを含むことになる。

8.2.5 **リクエストの処理** (Processing the Request)

これ以前のサブ節のチェックをすべてパスしたと仮定すると、UAS の処理はメソッド固有のものになる。10 節では REGISTER リクエスト、11 節では OPTIONS リクエスト、13 節では INVITE リクエスト、そして 15 節では BYE リクエストを扱う。

8.2.6 **レスポンスの生成** (Generating the Response)

UAS が、リクエストに対するレスポンスを構成したいときは、このあとのサブ節で詳細されている手順に従う。本節で詳述されていない、該当するレスポンスコードに固有の付加的な動作も必要とされるかもしれない。

レスポンスの生成に関連するすべての手順が完了するとすぐに、UAS はそのレスポンスをリクエストを受け取ったサーバトランザクションに返す。

8.2.6.1 **暫定レスポンスの送信** (Sending a Provisional Response)

レスポンスの生成のための、メソッドに非固有の主たるガイドラインは、UAS は非 INVITE リクエストに対して暫定レスポンス(provisional response)を発行するべきではない[SHOULD NOT]ということである。UAS はむしろ、非 INVITE リクエストに対して可能な限り早く最終レスポンス(Final response)を返すべきである[SHOULD]。

100(Trying)レスポンスが生成されるとき、リクエストに存在するいかなる Timestamp ヘッダフィールドもこの 100(Trying)レスポンスにコピーされなくてはならない[MUST]。レスポンスの生成に遅延が発生する場合、UAS はレスポンスの Timestamp 値に delay 値を追加するべきである[SHOULD]。この値はリクエストの受信とレスポンスの送信の間の時間差を、秒単位で含まなくてはならない[MUST]。

8.2.6.2 **ヘッダとタグ** (Headers and Tags)

レスポンスの From フィールドは、リクエストの From ヘッダフィールドと等しくなくてはならない[MUST]。レスポンスの Call-ID ヘッダフィールドは、リクエストの Call-ID ヘッダフィールドと等しくなくてはならない[MUST]。レスポンスの CSeq ヘッダフィールドは、リクエストの CSeq フィールドと等しくなくてはならない[MUST]。レスポンスの Via ヘッダフィールド値はリクエストの Via ヘッダフィールド値に等しくなければならず[MUST]、同じ並び順を維持しなくてはならない[MUST]。

リクエストが、To タグを含んだ場合、レスポンスの To ヘッダフィールドは、リクエストのものと等しくなくてはならない[MUST]。しかしながら、リクエストの To ヘッダフィールドがタグを含まなかった場合は、レスポンスの To ヘッダフィールドの URI は、その To ヘッダフィールドの URI に等しくなくてはならない[MUST]。さらに UAS は、レスポンスの To ヘッダフィールドにタグを追加しなくてはならない[MUST](100(Trying)レスポンスは例外とする。それにはタグを存在させてもよい[MAY])。このことは、おそらくはダイアログ ID のコンポーネントをもたらすことによって、応答する UAS を識別するために役立つ。同じタグがそのリクエストに対するすべてのレスポンス(最終と暫定の両方)に使用されなくてはならない[MUST](前と同じく 100(Trying)を例外とする)。タグの生成手順は 19.3 節で定義されている。

8.2.7 ステートレス UAS の動作 (Stateless UAS Behavior)

ステートレス UAS とは、トランザクションステートを保持しない UAS のことである。それは通常どおりにリクエストに返答するが、レスポンスが送られた後に UAS が通常は保持するいかなる状態も破棄する。ステートレス UAS が、リクエストの再送を受け取る場合、それは、単にリクエストの最初のインスタンスに返答するかのように、レスポンスを再度生成して再送する。リクエストが同一の場合、メソッドに対するリクエスト処理が常に同じレスポンスという結果にならない限り、UAS はステートレスではありえない。これは、例えば、ステートレスレジストラサーバを除外する。ステートレス UAS は、トランザクションレイヤを使用しない。ステートレス UAS は、トランスポートレイヤから直接リクエストを受け取り、トランスポートレイヤに直接レスポンスを送る。

ステートレス UAS の役割は、チャレンジレスポンスが発行される認証されていないリクエストをハンドリングするために、主に必要とされる。もしも、認証されていないリクエストがステートフルにハンドリングされると、認証されていない悪意のある大量のリクエストが、効果的に UAS の呼処理をスローダウンまたは完全に止めてしまうかもしれない大量のトランザクションステートを生成することがありうる。詳細は 26.1.5 節参照のこと。

ステートレス UAS の最も重要な動作は以下である。

- ステートレス UAS は、暫定レスポンス(provisional response)(1xx)を送ってはいけない[MUST NOT]。
- ステートレス UAS は、レスポンスを再送してはいけない[MUST NOT]。
- ステートレス UAS は、ACK リクエストを無視しなくてはならない[MUST]。
- ステートレス UAS は、CANCEL リクエストを無視しなくてはならない[MUST]
- レスポンスに対して To ヘッダタグが、ステートレスな方法で生成されなくてはならない[MUST](同じリクエストに対しては同じタグを一貫して生成するやり方で)。タグの生成についての情報は 19.3 節参照のこと。

他のすべての点において、ステートレス UAS は、ステートフル UAS と同じように動作する。UAS はそれぞれの新規リクエストに対して、ステートフルまたはステートレスのいずれかのモードで動作できる。

8.3 リダイレクトサーバ (Redirect Servers)

あるアーキテクチャーではリダイレクションによって、リクエストをルートする役目があるプロキシサーバの処理負荷を減らし、シグナリングパスの堅牢性を向上させることが望ましいかもしれない。

リダイレクションは、サーバがリクエストのためのルーティング情報をレスポンスに含めてクライアントに差し戻すことを認める。それによって、リクエストのターゲットの場所を特定することを手助けするにもかかわらず、それ自身をこのトランザクションの今後のメッセージングのループから除外する。リクエストの発信元が、リダイレクションを受け取ったときは、受け取った URI に基づいて新規リクエストを送る。URI をネットワークのコアからエッジに伝播することにより、リダイレクションは、かなりのネットワークスケ

ーラビリティが考慮されている。

リダイレクトサーバは、論理的に、サーバトランザクションレイヤと、ある種のロケーションサービスにアクセス権を持つトランザクションユーザから構成される。(レジストラサーバとロケーションサービスの詳細については 10 節参照)。このロケーションサービスは事実上、一つの URI とその URI のターゲットを見つけることができる一つ以上の選択可能なロケーションのセットとの間のマッピング情報を含むデータベースである。

リダイレクトサーバは、それ自身でいかなる SIP リクエストも発行しない。CANCEL 以外のリクエストを受け取った後で、サーバは、そのリクエストを拒否するか、ロケーションサービスから選択可能なロケーションのリストを収集し、クラス 3xx の最終レスポンス(Final response)を返送する。well-formed な CANCEL リクエストでは、2xx レスポンスを返送するべきである[SHOULD]。このレスポンスは、SIP トランザクションを終了する。リダイレクトサーバは SIP トランザクション全体を通してトランザクションステートを保持する。リダイレクトサーバ間のループ転送を検知するのはクライアントの義務である。

リダイレクトサーバが、リクエストに対して 3xx レスポンスを返送するとき、それは Contact ヘッダフィールドに(一つ以上の)選択可能なロケーションのリストを入れ込む。Contact ヘッダフィールド値の expires パラメータも、Contact データの生存期間を示すために供給することができる。

Contactヘッダフィールドは、試行するための新しいロケーションまたはユーザ名を与える URI を含むか、あるいは単に付加的なトランスポートパラメータを指定することができる。301(Moved Permanently)レスポンス、または 302(Moved Temporarily)レスポンスも最初のリクエストでターゲットとされたのと同じロケーションとユーザ名を与えることができるが、試行するための別のサーバまたはマルチキャストアドレスのような付加的なトランスポートパラメータ、あるいは SIP トランスポートの UDP から TCP(あるいはその逆)への変更を指定する。

しかしながら、リダイレクトサーバは Request-URI の URI と同じ URI にリクエストをリダイレクトしてはいけない[MUST NOT]。そのかわりに、その URI が、自分自身を指すものでない場合には、サーバはリクエストを送信先 URI にプロキシしてもよい[MAY]。または、404 で拒否してもよい[MAY]。

> クライアントがアウトバウンドプロキシを使用しており、そのプロキシが実際にはリクエストをリダイレクトする場合、無制限のリダイレクトループの可能性が出てくる。

Contact ヘッダフィールド値は、オリジナルにコールしたものとは別のリソースも参照してもよい[MAY]ということに注意すること。たとえば、PSTN ゲートウェイに接続された SIP コールは、「おかけになった電話番号は、変更されました」というような特別な通知を配送する必要があるかもしれない。

Contact レスポンスヘッダフィールドは、SIP URI に制限されずに、着信側パーティに到達できる場所を示す適切などのような URI でも含むことができる。たとえば、電話番号、FAX、irc を表す URI(それらが定義されていれば)、または mailto:(RFC2368 参考文献[32])の URL を含むこともありうる。26.4.4 節では、SIPS URI を非 SIPS URI にリダイレクトすることの意味と制限について論じる。

Contact ヘッダフィールド値の expires パラメータは、URI がどのくらいの間有効かを示す。expires は、秒

を表す数字である。このパラメータが提供されていない場合、URI がどれだけの間有効かは、Expires ヘッダフィールドの値が決定する。不正な形式のものは、3600 に等価なものとして扱われるべきである[SHOULD]。

これは適度なレベルで、このヘッダフィールドに絶対時間を認めていた RFC2543 との下位互換性を提供する。絶対時間を受け取った場合、それは不正な形式として扱われ、3600 に初期化される。

リダイレクトサーバは、理解できない機能(理解不能なヘッダフィールド、Require の未知のすべてのオプションタグ、またはメソッド名さえも含む)を無視したうえで、当該リクエストのリダイレクションを進めなくてはならない[MUST]。

９．リクエストのキャンセル (Canceling a Request)

前節では、リクエストの生成、およびあらゆるメソッドのリクエストに対するレスポンスを処理するための一般的な UA の動作について議論した。この節では、CANCEL と呼ばれる汎用メソッドについて論じる。

その名称が示すように、CANCEL リクエストは、クライアントから送られたそれ以前のリクエストをキャンセルするために使用される。具体的には、それは UAS にリクエストの処理を中止してそのリクエストに対してエラーレスポンスを生成するように依頼する。CANCEL は、UAS がすでに最終レスポンス(Final response)を与えたリクエストには何ら影響しない。このためそれは、応答するのにサーバが長時間を要するリクエストをキャンセルするのにもっとも有用である。この理由により CANCEL は、レスポンスを生成するのに長時間を要する INVITE リクエストに対してもっとも有用である。この用法では、INVITE に対する CANCEL リクエストを受け取る UAS(まだ INVITE に対する最終レスポンス(Final response)を送っていない)は、呼び出し音を鳴らすのを止めてから、特定のエラーレスポンス(487)で INVITE に応答することになる。

CANCEL リクエストは、プロキシとユーザエージェントクライアント双方で構築し送ることができる。15 節では、どのような状況で UAC が、INVITE を CANCEL するか、16.10 節では、プロキシでの CANCEL の使用法を論じる。

ステートフルプロキシは、ダウンストリームエレメントから受け取るレスポンスを単にフォワードするのではなく、CANCEL に対してレスポンスする。そのため、CANCEL は各ステートフルプロキシでレスポンスされるので、「ホップバイホップ」リクエストと呼ばれる。

9.1 クライアントの動作 (Client Behavior)

CANCEL リクエストは、INVITE 以外のリクエストをキャンセルするために送るべきではない[SHOULD NOT]。

INVITE 以外のリクエストは、即座にレスポンスされるので、非 INVITE リクエストに対して CANCEL を送ることは常に競合状態(race condition)を作り出すことになるだろう。

CANCEL リクエストを構築するために以下の手順が用いられる。CANCEL リクエスト中の Request-URI、Call-ID、To、Cseq の数字部分、From ヘッダフィールドは、タグも含めて、キャンセルされるリクエストに含まれるものと同じでなくてはならない[MUST]。クライアントが構築した CANCEL は、キャンセルされるリクエストの最初の Via 値にマッチする、ただ一つの Via ヘッダフィールド値を持たなくてはならない[MUST]。これらのヘッダフィールドに同じ値を使用することは、CANCEL を、それがキャンセルするリクエストにマッチングすることを可能にする(9.2 節で、そのようなマッチングがどのように行われるか示す)。しかしなが

ら、CSeq ヘッダフィールドのメソッド部分は、CANCEL の値を持たなくてはならない[MUST]。これは、それがそれ自身の権限を持ったトランザクションとして識別され処理されることを認める(17 節参照)。

キャンセルされるリクエストが、Route ヘッダフィールドを含む場合、CANCEL リクエストはその Route ヘッダフィールドの値を含まなくてはならない[MUST]。

これはステートレスプロキシが、CANCEL リクエストを適切にルーティングするために必要とされる。

CANCEL リクエストは、いかなる Require あるいは Proxy-Require ヘッダフィールドも含んではいけない[MUST NOT]。

CANCEL が構築されるとすぐに、クライアントは、キャンセルされるリクエスト(ここでは、オリジナルリクエストと呼ぶ)に対する何らかのレスポンス(暫定または最終)を受け取っているかどうか調べるべきである[SHOULD]。

暫定レスポンス(provisional response)を一つも受け取っていなければ、CANCEL リクエストを送ってはいけない[MUST NOT]。むしろ、クライアントは、そのリクエストを送る前に暫定レスポンス(provisional response)の到着を待たなくてはならない[MUST]。オリジナルリクエストがすでに最終レスポンス(Final response)を生成している場合、CANCEL はすでに最終レスポンス(Final response)を生成したリクエストに影響を与えないため、それは有効なノーオペレーション命令(no-op)であるので、CANCEL を送るべきではない[SHOULD NOT]。クライアントが CANCEL を送ることを決定するとき、それは CANCEL のためのクライアントトランザクションを生成し、それに送信先アドレス、ポート、トランスポートと共に CANCEL リクエストを渡す。CANCEL のための送信先アドレス、ポート、トランスポートは、オリジナルリクエストを送るために使用されたものと同じでなくてはならない[MUST]。

もし、以前に送ったリクエストに対するレスポンスを受け取る前に CANCEL を送ることが許可されていたとしたら、サーバが、オリジナルリクエストを受け取る前に、CANCEL を受信することが起こりうる。

オリジナルリクエストおよび CANCEL トランザクションに関連するトランザクションは、共に独自に完了するということに注意すること。しかしながら、リクエストをキャンセルする UAC がオリジナルリクエストに対する 487(RequestTerminated)レスポンスを受け取ることに頼ることはできない(RFC2543 に準拠する UAS はそのようなレスポンスを生成しないので)。オリジナルリクエストに対する最終レスポンス(Final response)が 64*T1 秒(T1 は 17.1.1.1 節で定義されている)以内にない場合、クライアントは、オリジナルトランザクションがキャンセルされたとみなすべきであり[SHOULD]、オリジナルリクエストをハンドリングするクライアントトランザクションを破棄するべきである[SHOULD]。

9.2 サーバの動作 (Server Behavior)

CANCEL メソッドは、サーバ側の TU が保留中のトランザクションをキャンセルすることを要求する。TU は CANCEL リクエストを取得し、リクエストのメソッドが CANCEL または ACK 以外の何かであると仮定して、17.2.3 節のトランザクションマッチング手順を適用して、キャンセルされるトランザクションを決定する。マッチするトランザクションがキャンセルされるトランザクションである。

サーバでの CANCEL リクエスト処理は、サーバのタイプに依存する。ステートレスプロキシはそれをフォ

ワードするだろう。ステートフルプロキシはそれにレスポンスしてそれ自身でいくつかの CANCEL リクエストを生成するかもしれない。また UAS はそれにレスポンスするだろう。プロキシでの CANCEL の扱いについては 16.10 節参照のこと。

UAS は最初に、8.2 節に述べられている一般的な UAS 処理にしたがって CANCEL リクエストを処理する。しかしながら、CANCEL リクエストはホップバイホップであり、再提示できないので、Authorization ヘッダフィールドに適切な信用証明書を得るためにサーバがチャレンジすることはできない。CANCEL リクエストは Require ヘッダフィールドも含まないことにも注意すること。

上記の手順に従って CANCEL にマッチするトランザクションを UAS が見つけられなかった場合、UAS は CANCEL に対して 481(Call Leg/Transaction Does NotExist)でレスポンスするべきである[SHOULD]。オリジナルリクエストに対するトランザクションがまだ存在する場合、CANCEL リクエスト受信時の UAS の動作は、それがオリジナルリクエストに対して既に最終レスポンス(Final response)を送ってしまったかどうかに依存する。最終レスポンス(Final response)を送ってしまった場合、CANCEL リクエストは、オリジナルリクエストの処理にも、いかなるセッションステートにも、オリジナルリクエストに対して生成されたレスポンスにも何ら影響を与えない。UAS がオリジナルリクエストに対して最終レスポンス(Final response)を発行していなかった場合、その動作はオリジナルリクエストのメソッドに依存する。オリジナルリクエストが INVITE だった場合、UAS は INVITE に対して直ちに 487(Request Terminated)でレスポンスするべきである[SHOULD]。CANCEL リクエストは、この仕様で定義される他のどのメソッドとのトランザクション処理にも、影響を与えない。

CANCEL が既存のトランザクションにマッチする限り、オリジナルリクエストのメソッドに関係なく、UAS は CANCEL リクエスト自体に 200(OK)レスポンスで答える。このレスポンスは、CANCEL に対するレスポンスの To タグとオリジナルリクエストに対するレスポンスの To タグが同じであるべきである[SHOULD]ということに注意して、8.2.6 節で述べられている手順にしたがって構築される。CANCEL に対するレスポンスはトランスミッションのためにサーバトランザクションに渡される。

１０．登録 (Registrations)

10.1 概要 (Overview)

SIP は発見能力(discovery capability)を提供する。ユーザが他のユーザとセッションを開始することを望む場合、SIP は、着呼側ユーザに到達できる現在のホストを発見しなければならない。この発見プロセスは、リクエストの受信、ユーザのロケーションについての知識に基づいてそれをどこに送るかの決定、そしてそれをそこに送ることに責任がある、プロキシサーバおよびリダイレクトサーバのような SIP のネットワークエレメントによって達成されることが多い。これを行うために、SIP のネットワークエレメントは、特定のドメインのためにアドレスのバインディングを提供するロケーションサービスとして知られる抽象サービスを検索する。これらのアドレスバインディングは、送られてきた SIP URI または SIPS URI(たとえば、sip:bob@Biloxi.com)を何らかの形で希望するユーザに「近い」一つ以上の URI(たとえば、sip:bob@engineering.Biloxi.com)にマップする。つまるところ、プロキシは、希望する受信者が現在いるところのユーザエージェントに、受け取った URI をマップするロケーションサービスを検索することになる。

登録は、特定のドメインのためのロケーションサービス内に、 address-of-record(AOR)の URI と一つ以上のコンタクトアドレスを関連付けるバインディングを生成する。したがって、そのドメインのためのプロキシが、Request-URI が address-of-record(AOR)にマッチするリクエストを受け取るとき、そのプロキシは

リクエストを、その address-of-record(AOR)に登録されているコンタクトアドレスにフォワードする。一般的に、あるドメインのロケーションサービスに address-of-record(AOR)を登録することは、その address-of-record(AOR)に対するリクエストがそのドメインにルーティングされるときにのみ意味をなす。ほとんどの場合、これは登録のドメインが address-of-record(AOR)の URI のドメインにマッチする必要があることを意味する。

ロケーションサービスのコンテンツを確定するための方法がいろいろとある。一つの方法は、管理上のものである。上記の例では、企業データベースへのアクセスを通して、Bob がエンジニアリング部門のメンバーであることがわかる。その一方で、SIP は、バインディングを明確に生成するために、UA のためのメカニズムを提供する。このメカニズムは登録として知られている。

登録は、レジストラサーバとして知られる特別なタイプの UAS に REGISTER リクエストを送ることを必要とする。レジストラサーバは、ドメインのためのロケーションサービスのフロントエンドとして動作し、REGISTER リクエストのコンテンツに基づいてマッピングを読み書きする。このロケーションサービスは、その後通常は、そのドメインへのリクエストのルーティングに責任があるプロキシサーバによって検索される。

登録プロセス全体の図解は図 2 で与えられている。レジストラサーバとプロキシサーバは、ネットワーク上の一つのデバイスで務めることができる論理的な役割であることに注意すること。明確化のために、これらの 2 つのサーバは図では分割されている。またこれら 2 つのサーバが別々のエレメントの場合、UA はレジストラサーバに到達するためにプロキシサーバを介してリクエストを送信してもよいことに注意すること。

SIP はロケーションサービスを実装するための特定のメカニズムを指示しない。

唯一の要求は、あるドメインのためのレジストラサーバはロケーションサービスへのデータを読み書きできなくてはならず[MUST]、そのドメインのためのプロキシまたはリダイレクトサーバはそのデータを読むことができなくてはならない[MUST]ということである。レジストラサーバは、同じドメインのための特定の SIP プロキシサーバと同じ場所に配置 してもよい[MAY]。

10.2 REGISTER リクエストの構築 (Constructing the REGISTER Request)

REGISTER リクエストはバインディングの追加、削除、クエリーを行う。REGISTER リクエストは、address-of-record(AOR)と一つ以上のコンタクトアドレスの間の新規バインディングを追加できる。特定の address-of-record(AOR)のための登録は適切に認可された第三者(third-party)が実行することができる。クライアントは以前のバインディングを削除したり、ある address-of-record に対してどのバインディングが現在適当かを決定するためにクエリーを行ったりすることもできる。

注記がなければ、REGISTER リクエストの構築と REGISTER リクエストを送るクライアントの動作は、8.1 節と 17.1 節で記述されている一般的な UAC の動作と同一である。

REGISTER リクエストはダイアログを確立しない。UAC は REGISTER リクエストに、8.1 節で述べられている既存のルートセットに基づいて Route ヘッダフィールドを含めてもよい[MAY]。Record-Route ヘッダフィールドは REGISTER リクエストまたはレスポンスにおいて何ら意味を持たず、存在した場合には無視されなくてはならない[MUST]。特に、UAC は REGISTER リクエストに対するいかなるレスポンス中の Record-Route ヘッダフィールドの存在/不在に基づいても、新しいルートセットを生成してはいけない[MUST NOT]。

以下のヘッダフィールドは、Contact を除いて、REGISTER リクエストに含めなくてはならない[MUST]。Contact ヘッダフィールドは含められてもよい[MAY]。

Request-URI:

Request-URI は、登録が行われるロケーションサービスのドメインを示す(例: sip:chicago.com)。SIP URI の userinfo と@コンポーネントが存在してはいけない[MUST NOT]。

To:

To ヘッダフィールドは、登録が生成、クエリー、あるいは修正される Adress-of-Record を含む。To ヘッダフィールドと Request- URI フィールドは､前者がユーザネームを含むので通常は異なる。この address-of-record (AOR)は SIP URI または SIPS URI でなくてはならない[MUST]。

From:

From ヘッダフィールドは、登録を行う人の address-of-record(AOR)を含む。リクエストが第三者(third-party)による登録でなければ、この値は To ヘッダフィールドと同じである。

Call-ID:

UAC からのすべての登録は、特定のレジストラサーバに送られる登録に対して、同じ Call-ID ヘッダフィールド値を使用するべきである[SHOULD]。

同一のクライアントが異なる Call-ID 値を使用した場合、レジストラサーバは、遅延した REGISTER リクエストが、順番が狂って到着したのかあるいはそうでないのか検知できない。

CSeq:

CSeq 値は REGISTER リクエストの適切な並び順を保証する。UA は、同じ Call-ID を持つ各 REGISTER リクエストに対して CSeq 値を１ずつインクリメントしなくてはならない[MUST]。

Contact:

REGISTER リクエストは､アドレスバインディングを含むゼロ個以上の値を持つ Contact ヘッダフィールドを含んでもよい[MAY]。

UA は、レジストラサーバから以前に送った登録への最終レスポンス(Final response)を受け取るか、以前の REGISTER リクエストがタイムアウトするまで、新たな登録(すなわち、再送とは違い新しい Contact ヘッダフィールド値を含む)を送ってはいけない[MUST NOT]。

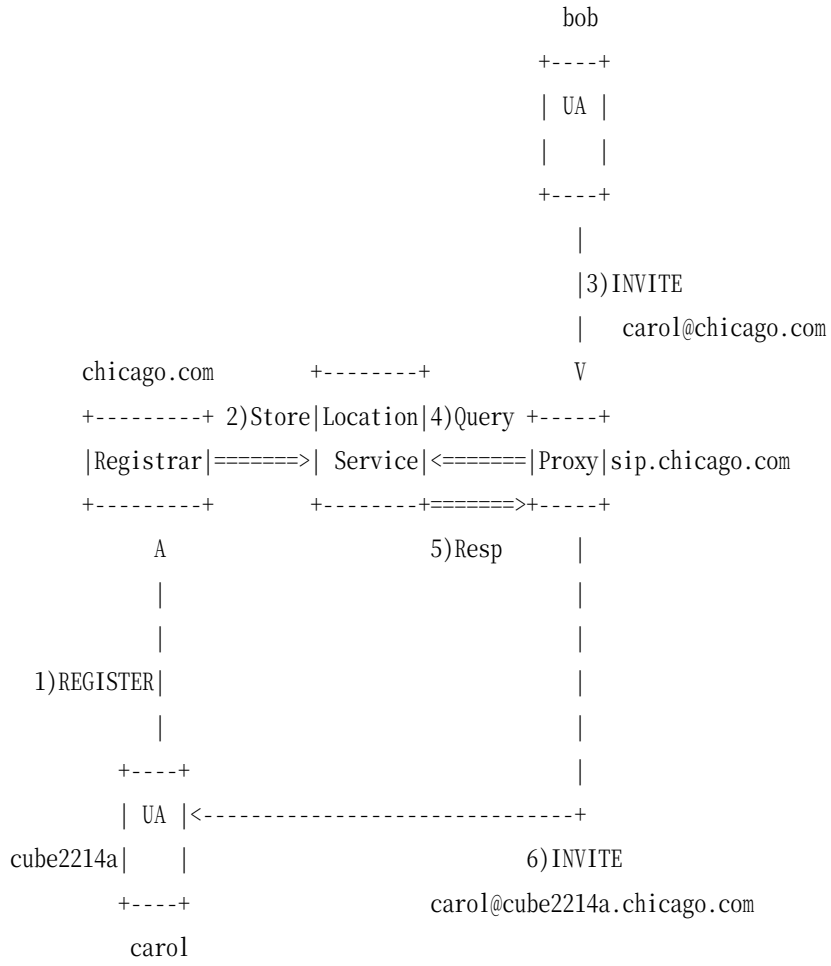

図 2: REGISTER の例

以下の Contact ヘッダパラメータは REGISTER リクエストにおいて特別な意味を持つ。

action:
RFC2543 からの action パラメータは反対されている。UAC は action パラメータを使用するべきではない[SHOULD NOT]。

expires:
expiresパラメータは、どれだけの期間バインディングが有効であってほしいとUAが望むかを示す。値は秒を表す数字である。このパラメータが提供されない場合、Expires ヘッダフィールドの値が代わりに使用される。実装では、 2^{32}-1(4294967295 秒または 136 年)よりも大きい値を 2^{32}-1 と等価なものとして扱ってもよい[MAY]。不正な形式のものは 3600 と等価なものとして扱うべきである[SHOULD]。

10.2.1 バインディングの追加 (Adding Bindings)

レジストラサーバに送られた REGISTER リクエストは address-of-record(AOR)に対する SIP リクエストがフォワードされるべき[SHOULD]コンタクトアドレス(一つまたは複数)を含む。address-of-record(AOR)は、

REGISTER リクエストの To ヘッダフィールドに含まれる。

リクエストの Contact ヘッダフィールド値は一般的に特定の SIP エンドポイントを識別する SIP URI または SIPS URI からなる(たとえば、sip:carol@cube2214a.chicago.com)が、どのような URI スキームで使用してもよい[MAY]。SIP UA は、たとえば、電話番号(tel URL で。RFC2806 参考文献[9])または E メールアドレス(mailto: URL で。RFC2368 参考文献[32])を address-of-record(AOR)に対する Contact として選択することができる。

たとえば、Carol(address-of-record(AOR)が sip:carol@chicago.com)は、chicago.com ドメインのレジストラサーバに登録するだろう。彼女の登録はその後、chicago.com ドメインのプロキシサーバによって Carol の address-of-record(AOR)に対するリクエストを彼女の SIP エンドポイントにルーティングするために使用されるだろう。

クライアントがいったんレジストラサーバでバインディングを確立すると、それは必要に応じて、既存のバインディングに対する新しいバインディングまたは修正を含む登録を送信してもよい[MAY]。REGISTER リクエストに対する 2xx レスポンスは、このレジストラサーバでこの address-of-record(AOR)に対して登録されているバインディングの完全なリストを Contact ヘッダフィールドに含む。

REGISTER リクエストの To ヘッダフィールドの address-of-record(AOR)が SIPS URI の場合、そのリクエストのすべての Contact ヘッダフィールド値も SIPS URI であるべきである[SHOULD]。クライアントは、SIPS address-of-record(AOR)の下には非 SIPS URI だけを登録するべきである(コンタクトアドレスで示されるリソースのセキュリティが他の方法で保証されているときは)。これは、SIP 以外のプロトコルを呼び出す URI か、または TLS 以外のプロトコルで安全を確保されている SIP デバイスに当てはまるかもしれない。

登録はすべてのバインディングをアップデートする必要がない。一般的に、UA はそれ自身のコンタクトアドレスだけをアップデートする。

10.2.1.1 Contact **アドレスの有効期限間隔の設定** (Setting the Expiration Interval of Contact Addresses)

クライアントが REGISTER リクエストを送るとき、クライアントはどれだけの間登録が有効であってほしいと望むかを示す有効期限間隔を提案してもよい[MAY](10.3 節で記述されているように、レジストラサーバはそれ自身のローカルポリシーに基づいて実際の時間間隔を選択する)。

クライアントがバインディングの有効期限間隔を提案することができる二つの方法がある。Expires ヘッダフィールドあるいは Contact ヘッダの expires パラメータを通してである。後者は、一つの REGISTER で二つ以上のバインディングが与えられたときに、バインディングごとに有効期限間隔を提案することを可能にし、一方、前者は、expires パラメータを含まないすべての Contact ヘッダフィールド値のための有効期限間隔を提案する。

REGISTER に、提案される有効期限時間を表現するいずれのメカニズムも存在しない場合、クライアントはサーバに選択させるという要望を示している。

10.2.1.2 Contact **アドレス間のプリファレンス** (Preference among Contact Assresses)

REGISTER リクエストで二つ以上の Contact が送られる場合、登録を行う UA はこれらの Contact ヘッダフ

ィールド値の中のすべての URI を、To フィールドに存在する address-of-record(AOR)に関連付けることを意図している。このリストは、Contact ヘッダフィールド中の q パラメータで優先順位付けできる。q パラメータは、この address-of-record(AOR)に対する他のバインディングと比べて、個々の Contact ヘッダフィールド値に対する相対的なプリファレンスを示す。16.6 節で、プロキシサーバがこのプリファレンス値をどのように使用するかについて記述する。

10.2.2 バインディングの削除 (Removing Bindings)

登録はソフトステートであり、リフレッシュされなければ期限切れになるが、明示的に削除することもできる。クライアントは、10.2.1 節で述べられるようにレジストラサーバによって選択された有効期限間隔に、影響を与えることを試みることができる。UA は REGISTER リクエスト中のコンタクトアドレスに 0 という有効期限間隔を指定することで、バインディングの即時削除を要求する。バインディングがその有効期限間隔が経過する前に削除されることが可能なように、UA はこのメカニズムをサポートするべきである[SHOULD]。

REGISTER 固有の Contact ヘッダフィールド値である「*」は、すべての登録に適用されるが、Expires ヘッダが 0 という値で存在しなければ使用してはいけない[MUST NOT]。

Contact ヘッダフィールド値「*」の使用は、登録を行う UA が、ある address-of-record(AOR)に関連付けられたすべてのバインディングを、それらの正確な値を知ることなく、削除することを可能にする。

10.2.3 バインディングの取得 (Fetching Bindings)

REGISTER リクエストに対する成功レスポンス(success response)は、リクエストが Contact ヘッダフィールドを含んでいたかどうかに関わらず、既存のバインディングの完全なリストを含む。REGISTER リクエストに Contact ヘッダフィールドが存在しなかった場合、バインディングのリストは変更されない。

10.2.4 バインディングのリフレッシュ (Refreshing Bindings)

それぞれの UA は、それが以前に確立したバインディングをリフレッシュする責任を負う。UA は、他の UA によってセットアップされたバインディングをリフレッシュするべきではない[SHOULD NOT]。

レジストラサーバからの 200(OK)レスポンスは、現在のすべてのバインディングを列挙する Contact フィールドのリストを含む。UA は、19.1.4 節の比較ルールを使用して、各コンタクトアドレスがその UA が生成したものかどうかを確認するために比較を行う。その UA が生成したものであれば、expires パラメータまたは(それがなければ)Expires フィールド値にしたがって、有効期限時間間隔をアップデートする。UA はそれから、それの各バインディングに対して、有効期限間隔が経過する前に、REGISTER リクエストを発行する。UA は、いくつかのアップデートを一つの REGISTER リクエストにまとめてもよい[MAY]。

UA は、一つのブートサイクルの間、すべての登録に対して同一の Call-ID を使用するべきである[SHOULD]。登録のリフレッシュは、リダイレクトしたのでなければ、オリジナルの登録と同じネットワークアドレスに送られるべきである[SHOULD]。

10.2.5 内部クロックの設定 (Setting the Internal Clock)

REGISTER リクエストへのレスポンスが Date ヘッダフィールドを含む場合、クライアントは何らかの内部クロックを設定するための現在時間を知るために、このヘッダフィールドを使用してもよい[MAY]。

10.2.6 **レジストラサーバの発見** (Discovering a Register)

UA は登録を送るアドレスを決定するために 3 つの方法を使用できる。あらかじめ設定することによって、address-of-record(AOR)を使用することによって、およびマルチキャストによってである。この仕様の適用範囲外の方法で、UA にレジストラサーバのアドレスを設定することができる。設定済みのレジストラサーバアドレスがない場合、UA は Request-URI として address-of-record(AOR)のホスト部分を使い、通常の SIP サーバのロケーション特定メカニズム(参考文献[4])を使用して、リクエストをそこに送るべきである[SHOULD]。たとえば、ユーザ“ sip:carol@chicago.com ”の UA は、REGISTER リクエストを“ sip:chicago.com ”に宛てて送る。

最後に、UA はマルチキャストを使用するように設定できる。マルチキャストによる登録は、「すべての SIP サーバ」のウェルノンマルチキャストアドレスである sip.mcast.net(IPv4 では 224.0.1.75)に宛てて送られる。IPv6 のウェルノンマルチキャストアドレスはまだ割り当てられていない。この割り当ては必要になったときにドキュメント化されるだろう。SIP UA はそのアドレスを待ち受けして他のローカルユーザのロケーションを認識するために使用してもよい[MAY](参考文献[33]参照)。しかしながら、それらはリクエストに応答しない。

> マルチキャストによる登録は、たとえば複数の会社が同一の LAN を共有するような、ある種の環境では適切ではないかもしれない。

10.2.7 **リクエストの送信** (Transmitting a Request)

いったん REGISTER メソッドが構築され、メッセージのデスティネーションが識別されたら、UAC は REGISTER をトランザクションレイヤに渡すために、8.1.2 節で記述した手順にしたがう。

REGISTER に対するレスポンスが得られなかったためにトランザクションレイヤがタイムアウトエラーを返却する場合、UAC は同じレジストラサーバにすぐに再試行するべきではない[SHOULD NOT]。

> 即時再試行もまたタイムアウトする可能性が高い。タイムアウトを引き起こした状況が正されるための理にかなう時間だけ待つことは、ネットワーク上の不要な負荷を軽減する。特定の時間間隔は指示されていない。

10.2.8 **エラーレスポンス** (Error Responses)

UA が 423(Interval Too Brief)レスポンスを受け取る場合、UA は、REGISTER リクエストのすべてのコンタクトアドレスの有効期限間隔を、423(Interval Too Brief)レスポンスの Min-Expires ヘッダフィールドの有効期限間隔以上にしたあとで、登録を再試行してもよい[MAY]。

10.3 REGISTER **リクエスト処理** (Processing REGISTER Requests)

レジストラサーバは、REGISTER リクエストにレスポンスし、それの管理ドメイン内のプロキシサーバとリダイレクトサーバがアクセス可能なバインディングのリストを保持する UAS である。レジストラサーバは、8.2 節と 17.2 節にしたがってリクエストをハンドリングするが、REGISTER リクエストのみを受け入れる。レジストラサーバは 6xx レスポンスを生成してはいけない[MUST NOT]。

レジストラサーバは必要に応じて REGISTER リクエストをリダイレクトしてもよい[MAY]。レジストラサーバにとって一般的な一つの用法は、マルチキャスト REGISTER リクエストをレジストラサーバ自身のユニキ

ャストインターフェースに 302(Moved Temporarily)レスポンスでリダイレクトするために、マルチキャストインターフェースを待ち受け中になるだろう。

レジストラサーバは、REGISTER リクエストに Record-Route ヘッダフィールドが含まれていた場合には、それを無視しなくてはならない[MUST]。レジストラサーバは、REGISTER リクエストに対するいかなるレスポンスにも Record-Route ヘッダフィールドを含めてはいけない[MUST NOT]。

REGISTER を未知のリクエストとして扱い、Record-Route ヘッダフィールド値を追加したプロキシサーバをトラバースしたリクエストを、レジストラサーバが受け取るかもしれない。

レジストラサーバはそれがバインディングを保持するドメインのセットを知っていなければならない(たとえば、設定によって)。REGISTER リクエストはレジストラサーバに受け取られた順番に処理されなくてはならない[MUST]。REGISTER リクエストはまた、非分割的(atomically)に処理されなくてはならない[MUST]。つまり、個々の REGISTER リクエストは完全に処理されるか、あるいはまったく処理されないかのいずれかである。各 REGISTER メッセージは、他のいかなる登録またはバインディングの変更からも独立して処理されなくてはならない[MUST]。

REGISTER リクエストを受け取るとき、レジストラサーバは以下のステップにしたがう。

1. レジストラサーバは、Request-URI で識別されるドメインのためのバインディングにアクセスできるかどうか確定するために Request-URI を検査する。アクセスできず、かつ、レジストラサーバがプロキシサーバとしても動作する場合、レジストラサーバは、16 節で記述されているメッセージをプロキシするための通常の動作にしたがって、宛先にされているドメインにリクエストをフォワードするべきである[SHOULD]。

2. レジストラサーバがすべての必要な拡張をサポートすることを保証するために、レジストラサーバは 8.2.2 節で UAS のために記述したように Require ヘッダフィールド値を処理しなくてはならない[MUST]。

3. レジストラサーバは UAC を認証するべきである[SHOULD]。SIP ユーザエージェントの認証のためのメカニズムは 22 節に記述している。登録の動作は、SIP のための通常の認証フレームワークを決してオーバーライドしない。認証メカニズムが利用できない場合、レジストラサーバは、リクエストの発信元のアイデンティティを主張するものとして From のアドレスを採用してもよい[MAY]。

4. レジストラサーバは、認証されたユーザがこの address-of-record(AOR)に対する登録を修正することが認可されているかどうか確定するべきである[SHOULD]。たとえば、レジストラサーバは、ユーザがバインディングを修正するための認可を持つ address-of-record(AOR)のリストにユーザ名をマップした認可データベースを検索するかもしれない。認証されたユーザがバインディングを修正することを認可されていない場合、レジストラサーバは 403(Forbidden)を返送して残りのステップをスキップしなくてはならない[MUST]。

第三者(third-party)による登録をサポートするアーキテクチャーでは、 一つのエンティティが、複数の address-of-record (AOR)に関連付けられた登録をアップデートする責任を負うかもしれない。

5. レジストラサーバはリクエストの To ヘッダフィールドから address-of-record(AOR)を抽出する。address-of-record(AOR)が Request-URI のドメインに対して有効でない場合、レジストラサーバは 404(Not Found)レスポンスを送って残りのステップをスキップしなくてはならない[MUST]。それからその URI は正規化形式(canonical form)に変換されなくてはならない[MUST]。そうするために、(user-param を含む)すべての URI パラメータが削除されなくてはならず[MUST]そしてエスケープされたいかなるキャラクターもエスケープされていない形に変換されなくてはならない[MUST]。その結果はバインディングリストへのインデックスとして役に立つ。

6. レジストラサーバはリクエストが Contact ヘッダフィールドを含むかどうか確認する。含まれていなければ、最後のステップまでスキップする。Contact ヘッダフィールドが存在する場合、レジストラサーバは、特別な値「*」と Expires フィールドを含む一つの Contact フィールド値があるかどうか確認する。リクエストが追加の Contact フィールドまたはゼロ以外の有効期限時間を持つ場合、リクエストは無効であり、レジストラサーバは 400(Invalid Request)を返送して残りのステップをスキップしなくてはならない[MUST]。持たない場合には、レジストラサーバは、Call-ID がそれぞれのバインディングに対して保存してある値と合致するかどうか確認する。合致しない場合、レジストラサーバはそのバインディングを削除しなくてはならない[MUST]。合致する場合、レジストラサーバはリクエストの CSeq がそのバインディングのために保存されている値よりも大きいときにのみ、そのバインディングを削除しなくてはならない[MUST]。そうでなければ、アップデートは中止されなくてはならず[MUST]、また、リクエストは失敗する。

7. レジストラサーバは、Contact ヘッダフィールドの各コンタクトアドレスを順番に処理する。各アドレスに対して、レジストラサーバは以下のように有効期限間隔を決定する。

   - フィールド値が expires パラメータを持っている場合、その値を要求された有効期限として扱わなくてはならない[MUST]。

   - そのようなパラメータはないがリクエストが Expires ヘッダフィールドを持つ場合、その値を要求された有効期限として扱わなくてはならない[MUST]。

   - どちらもない場合、ローカルで設定されている初期値を要求された有効期限として扱わなくてはならない[MUST]。

レジストラサーバは要求された有効期限間隔よりも短い有効期限を選択してもよい[MAY]。要求された有効期限間隔がゼロより大きく、かつ(AND)1 時間未満で、かつ(AND)レジストラサーバで設定されている最小値よりも小さい場合にのみ、レジストラサーバは 423(Interval Too Brief)レスポンスで登録を拒否してもよい[MAY]。このレスポンスは、レジストラサーバが受け取ることを望む有効期限間隔の最小値を示す Min-Expires ヘッダフィールドを含まなくてはならない[MUST]。それから残りのステップをスキップする。

レジストラサーバが登録間隔(registration interval)を設定することを認めることは、レジストラサーバが保持することを必要とする状態を制限し、登録が古くなる可能性を減らす一方で、過剰に頻繁な登録のリフレッシュからレジストラサーバを保護する。登録の有効期限間隔は、サービスの生成におい

て頻繁に使用される。一つの例は、あるターミナルでユーザが短時間の間だけ有効であるときの、follow-me サービスである。したがって、レジストラサーバは短期間の登録を受け入れるべきである。リクエストは、その間隔があまりにも短いためにリフレッシュがレジストラサーバのパフォーマンスを落とす場合にのみ、拒否されるべきである。

それから、各アドレスに対して、レジストラサーバは URI 比較ルールを使用して現在のバインディングのリストを検索する。バインディングが存在しない場合、それは仮に追加される。バインディングが存在する場合、レジストラサーバは Call-ID 値を確認する。既存のバインディング中の Call-ID 値がリクエストの Call-ID 値と異なる場合、そのバインディングは、有効期限時間がゼロであれば削除され、そうでなければアップデートされなくてはならない[MUST]。Call-ID 値が同じ場合、レジストラサーバは CSeq 値を比較する。CSeq 値が既存のバインディングのものより大きい場合、それは上記のようにバインディングをアップデートするか削除しなくてはならない[MUST]。大きくない場合、アップデートは中止されなくてはならず[MUST]、そのためリクエストは失敗する。

このアルゴリズムは、同一の UA からくる順番が狂ったリクエストが無視されることを確実にする。

各バインディングレコードは、リクエストから Call-ID と CSeq 値を記録する。

バインディングのアップデートは、バインディングのすべてのアップデートと追加が成功する場合にだけ、コミットされなくてはならない(すなわち、プロキシまたはリダイレクトサーバから見るようにされる)[MUST]。一つでも失敗した場合(たとえば、バックエンドデータベースのコミットが失敗したため)、リクエストは 500 (Server Error)レスポンスで失敗しなくてはならず[MUST]、すべての仮のバインディングアップデートは削除されなくてはならない[MUST]。

8. レジストラサーバは 200(OK)レスポンスを返送する。レスポンスは現在のすべてのバインディングを列挙する Contact ヘッダフィールド値を含まなくてはならない[MUST]。各 Contact 値は、レジストラサーバによって選択された有効期限間隔を示す expires パラメータを含まなくてはならない[MUST]。レスポンスは Date ヘッダフィールドを含むべきである[SHOULD]。

１１．能力のクエリー (Querying for Capabilities)

SIP メソッドの OPTIONS は、UA が、他の UA またはプロキシサーバにそれの能力を問い合わせることを許可する。これは、クライアントが、パーティを呼び出さずに(without ringing)サポートされているメソッド、コンテントタイプ、拡張、CODEC などについての情報を発見することを可能にする。たとえば、クライアントが INVITE に、デスティネーションの UAS がサポートしているという確証がないオプションをリストした Require ヘッダフィールドを挿入する前に、そのクライアントは、このオプションが Supported ヘッダフィールドで返送されるかどうか知るために、デスティネーションの UAS に OPTIONS で問い合わせることができる。すべての UA は OPTIONS メソッドをサポートしなくてはならない[MUST]。

OPTIONS リクエストのターゲットは、Request-URI は他の UA または SIP サーバを識別することができる Request-URI によって識別される。OPTIONS がプロキシサーバに宛てて送られる場合、Request-URI は、それが REGISTER リクエストに対して設定される方法と同じように、ユーザ部分なしで設定される。

あるいは、Max-Forwards ヘッダフィールドの値が 0 で OPTIONS リクエストを受け取るサーバは、

Request-URI に関係なくそのリクエストにレスポンスしてもよい[MAY]。

この動作は HTTP/1.1 では一般的である。この動作は、インクリメントされた Max-Forwards 値で連続して OPTIONS リクエストを送ることによって、個々のホップサーバの能力を確認するための「traceroute」機能として使用することができる。

一般的な UA 動作の場合のように、トランザクションレイヤは、OPTIONS が何のレスポンスも生み出さない場合にタイムアウトエラーを返却することができる。これは、ターゲットは到達不可能でそれゆえ利用できないということを示すかもしれない。

OPTIONS リクエストは、確立されたダイアログで後ほど使用できる能力を相手に問い合わせるために、そのダイアログの一部として送ってもよい[MAY]。

11.1 OPTIONS リクエストの構築 (Construction of OPTIONS Request)

OPTIONS リクエストは、8.1.1 節で議論した SIP リクエストのための標準ルールを使用して構築される。

OPTIONS の中に Contact ヘッダフィールドが存在してもよい[MAY]。

UAC がレスポンスで受け取りたいと望むメッセージボディのタイプを示すために、Accept ヘッダフィールドが含まれるべきである[SHOULD]。一般的に、これは、UA のメディア能力(たとえば SDP(application/sdp))を記述するために使用されるフォーマットに設定される。

OPTIONS リクエストに対するレスポンスは、オリジナルリクエストの Request-URI が適用範囲にされると仮定される。しかしながら、確立されたダイアログの一部として OPTIONS が送られるときにのみ、以降のリクエストが OPTIONS に対するレスポンスを生成したサーバに受け取られることが保証される。

OPTIONS リクエストの例:

```
OPTIONS sip:carol@chicago.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
Max-Forwards: 70
To: <sip:carol@chicago.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:alice@pc33.atlanta.com>
Accept: application/sdp
Content-Length: 0
```

11.2 OPTIONS リクエストの処理 (Processing of OPTIONS Request)

OPTIONS に対するレスポンスは、8.2.6 節で議論したように SIP レスポンスの標準ルールを使用して構築される。選択されるレスポンスコードは、リクエストが INVITE だったときに選択されるものと同じでなくてはならない[MUST]。すなわち、UAS が呼を受け取る準備ができていれば 200(OK)が返送され、UAS がビジーであれば 486(Busy Here)が返送されるなどである。これは、OPTIONS リクエストが UAS の基本状態を確定す

るために使用されることを可能にする。この状態は、UAS が INVITE リクエストを受け入れるかどうかの印とすることができる。

ダイアログ内で受け取った OPTIONS リクエストは、ダイアログ外で構築されたものと同じ 200(OK)レスポンスを生成し、かつ、ダイアログに何ら影響を与えない。

OPTIONS のこの使用方法には、OPTIONS リクエストと INVITE リクエストのプロキシでのハンドリングの違いを原因とする制限がある。フォークされた INVITE が複数の 200(OK)レスポンスを返送することになる一方、フォークされた OPTIONS はただ一つの 200(OK)レスポンスを返すことになる。これは非 INVITE ハンドリングを使用してプロキシで処理されるためである。規範となる詳細については 16.7 節参照のこと。

OPTIONS に対するレスポンスがプロキシサーバによって生成される場合、プロキシはサーバの能力をリストする 200(OK)を返送する。そのレスポンスはメッセージボディを含まない。

Allow、Accept、Accept-Encoding、Accept-Language、および Supported ヘッダフィールドは OPTIONS リクエストに対する 200(OK)レスポンス中に存在するべきである[SHOULD]。レスポンスがプロキシによって生成される場合、プロキシはメソッドを不可知なために意味が曖昧になるので Allow ヘッダフィールドは省略されるべきである[SHOULD]。Contact ヘッダフィールドは 200(OK)レスポンスに存在してもよく[MAY]、3xx レスポンスと同じ意味合い(semantics)を持つ。すなわち、それらはユーザに到達する選択肢となる名前とメソッドのセットをリストすることができる。Warning ヘッダフィールドが存在してもよい[MAY]。

メッセージボディは送られてもよく[MAY]、それのタイプは OPTIONS リクエストの Accept ヘッダフィールドによって確定される(Accept ヘッダフィールドが存在しない場合には application/sdp が初期値である)。タイプがメディア能力を記述するものを含む場合、UAS はその目的のために、そのレスポンスにボディを含めるべきである[SHOULD]。application/sdp の場合にそのようなボディを構築することについての詳細は、参考文献[13]に記述されている。

UAS によって生成された OPTIONS に対するレスポンスの例(11.1 節のリクエストに対応する):

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
;received=192.0.2.4
To: <sip:carol@chicago.com>;tag=93810874
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:carol@chicago.com>
Contact: <mailto:carol@chicago.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
```

```
Content-Type: application/sdp
Content-Length: 274
```

(SDP は表示していない)

１２．ダイアログ (Dialogs)

ユーザエージェントのためのキーコンセプトはダイアログのコンセプトである。ダイアログは、しばらくの間持続する 2 つのユーザエージェント間の、ピア・トゥ・ピアの SIP リレーションシップを表す。ダイアログは、ユーザエージェント間のメッセージの順序付けと、それらの間の適切なルーティングを容易にする。ダイアログは、その中で SIP メッセージを解釈するためのコンテキストを表す。8 節では、ダイアログ外のリクエストとレスポンスのための、メソッドに依存しない UA 処理について議論した。本節では、それらのリクエストとレスポンスがダイアログを構築するためにどのように使用されるのか、そしてそれに続くリクエストとレスポンスがダイアログ内でどのように送られるのかを議論する。

ダイアログは各 UA において、Call-ID 値、ローカルタグ、およびリモートタグから成るダイアログ ID で識別される。ダイアログに関係する各 UA におけるダイアログ ID は同じではない。特に、一つの UA におけるローカルタグは、相手 UA におけるリモートタグに等しい。これらのタグは、一意なダイアログ ID の生成を容易にする不透明トークン(opaque token)である。

ダイアログ ID は、すべてのレスポンスおよび To フィールドに tag を含むいかなるリクエストとも関連付けられている。メッセージのダイアログ ID を計算するルールは、SIP エレメントが UAC であるか UAS であるかに依存する。UAC では、そのダイアログ ID の Call-ID 値はメッセージの Call-ID に設定され、リモートタグはメッセージの To フィールドの tag に設定され、ローカルタグはメッセージの From フィールドの tag に設定される(これらのルールはリクエストとレスポンスの両方に適用される)。UAS については予想できるように、ダイアログ ID の Call-ID 値はメッセージの Call-ID に設定され、リモートタグはメッセージの From フィールドの tag に設定され、ローカルタグはメッセージの To フィールドの tag に設定される。

ダイアログは、そのダイアログ内で更にメッセージ送信を行うために必要とされる、状態の特定の要素を含む。この状態は、ダイアログ ID、ローカルシーケンス番号(UA からその相手へのリクエストを順番付けするために使用される)、リモートシーケンス番号(相手から UA へのリクエストを順番付けするために使用される)、ローカル URI、リモート URI、リモートターゲット、「secure」というブーリアンフラグ、およびルートセット(順番に並べられた URI のリスト)から成る。ルートセットは、相手にリクエストを送るためにトラバースする必要があるサーバのリストである。ダイアログは、暫定レスポンス(provisional response)で生成されたときになる「early」状態になることができ、その後 2xx 最終レスポンス(Final response)が到着するときに「confirmed」ステートに移行する。その他のレスポンスに対して、またはそのダイアログ上でまったくレスポンスが到着しない場合には、early ダイアログは終了する。

12.1 ダイアログの生成 (Creation of a Dialog)

ダイアログは、特定のメソッドを持つリクエストに対する失敗ではないレスポンスの生成を通して生成される。この仕様では、(リクエストが INVITE だった)To tag を持つ 2xx レスポンスおよび 101 から 199 レスポンスのみがダイアログを確立するだろう。リクエストに対する非最終レスポンス(non-final response)で確立されたダイアログは、early ステートにあり、それは early ダイアログと呼ばれる。拡張でダイアログを生成するための他の方法を定義してもよい[MAY]。13 節で、INVITE メソッド固有の更なる詳細について述

べる。ここでは、メソッドに依存しないダイアログステートの生成のためのプロセスを記述する。

UA は以下に述べられるように、ダイアログ ID コンポーネントに値を割り当てなくてはならない[MUST]。

12.1.1 UAS の動作 (UAS behavior)

UAS が、ダイアログを確立するレスポンス(たとえば、INVITE に対する 2xx)でリクエストにレスポンスするとき、その UAS はリクエストからすべての Record-Route ヘッダフィールド値をレスポンスにコピーしなくてはならず[MUST](UAS が知っていようと知らなかろうと、URI、URI パラメータ、およびすべての Record-Route ヘッダフィールドパラメータを含む)、それらの値の順番も保持しなくてはならない[MUST]。UAS は、レスポンスに Contact ヘッダフィールドを追加しなくてはならない[MUST]。その Contact ヘッダフィールドは、UAS が、ダイアログ中のそれ以後のリクエスト(INVITE の場合は 2xx レスポンスに対する ACK を含む)にコンタクトしてほしいと望むアドレスを含む。通常、この URI のホスト部分は、そのホストの IP アドレスまたは FQDN である。Contact ヘッダフィールドで提供される URI は SIP URI または SIPS URI でなくてはならない[MUST]。ダイアログを開始したリクエストが、(それが存在したとして)Request-URI ヘッダフィールド値に、または先頭の Record-Route ヘッダフィールド値に SIPS URI を含んでいた場合、または Record-Route ヘッダフィールドが存在しないときに Contact ヘッダフィールドを含んでいた場合、レスポンス中の Contact ヘッダフィールドは SIPS URI でなくてはならない[MUST]。その URI はグローバルスコープを持つべきである[SHOULD](すなわち、このダイアログ外のメッセージでも同じ URI を使用できる)。同じように、INVITE の Contact ヘッダフィールド中の URI のスコープも、このダイアログに限定されない。そのためそれは、たとえこのダイアログ外であっても、UAC へのメッセージ中で使用できる。

それから UAS は、ダイアログの状態を構築する。この状態はダイアログの継続する間保持されなくてはならない[MUST]。

リクエストが TLS 上で到着し、Request-URI が SIPS URI を含んだ場合、「secure」フラグが TRUE に設定される。

ルートセットには、リクエストから順番にすべての URI パラメータを維持したまま取り出した、Record-Route ヘッダフィールド中の URI リストを設定しなくてはならない[MUST]。リクエストに Record-Route ヘッダフィールドが存在しない場合、ルートセットには空のセットを設定しなくてはならない[MUST]。このルートセットは、たとえそれが空でも、このダイアログの以降のリクエストのために、すべての pre-existing ルートセットをオーバーライドする。リモートターゲットには、リクエストの Contact ヘッダフィールドの URI を設定しなくてはならない[MUST]。

リモートシーケンス番号には、リクエストの CSeq ヘッダフィールドのシーケンス番号値を設定しなくてはならない[MUST]。ローカルシーケンス番号は、空でなくてはならない[MUST]。ダイアログ ID の呼識別子(call identifier)コンポーネントには、リクエストの Call-ID 値が設定されなくてはならない[MUST]。ダイアログ ID のローカルタグコンポーネントには、リクエスト(常に tag を含む)に対するレスポンスの To フィールドの tag が設定されなくてはならず[MUST]、ダイアログ ID のリモートタグコンポーネントには、リクエストの From フィールドの tag を設定しなくてはならない[MUST]。UAS は、From フィールドに tag を持たないリクエストを受け取ることに備えなくてはならない[MUST]。この場合、tag はヌル値を持つとみなされる。

これは、From の tag を義務付けていなかった RFC2543 との下位互換性を保つためである。

リモート URI には、From フィールドの URI を設定しなくてはならず[MUST]、ローカル URI には、To フィールドの URI を設定しなくてはならない[MUST]。

12.1.2 UAC の動作 (UAC behavior)

ダイアログを確立できるリクエスト(たとえば INVITE)を UAC が送るとき、それはリクエストの Contact ヘッダフィールドに、グローバルスコープ(すなわち、それと同じ SIP URI がこのダイアログ外のメッセージでも使用できる)を持つ SIP または SIPS URI を提供しなくてはならない[MUST]。リクエストが、値が SIPS URI である Request-URI または先頭の Route ヘッダフィールドを持つ場合、Contact ヘッダフィールドは SIPS URI を含まなくてはならない[MUST]。

UAC が、ダイアログを確立するレスポンスを受け取るとき、それはダイアログの状態を構築する。この状態は、そのダイアログが継続する間保持されなくてはならない[MUST]。

リクエストが TLS 上で送られ、Request-URI が SIPS URI を含んでいた場合、「secure」フラグが TRUE に設定される。

ルートセットには、レスポンスから逆順にすべての URI パラメータを維持したまま取り出した、Record-Route ヘッダフィールド中の URI リストを設定しなくてはならない[MUST]。レスポンスに Record-Route ヘッダフィールドが存在しない場合、ルートセットには空のセットを設定しなくてはならない[MUST]。このルートセットは、たとえそれが空でも、このダイアログの以降のリクエストのために、すべての pre-existing ルートセットをオーバーライドする。リモートターゲットには、レスポンスの Contact ヘッダフィールドの URI を設定しなくてはならない[MUST]。

ローカルシーケンス番号には、リクエストの CSeq ヘッダフィールドのシーケンス番号値を設定しなくてはならない[MUST]。リモートシーケンス番号は、空でなくてはならない[MUST](それは、リモート UA がダイアログ内でリクエストを送るときに確立される)。ダイアログ ID の呼識別子(call identifier)コンポーネントには、リクエストの Call-ID 値が設定されなくてはならない[MUST]。ダイアログ ID のローカルタグコンポーネントには、リクエストの From フィールドの tag が設定されなくてはならず[MUST]、ダイアログ ID のリモートタグコンポーネントには、レスポンスの To フィールドの tag を設定しなくてはならない[MUST]。UAC は、To フィールドに tag を持たないレスポンスを受け取ることに備えなくてはならない[MUST]。この場合、tag はヌル値を持つとみなされる。

これは、To の tag を義務付けていなかった RFC2543 との下位互換性を保つためである。

リモート URI には、To フィールドの URI を設定しなくてはならず[MUST]、ローカル URI には、From フィールドの URI を設定しなくてはならない[MUST]。

12.2 ダイアログ内のリクエスト (Requests within a Dialog)

2 つの UA 間でいったんダイアログが確立されると、どちらかが必要なときにそのダイアログ内で新規トランザクションを開始してもよい[MAY]。リクエストを送る UA は、そのトランザクションにおいて UAC の役割を果たす。リクエストを受け取る UA は、UAS の役割を果たす。これらの役割は、ダイアログを確立したトラ

ンザクションの間にUAが果たす役割とは異なるかもしれないことに注意すること。

ダイアログ内のリクエストはRecord-RouteおよびContactヘッダフィールドを含んでもよい[MAY]。しかしながら、これらのリクエストは、リモートターゲットURIを修正するかもしれないが、ダイアログのルートセットを修正する要因にはならない。具体的には、ターゲットリフレッシュリクエストでないリクエストはダイアログのリモートターゲットURIを修正しない。また、ターゲットリフレッシュリクエストであるリクエストはダイアログのリモートターゲットURIを修正する。INVITEで確立されたダイアログにおいては、定義されている唯一のターゲットリフレッシュリクエストはre-INVITE(14節参照)である。他の拡張は、その他の方法で確立されたダイアログのための別のターゲットリフレッシュリクエストを定義してもよい。

ACKはターゲットリフレッシュリクエストではない(NOT)ことに注意すること。

ターゲットリフレッシュリクエストはダイアログのリモートターゲットURIのみをアップデートし、Record-Routeから形成されたルートセットはアップデートしない。後者をアップデートすることは、RFC2543に準拠したシステムとの下位互換性に関する深刻な問題を招くことになる。

12.2.1UAC の動作 (UAC Behavior)

12.2.1.1 リクエストの生成 (Generating the Request)

ダイアログ内のリクエストは、そのダイアログの一部として保存されている状態の多くのコンポーネントを使用して構築される。

リクエストのToフィールドのURIには、ダイアログステートのリモートURIが設定されなくてはならない[MUST]。リクエストのToヘッダフィールドのtagには、ダイアログIDのリモートタグが設定されなくてはならない[MUST]。リクエストのFrom URIには、ダイアログステートのローカルURIが設定されなくてはならない[MUST]。リクエストのFromヘッダフィールドのtagには、ダイアログIDのローカルタグを設定しなくてはならない[MUST]。リモートタグの値がヌルの場合、tagパラメータはToヘッダフィールドから省略されなくてはならない[MUST]。ローカルタグの値がNullの場合、tagパラメータはFromヘッダフィールドから省略されなくてはならない[MUST]。

これ以降のリクエストにおけるオリジナルリクエストのToおよびFromヘッダフィールドのURIの利用は、URIをダイアログの特定のために使用していたRFC2543との下位互換性のためになされる。この仕様では、ダイアログの特定のためにtagだけが使用される。この仕様の今後の改定において、ダイアログ内リクエスト(mid-dialog request)にオリジナルのToとFromのURIを反映することを必須とすることは反対されるだろう。

リクエストのCall-IDには、そのダイアログのCall-IDが設定されなくてはならない[MUST]。ダイアログ内のリクエストは、厳密に単調に増加する連続したCSeqシーケンス番号(1ずつ増加する)をそれぞれの方向で含まなくてはならない[MUST](当然、ACKとCANCELを例外とする。それらの番号は承認されるリクエストまたはキャンセルされるリクエストに等しい)。したがって、ローカルシーケンス番号が空でない場合、ローカルシーケンス番号の値は1だけ増加しなくてはならず[MUST]、この値はCSeqヘッダフィールドに置かれなくてはならない[MUST]。ローカルシーケンス番号が空の場合、8.1.1.5節のガイドラインを使用して初期値が選択されなくてはならない[MUST]。CSeqヘッダフィールド値のメソッドフィールドは、リクエストのメソッドとマッチしなくてはならない[MUST]。

32 ビットの長さでは、クライアントは、繰り返しが必要になるまでに、単一の呼の中で 1 秒に 1 回のリクエストを約 136 年間生成できる。シーケンス番号の初期値は、それと同じ呼でそれ以降に送るリクエストで、繰り返しが起こらないように選ばれる。0 以外の初期値は、クライアントが時間に基づくシーケンス番号の初期値を使用することを可能にする。たとえば、クライアントは、32 ビット秒クロックの上位 31 ビット(31 most significant bits)をシーケンス番号の初期値として選ぶことができる。

UAC はリクエストの Request-URI と Route ヘッダフィールドを構築するために、リモートターゲットとルートセットを使用する。

ルートセットが空の場合、UAC は Request-URI にリモートターゲットの URI を置かなくてはならない[MUST]。UAC はリクエストに Route ヘッダフィールドを追加してはいけない[MUST NOT]。

ルートセットが空でなく、ルートセットの最初の URI が lr パラメータ(19.1.1 節参照)を含む場合、UAC は Request-URI にリモートターゲットの URI を置かなくてはならず[MUST]、ルートセット値を(すべてのパラメータも含めて)その順番で包含する Route ヘッダフィールドを含めなくてはならない[MUST]。

ルートセットが空でなく、ルートセットの最初の URI が lr パラメータを含まない場合、UAC はルートセットの最初の URI を、Request-URI に認められていないすべてのパラメータを取り去って、Request-URI に置かなくてはならない[MUST]。UAC は、残りのルートセット値を(すべてのパラメータも含めて)その順番で包含する Route ヘッダフィールドを追加しなくてはならない[MUST]。それから UAC は、Route ヘッダフィールドに最後の値として、リモートターゲットの URI を置かなくてはならない[MUST]。

たとえば、リモートターゲットが sip:user@remoteua で、ルートセットが以下の値を含む場合、

<sip:proxy1>,<sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>

リクエストは、以下の Request-URI と Route ヘッダフィールドとともに形成される。

```
METHOD sip:proxy1
Route: <sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>,<sip:user@remoteua>
```

ルートセットの最初の URI が lr パラメータを含まない場合、指示されたプロキシはこのドキュメントで述べられているルーティングメカニズムを理解せず、メッセージをフォワードする間に Request-URI を受け取った 最初の Route ヘッダフィールド値で置き換えて、RFC2543 で規定されているとおりに動作する。Route ヘッダフィールドの最後に Request-URI を置くことは、ストリクトルータを通るときにもその Request-URI の情報を維持する(リクエストがルースルータに到達したときに、それは Request-URI に返送される)。

UAC は、ダイアログ内のいかなるターゲットリフレッシュリクエストにも Contact ヘッダフィールドを含めるべきであり[SHOULD]、それを変更する必要がないなら、URI はダイアログ内のそれ以前のリクエストで使用されたものと同じであるべきである[SHOULD]。secure フラグが TRUE の場合、その URI は SIPS URI でなくてはならない[MUST]。12.2.2 節で議論されるように、ターゲットリフレッシュリクエストの Contact ヘッ

ダフィールドは、リモートターゲットのURIをアップデートする。これは、UAのアドレスがダイアログの継続期間中に変わる場合に、UAが新しいコンタクトアドレスを提供することを可能にする。

しかしながら、ターゲットリフレッシュリクエストでないリクエストは、ダイアログのためのリモートターゲットのURIに影響を与えない。

リクエストの残りの部分は8.1.1節で記述したように形成される。

リクエストが構築されるとすぐに、サーバのアドレスが計算されて、ダイアログ外のリクエストのためのものと同じ手順(8.1.2節)を使用してリクエストが送られる。

8.1.2節の手順は、通常、Routeヘッダフィールドの先頭の値またはRouteヘッダフィールドが存在しない場合にはRequest-URIによって示されるアドレスに、リクエストが送られるという結果になる。特定の制限にしたがうと、それらはリクエストが代替アドレス(たとえば、ルートセットで表されていないデフォルトのアウトバウンドプロキシ)に送られることを可能にする。

12.2.1.2 レスポンスの処理 (Processing the Responses)

UACはトランザクションレイヤからリクエストに対するレスポンスを受け取る。クライアントトランザクションがタイムアウトを返却する場合、これは408(Request Timeout)レスポンスとして処理される。

ダイアログ内で送られたリクエストに対する3xxレスポンスを受け取るUACの動作は、リクエストがダイアログ外で送られた場合と同じである。この動作は8.1.3.4節で記述された。

しかしながら、UACが代替のロケーションを試すとき、それはリクエストのRouteヘッダを構築するためにダイアログのルートセットを依然として使用する、ということに注意すること。

ターゲットリフレッシュリクエストに対して2xxレスポンスをUACが受け取るとき、それはダイアログのリモートターゲットURIをそのレスポンスのContactヘッダフィールド(もし存在すれば)のURIで置き換えなくてはならない[MUST]。

ダイアログ内のリクエストに対するレスポンスが481(Call/Transaction Does NotExist)または408(Request Timeout)の場合、UACはダイアログを終了するべきである[SHOULD]。そのリクエストに対して何らのレスポンスも受け取らない場合にも、UACはダイアログを終了するべきである[SHOULD](クライアントトランザクションはタイムアウトについてTUに通知するだろう)。

INVITEが開始したダイアログでは、ダイアログの終了はBYEを送ることからなる。

12.2.2 UASの動作 (UAS Behavior)

ダイアログ内で送られるリクエストは、他のいかなるリクエストとも同様、非分割(atomic)である。特定のリクエストがUASに受け入れられる場合、それに関連付けられるすべての状態の変更が行われる。リクエストが拒否される場合、状態の変更は行われない。

INVITEのようないくつかのリクエストは、状態のいろいろな部分に影響することに注意すること。

UAS はトランザクションレイヤからリクエストを受け取る。リクエストが To ヘッダフィールドに tag を持つ場合、UAS コアはそのリクエストに対応するダイアログ識別子(dialog identifier)を計算し、それを既存のダイアログと比較する。マッチするものがあれば、これはダイアログ内リクエスト(mid-dialog request)である。この場合、8.2 節で議論されているダイアログ外のリクエストのためのものと同じ処理ルールを、UAS は最初に適用する。

リクエストが To ヘッダフィールドに tag を持つが、ダイアログ識別子が既存のどのダイアログにもマッチしない場合、その UAS はクラッシュしてリスタートしたか、あるいは別の(おそらく故障した)UAS に対するリクエストを受け取ったのかもしれない(それらの UAS は、リカバリーを提供しているもう一方の UAS に対する To tag であったことを、その UAS が識別できるように、To tag を構築できる)。別の可能性は、やってくるリクエストが単純にミスルーティングされただけということである。To tag に基づいて、UAS はそのリクエストを受け入れることも拒否することもできる[MAY]。容認できる To tag へのリクエストを受け入れることは、堅牢性を提供し、それゆえ、ダイアログはたとえクラッシュしても存続する。この能力をサポートすることを望む UA は、リブートをまたいだとしても単調に増加する CSeq シーケンス番号を選択すること、ルートセットの再構築、および範囲外の RTP タイムスタンプとシーケンス番号を受け取ること、のようないくつかの問題を考慮しなければならない。

UAS が、ダイアログを再生成することを望まないので、そのリクエストを拒否したい場合は、481(Call/Transaction Does Not Exist)レスポンスでリクエストにレスポンスしてそれをサーバトランザクションに渡さなくてはならない[MUST]。

ダイアログの状態をどのような形であれ変更しないリクエストをダイアログ内で受け取ることがあるかもしれない(たとえば、OPTIONS リクエスト)。それらはダイアログ外で受け取られたかのように処理される。

リモートシーケンス番号が空の場合、それはリクエスト中の CSeq ヘッダのシーケンス番号値に設定されなくてはならない[MUST]。リモートシーケンス番号は空ではないが、リクエストのシーケンス番号がリモートシーケンス番号よりも小さい場合、そのリクエストは順番が狂っており、500(ServerInternal Error)レスポンスで拒否されなくてはならない[MUST]。リモートシーケンス番号が空でなく、リクエストのシーケンス番号がリモートシーケンス番号よりも大きい場合、そのリクエストは順番どおりである。CSeq シーケンス番号がリモートシーケンス番号より 2 以上大きいこともありうる。これはエラー状態ではなく、UAS は前回受け取ったリクエストよりも 2 以上大きな CSeq 値を持つリクエストを受け取って処理できるようになっているべきである[SHOULD]。それから UAS は、リモートシーケンス番号にリクエスト中の CSeq ヘッダフィールド値のシーケンス番号を設定しなくてはならない[MUST]。

> UAC によって生成されたリクエストにプロキシがチャレンジする場合、UAC は信用証明書を持つリクエストを再提出しなければならない。再提出されたリクエストは新しい CSeq 番号を持つ。UAS は最初のリクエストを見ることはないので、CSeq 番号が飛んでいることに気付くだろう。このような番号の飛びは、いかなるエラー状態を表すものでもない。

UAS がターゲットリフレッシュリクエストを受け取るときは、ダイアログのリモートターゲット URI をそのリクエストの Contact ヘッダフィールドの URI(もし存在すれば)で置き換えなくてはならない[MUST]。

12.3 **ダイアログの終了** (Termination of a Dialog)

ダイアログ外のリクエストが非 2xx 最終レスポンス(Final response)を生成する場合、メソッドに依存しないで、そのリクエストに対する暫定レスポンス(provisional response)を介して生成されたいかなる early ダイアログも終了される。confirmed ダイアログを終了するためのメカニズムはメソッド固有である。この仕様では、BYE メソッドがセッションとそれに関連付けられたダイアログを終了する。詳細については 15 節参照のこと。

１３．セッションの開始 (Initiating a Session)

13.1 **概要** (Overview)

ユーザエージェントクライアントがセッションの開始を望むとき(たとえば、オーディオ、ビデオ、またはゲーム)、それは INVITE リクエストを作成する。INVITE リクエストはサーバにセッションの確立を依頼する。このリクエストは、最終的にその招待を受け入れる可能性があるひとつ以上の UAS に到達するように、プロキシによってフォワードされるかもしれない。これらの UAS は、その招待を受け入れるかどうかユーザに問い合わせる必要があることが多い。しばらくして、それらの UAS は 2xx レスポンスを送ることでその招待を受け入れることができる(そのセッションが確立されることを意味する)。招待が受け入れられなかった場合、拒否の理由によって、3xx、4xx、5xx、あるいは 6xx レスポンスが送られる。最終レスポンス(Final response)を送る前に、UAS は、着信側ユーザにコンタクトしている進捗状態を UAC に報告するために、暫定レスポンス(provisional response)(1xx)を送ることもできる。

おそらくひとつ以上の暫定レスポンス(provisional response)を受け取った後、UAC はひとつ以上の 2xx レスポンスまたはひとつの非 2xx 最終レスポンス(Final response)を受け取る。INVITE に対する最終レスポンス(Final response)を受け取るまでにかかる長い時間のために、INVITE トランザクションの信頼性メカニズムは、(OPTIONS のような)他のリクエストのものとは異なる。最終レスポンス(Final response)を受け取るとすぐに、UAC はそれが受け取るすべての最終レスポンス(Final response)に対して ACK を送る必要がある。この ACK を送るための手順は、レスポンスの種類に依存する。300 から 699 までの最終レスポンス(Final response)では、ACK 処理はトランザクションレイヤで行われ、ひとつのルールセット(17 節参照)に従う。2xx レスポンスでは、ACK は UAC コアで生成される。

INVITE に対する 2xx レスポンスはセッションを確立し、また、INVITE を発行した UA と 2xx レスポンスを生成した UA との間にダイアログを生成する。したがって、(INVITE がフォークされたために)異なる複数のリモート UA から複数の 2xx レスポンスを 受け取るとき、それぞれの 2xx は異なるダイアログを確立する。これらのすべてのダイアログは同一の呼の一部である。

本節では、INVITE を使用したセッションの確立の詳細について述べる。INVITE をサポートする UA は、ACK、CANCEL、および BYE もサポートしなくてはならない[MUST]。

13.2 UAC **の処理** (UAC Processing)

13.2.1 **最初の** INVITE **の生成** (Creating the Initial INVITE)

最初の INVITE はダイアログ外のリクエストに相当するので、その構築は 8.1.1 節の手順に従う。INVITE の特別な場合には付加的な処理が要求される。

Allow ヘッダフィールド(20.5 節)が INVITE 中に存在するべきである[SHOULD]。それは、INVITE を送る UA 上でダイアログの継続中にどのメソッドを呼び出すことができるかを示す。たとえば、ダイアログ内で INFO

リクエスト(参考文献[34])を受け取ることができるUAは、INFOメソッドをリストするAllowヘッダフィールドを含むべきである[SHOULD]。

Supportedヘッダフィールド(20.37節)がINVITE中に存在するべきである[SHOULD]。それはUACが理解できるすべての拡張を列挙する。

Acceptヘッダフィールド(20.1節)がINVITE中に存在してもよい[MAY]。それは、UAが、それが受け取るレスポンスと、INVITEで確立されたダイアログ内でそれに送られるそれ以降のすべてのリクエストにおいて、どのContent-Typeを受け入れ可能かを示す。Acceptヘッダはさまざまなセッション記述フォーマットのサポートを示すために特に有用である。

UACは、招待の有効性を制限するためにExpiresヘッダフィールド(22.19節)を追加してもよい[MAY]。Expiresヘッダフィールドで示された時間に達し、INVITEに対する最終回答を受け取っていなかった場合、9節にあるように、UACコアはINVITEに対してCANCELを生成するべきである[SHOULD]。

UACは、なかでもSubjectヘッダフィールド(20.36節)、Organizationヘッダフィールド(20.25節)、およびUser-Agentヘッダフィールド(20.41節)を追加することが有用だとしてもよい[MAY]。それらはすべてINVITEに関連する情報を含む。

UACはINVITEにメッセージボディを追加することを選択してもよい[MAY]。8.1.1.10節では、メッセージボディを記述するために必要になるヘッダフィールド(なかでもContent-Type)をどのように構築するかについて述べている。

セッション記述を含むメッセージボディのための特別なルールがある - セッション記述に対応するContent-Dispositionは「session」である。SIPでは、一つのUAが、それが提案するセッションの説明を含むオファー(offer)と呼ばれるセッション記述を送る、オファー/アンサーモデルを使用する。オファーは、希望するコミュニケーション手段(オーディオ、ビデオ、ゲーム)、それらの手段のパラメータ(たとえばCODECタイプ)、およびアンサー側(answerer)からメディアを受け取るためのアドレスを示す。相手UAは、どのコミュニケーション手段が受け入れられるか、それらの手段に適用するパラメータ、およびオファー側(Offerer)からメディアを受け取るためのアドレスを示す、アンサーと呼ばれる別のセッション記述で応答する。SIP INVITEが結果的に複数のダイアログになった場合、それぞれが別個のオファー/アンサーのやり取りがあるように、オファー/アンサーのやりとりはダイアログのコンテキスト内にある。オファー/アンサーモデルは、オファーとアンサーをいつ行うことができるかについての制限を定義する(例えば、オファーを処理している間は新たなオファーは作成できない)。このことは、オファーとアンサーがSIPメッセージのどこに現れることができるかについての制限を課すことになる。この仕様では、オファーとアンサーはINVITEリクエストとそれに対するレスポンス、およびACKにだけ現れることができる。オファーとアンサーの用法は更に制限される。最初のINVITEに対するルールは以下のようである。

- 最初のオファーは、INVITE中にあるか(そこにない場合には)UASからUACに返される最初の信頼性のある非失敗メッセージ中になくてはならない[MUST]。この仕様では、それは2xx最終レスポンス(Final response)である。

- 最初のオファーが INVITE 中にある場合、アンサーは、UAS から UAC に返されるその INVITE に関連する信頼性のある非失敗メッセージ中になくてはならない[MUST]。この仕様では、それは、その INVITE に対する最終 2xx レスポンスのみである。それとまったく同じアンサーを、アンサーが送られる前に送られるいかなる暫定レスポンス(provisional response)中にも置いてもよい[MAY]。UAC は、それが受け取る最初のセッション記述をアンサーとして扱わなくてはならず[MUST]、最初の INVITE に対するそれ以降のレスポンス中のいかなるセッション記述も無視しなくてはならない[MUST]。

- 最初のオファーが、UAS から UAC に返される最初の信頼性のある非失敗メッセージ中にある場合、アンサーはそのメッセージに対する承認 (acknowledgement)中になくてはならない[MUST](この仕様では、2xx レスポンスに対する ACK)。

- 最初のオファーに対するアンサーを送ったか受け取った後に、UAC は、そのメソッドに対して規定されたルールに基づくリクエスト中にそれに続く次のオファーを生成してもよい[MAY]。しかし、それができるのは、それ以前のすべてのオファーに対するアンサーを送信していて（※）、かつ、アンサーを受け取っていないいかなるオファーも送っていない場合だけである。

  [※訳注　：原文では"received"となっているが、"sent"の誤記と考えられる。本件については、SIP WG においてもバグと認識されている。]

- UAS は最初のオファーに対するアンサーをいったん送ったか受け取ったら、最初の INVITE に対するいかなるレスポンス中でも、それに続く次のオファーを生成してはいけない[MUST NOT]。これは、この仕様だけに基づく UAS が最初のトランザクションが完了するまで次のオファーを決して生成できないことを意味する。

具体的には、この仕様のみに準拠する UA について、上記のルールは 2 つのやりとり(exchanges)を規定する。オファーが INVITE 中にありアンサーが 2xx 中(おそらくは同じ値で 1xx にも)にあることを、あるいはオファーが 2xx 中にありアンサーが ACK 中にあることをである。INVITE をサポートするすべてのユーザエージェントは、これら 2 つのやりとりをサポートしなくてはならない[MUST]。

セッション記述プロトコル(SDP)(RFC2327 参考文献[1])は、セッションを記述する手段としてすべてのユーザエージェントがサポートしなくてはならず[MUST]、オファーとアンサーを構築するためのそれの用法は、参考文献[13]で定義されている手順にしたがわなくてはならない[MUST]。

上で述べられたオファー/アンサーモデルの制限事項は、Content-Disposition ヘッダフィールド値が「session」であるボディーにのみ適用される。したがって、INVITE と ACK 双方がボディメッセージを含むことが可能である(たとえば、INVITE が写真(Content-Disposition: render)を運び、ACK がセッション記述を運ぶ(Content-Disposition: session))。

Content-Disposition ヘッダフィールドがない場合、Content-Type が「application/sdp」のボディは disposition が「session」であることを意味する。一方、その他の Content-Type は「render」を意味する。

いったん INVITE が生成されると、UAC は、ダイアログ外でリクエストを送るために定義されている手順(8

節)にしたがう。これは、結果として、そのリクエストを送りその UAC にレスポンスを配信する、クライアントトランザクションを構築することになる。

13.2.2 INVITE に対するレスポンスの処理 (Processing INVITE Responses)

いったん INVITE が INVITE クライアントトランザクションに渡されると、UAC はその INVITE に対するレスポンスを待つ。INVITE クライアントトランザクションがレスポンスではなくタイムアウトを返却する場合、TU はあたかも 408(Request Timeout)レスポンスを受け取ったかのように動作する(8.1.3 節で述べられているように)。

13.2.2.1 1xx レスポンス (1xx Responses)

一つ以上の最終レスポンス(Final response)を受け取る前に、ゼロ個、1 個、あるいは複数個の暫定レスポンス(provisional response)が到着するかもしれない。INVITE リクエストに対する暫定レスポンス(provisional response)は「early ダイアログ」を生成することができる。暫定レスポンス(provisional response)が To フィールドに tag を持ち、そのレスポンスのダイアログ ID が既存のダイアログにマッチしない場合、12.1.2 節で定義されている手順を用いてダイアログが構築される。

early ダイアログは、最初の INVITE トランザクションが完了する前に、UAC がダイアログ内で相手にリクエストを送る必要がある場合にのみ必要とされる。暫定レスポンス(provisional response)中に存在するヘッダフィールドは、ダイアログが early ステートにある限り適用できる(たとえば、暫定レスポンス(provisional response)中の Allow ヘッダフィールドは、ダイアログが early ステートにある間だけ使用できるメソッドを含む)。

13.2.2.2 3xx レスポンス (3xx Responses)

3xx レスポンスは、着信者に到達できるかもしれない新しいアドレスを提供する一つ以上の Contact ヘッダフィールド値を含むことができる。3xx レスポンスのステータスコード(21.3 節参照)次第で、UAC はその新しいアドレスを試すことを選択してもよい[MAY]。

13.2.2.3 4xx、5xx、および 6xx レスポンス (4xx, 5xx and 6xx Responses)

INVITE に対して、一つの非 2xx 最終レスポンス(Final response)を受け取るかもしれない。4xx レスポンス、5xx レスポンス、および 6xx レスポンスは、エラーについての付加的な情報を得ることができるロケーションを示す Contact ヘッダフィールド値を含むことができる。それ以降の最終レスポンス(Final response)(エラー状態のときだけ到着する)は、無視されなくてはならない[MUST]。

すべての early ダイアログは、非 2xx 最終レスポンス(Final response)の受信時に終了するとみなされる。

非 2xx 最終レスポンス(Final response)を受け取った後、UAC コアは INVITE トランザクションが完了したとみなす。INVITE クライアントトランザクションは、そのレスポンスに対する ACK の生成をハンドリングする(17 節参照)。

13.2.2.4 2xx レスポンス (2xx Responses)

フォークするプロキシがあるため、一つの INVITE リクエストに対して複数の 2xx レスポンスが UAC に到着するかもしれない。各レスポンスは To ヘッダフィールドの tag パラメータによって区別され、それぞれ別個のダイアログ識別子を持った別個のダイアログに相当する。

2xx レスポンス中のダイアログ識別子が既存のダイアログのダイアログ識別子にマッチする場合、そのダイアログは「confirmed」ステートに移行しなくてはならず[MUST]、12.2.1.2 節の手順を用いて、2xx レスポンスに基づいてそのダイアログのルートセットを再計算しなくてはならない[MUST]。さもなければ、12.1.2 節の手順を用いて、「confirmed」ステートの新しいダイアログが構築されなくてはならない[MUST]。

再計算される状態の要素はルートセットだけであるということに注意すること。たとえばダイアログ内で送られる最も大きいシーケンス番号(リモートとローカルのもの)といった状態の他の要素は、再計算されない。ルートセットのみが下位互換性のために再計算される。RFC2543 では、Record-Route ヘッダフィールドのミラーリングを 2xx にのみ義務付けており、1xx には義務付けていなかった。しかしながら、ダイアログ内リクエスト(mid-dialog request)が、たとえばシーケンス番号を修正して、early ダイアログ内で送られたのかもしれないので、ダイアログの状態全体をアップデートすることはできない。

UAC コアは、トランザクションレイヤから受け取った各 2xx に対して ACK リクエストを生成しなくてはならない[MUST]。ACK のヘッダフィールドは、CSeq と認証に関するヘッダフィールドを除き、ダイアログ内で送られるすべてのリクエストと同じ方法で構築される(12 節参照)。CSeq ヘッダフィールドのシーケンス番号は、同意される(Acknowledged)INVITE のものと同じでなくてはならない[MUST]が、CSeq のメソッドは ACK でなくてはならない[MUST]。ACK は INVITE と同じ信用証明書を含まなくてはならない[MUST]。2xx が(上記のルールに基づく)オファーを含む場合、ACK はボディでアンサーを伝えなくてはならない[MUST]。2xx レスポンス中のオファーが受け入れ不可能な場合、UAC コアは ACK 中に有効なアンサーを生成してそれから直ちに BYE を送らなくてはならない[MUST]。

いったん ACK が構築されると、送信先アドレス、ポート、およびトランスポートを確定するために参考文献[4]の手順が使用される。しかしながら、リクエストは送信のために、クライアントトランザクションではなく、トランスポートレイヤに直接渡される。これは、トランザクションレイヤではなく UAC コアが ACK の再送をハンドリングするためである。ACK のトリガーとなった 2xx 最終レスポンス(Final response)の再送が到着するたびに、ACK はクライアントトランスポートに渡されなくてはならない[MUST]。

UAC コアは、最初の 2xx レスポンスを受信してから 64*T1 秒後に、INVITE トランザクションが完了するとみなす。この時点で、established ダイアログに移行していないすべての early ダイアログは終了される。いったん UAC コアによって INVITE トランザクションが完了したとみなされると、これ以上の 2xx レスポンスが到着するとは期待されない。

INVITE に対するどのような 2xx レスポンスに同意した(acknowledge)後でも、UAC がそのダイアログの継続を望まない場合は、UAC は 15 節で述べられているように BYE リクエストを送ることで、そのダイアログを終了しなくてはならない[MUST]。

13.3 UAS の処理 (UAS Processing)

13.3.1 INVITE の処理 (Processing of the INVITE)

UAS コアはトランザクションレイヤから INVITE リクエストを受け取る。UAS は最初に 8.2 節のリクエスト処理手順を実行する。この処理はダイアログの内部および外部両方のリクエストに適用されるものである。

これらの処理状態がレスポンスを生成せずに完了されると仮定すると、UAS コアは付加的な処理手順を実

行する。

1. リクエストが Expires ヘッダフィールドを含む INVITE の場合、UAS コアはこのヘッダフィールド値で示される秒数のタイマーを設定する。タイマーが切れるとき、招待は期限切れになったとみなされる。UAS が最終レスポンス(Final response)を生成する前に招待が期限切れになる場合には、487(Request Terminated)レスポンスが生成されるべきである [SHOULD]。

2. クエストがダイアログ内リクエスト(mid-dialog request)である場合、12.2.2 節で述べられているメソッドに依存しない処理が最初に適用される。それはセッションの修正も行うかもしれない。14 節で詳細を述べる。

3. リクエストが To ヘッダフィールドに tag を持つが、ダイアログ識別子が既存のどのダイアログにもマッチしない場合、UAS はクラッシュしてリスタートしたのかもしれないし、あるいは別の(おそらく故障した)UAS へのリクエストを受け取ったのかもしれない。このような状況下で堅牢性を実現するためのガイドラインを 12.2.2 節で提供する。

これ以降の処理は、INVITE がダイアログ外のもので、したがって新規セッションを確立するためのものであると仮定している。

INVITE はセッション記述を含むかもしれない。この場合、UAS はそのセッションのためのオファーを提示されている。たとえ INVITE がダイアログ外のものであっても、ユーザはすでにそのセッションの参加者である可能性がある。これは、ユーザが他の複数の参加者から同一のマルチキャストカンファレンスに招待されたときに起こり得る。希望するなら、UAS はこの重複を検知するために、セッション記述内で識別子を使用してもよい[MAY]。たとえば、SDP は origin(o)フィールドにセッション ID とバージョン番号を含む。ユーザがすでにそのセッションのメンバーであり、セッション記述に含まれているセッションパラメータが変化していなければ、UAS はその INVITE を黙って受け入れ(silently accept)てもよい[MAY](すなわち、ユーザに注意を促さずに 2xx レスポンスを送る)。

INVITE がセッション記述を含まない場合、UAS はセッションに参加することを依頼されており、UAC は UAS がセッションのオファーを提供することを頼んでいる。UAS は、UAC に返す最初の信頼性のある非失敗メッセージでオファーを提供しなくてはならない[MUST]。この仕様では、それは INVITE に対する 2xx レスポンスである。

UAS は進捗状態を示したり、招待を受け入れたり、招待をリダイレクトしたり、あるいは招待を拒否することができる。これらすべての場合において、UAS は 8.2.6 節で述べられている手順を用いてレスポンスを作成する。

13.3.1.1 **進捗状態** (Progress)

UAS が直ちに招待に答えられない場合、UAC に対してある種の進捗状態を示すことを選ぶことができる(たとえば、電話が鳴っているという合図)。これは 101～199 の暫定レスポンス(provisional response)で実現される。これらの暫定レスポンス(provisional response)は early ダイアログを確立し、それゆえ 8.2.6 節の手順に加えて 12.1.1 節の手順にしたがう。UAS は望むだけの数の暫定レスポンス(provisional response)を送ってもよい[MAY]。それらは各々、同一のダイアログ ID を示さなくてはならない[MUST]。しかしながら、

これらは信頼性を持って送られることはない。

UAS が INVITE に答えるために長い時間を希望する場合、プロキシがトランザクションをキャンセルすることを防ぐために、「時間の延長(extension)」を依頼する必要がある。プロキシは、トランザクション内のレスポンスとレスポンスの間に 3 分の間隔があるときに、トランザクションをキャンセルするオプションを持つ。キャンセルを防ぐために、UAS は非 100 暫定レスポンス(provisional response)を(暫定レスポンス(provisional response)のロストの可能性に対処するため)毎分、送らなくてはならない[MUST]。

ユーザが保留されたとき、または呼び出しに答えずにコミュニケーションが行われることを可能にする PSTN システムと相互に動作するときに、INVITE トランザクションは延長された期間中継続できる。後者は、音声自動レスポンスシステム(IVR)において一般的である。

13.3.1.2 INVITE がリダイレクトされる (The INVITE is Redirected)

UAS が呼のリダイレクトを決定した場合、3xx レスポンスが送られる。300(Multiple Choices)レスポンス、301(Moved Permanently)レスポンス、あるいは 302(Moved Temporarily)レスポンスは、試行されるべき一つ以上の新しいアドレスの URI を含む Contact ヘッダフィールドを含むべきである[SHOULD]。そのレスポンスは、再送を処理する、INVITE サーバトランザクションに渡される。

13.3.1.3 INVITE が拒否される (The INVITE is Rejected)

着信者がそのエンドシステムでもう一つの呼を受け取れないあるいは受け取りたくないときがある。そのような場合、486(Busy Here)が返送されるべきである[SHOULD]。他のどのエンドシステムでもこの呼を受け入れられないことを UAS が知っている場合、かわりに 600(Busy Everywhere)レスポンスが送られるべきである[SHOULD]。しかしながら、一般的に UAS がこのことを知ることができる可能性は低く、そのためこのレスポンスは通常使用されない。レスポンスは、再送を処理する、INVITE サーバトランザクションに渡される。

INVITE に含まれているオファーを拒否する UAS は、488(Not Acceptable Here)レスポンスを返送するべきである[SHOULD]。そのようなレスポンスは、オファーがなぜ拒否されたのかを説明する Warning ヘッダフィールド値を含むべきである[SHOULD]。

13.3.1.4 INVITE が受け入れられる (The INVITE is Accepted)

UAS コアは 2xx レスポンスを生成する。このレスポンスはダイアログを確立するので 8.2.6 節の手順に加えて 12.1.1 節の手順にしたがう。

INVITE に対する 2xx レスポンスは、Allow ヘッダフィールドと Supported ヘッダフィールドを含むべきであり[SHOULD]、Accept ヘッダフィールドを含んでもよい[MAY]。これらのヘッダフィールドを含むことで、調査することなしに、呼が継続する間、UAC が UAS のサポートしている機能と拡張を確定することが可能になる。

INVITE リクエストがオファーを含んでいて、UAS がまだアンサーを送っていない場合、2xx はアンサーを含まなくてはならない[MUST]。INVITE がオファーを含まなかった場合、UAS がまだオファーを送っていなかったら、2xx はオファーを含まなくてはならない[MUST]。

レスポンスが構築されるとすぐに、それは INVITE サーバトランザクションに渡される。しかしながら、

INVITE サーバトランザクションは、この最終レスポンス(Final response)を受け取ってそれをトランスポートに渡すと同時に破棄されるということに注意すること。したがって、ACK が到着するまでの間定期的にそのレスポンスを直接トランスポートに渡す必要がある。2xx レスポンスは、T1 秒で始まり、T2 秒になるまで再送のたびに 2 倍になる時間間隔で、トランスポートに渡される(T1 と T2 は 17 節で定義される)。レスポンスの再送は、そのレスポンスに対する ACK リクエストを受け取るときに停止する。これは、レスポンスを送るためにどのようなトランスポートプロトコルを使用するかによらない。

2xx はエンド・トゥ・エンドで再送されるので、UAS と UAC の間に UDP のホップがあるかもしれない。これらのホップをまたぐ確実な配送を保証するために、UAS でのトランスポートに信頼性があったとしても、レスポンスは定期的に再送される。

サーバが ACK を受け取ることなく 2xx レスポンスを 64*T1 秒の間再送する場合、ダイアログはコンファームされ(confirmed)るが、セッションは終了されるべきである[SHOULD]。これは、15 節で述べられているように BYE で実現される。

１４．既存セッションの変更 (Modifying an Existing Session)

成功した INVITE リクエスト(13 節参照)は、オファー/アンサーモデルを使用して 2 つのユーザエージェント間にダイアログとセッションを確立する。12 節で、ターゲットリフレッシュリクエストを使ってどのように既存のダイアログを修正するかを説明する(たとえば、ダイアログのリモートターゲットの URI を変更する)。本節では実際のセッションを修正する方法を述べる。この修正は、アドレスやポートの変更、メディアストリームの追加、メディアストリームの削除、などを伴うことができる。これは、セッションを確立したのと同じダイアログ内で新規 INVITE リクエストを送ることによって実現できる。既存のダイアログ内で送られた INVITE リクエストは re-INVITE として知られている。

単一の re-INVITE は、ダイアログとセッションのパラメータを同時に変更できるということに注意すること。

発信者と着信者のいずれも既存のセッションを変更できる。

メディアの失敗を検知したときの UA の動作はローカルポリシー次第である。しかしながら、ネットワークトラフィックが混雑しているときにトラフィックをあふれさせることを避けるため、re-INVITE や BYE を自動生成することは推奨されない[NOT RECOMMENDED]。いかなる場合でも、これらのメッセージが自動で送られる場合は、ランダムな間隔のあとで送られるべきである[SHOULD]。

上記のパラグラフは自動的に生成される re-INVITE と BYE について述べている、ということに注意すること。メディアの失敗時にユーザがハングアップする場合、UA は BYE リクエストを通常どおりに送る。

14.1 UAC の動作 (UAC Behavior)

INVITE 中のセッション記述に適用されるのと同じオファー/アンサーモデル(13.2.1 節)が re-INVITE にも適用される。結果として、たとえば、メディアストリームの追加を望む UAC は、そのメディアストリームを含む新規オファーを生成し、それを INVITE リクエストで相手に送る。変更部分だけではなく、完全なセッション記述が送られることに注意することが重要である。これは、様々なエレメントにおけるステートレスなセッション処理、およびフェールオーバーとリカバリー能力をサポートする。もちろん、UAC はセッショ

ン記述なしで re-INVITE を送ってもよい[MAY]。この場合、re-INVITE に対する信頼性のある最初の非失敗レスポンスがオファーを含む(この仕様では、これは 2xx レスポンスである)。

セッション記述のフォーマットがバージョン番号を表現する能力を持っている場合、オファー側(offerer)はセッション記述のバージョンが変更されたことを示すべきである[SHOULD]。

re-INVITE の To、From、Call-ID、CSeq、および Request-URI は、12 節で述べられている、既存ダイアログ内での通常のリクエストのためのものと同じルールにしたがって設定される。

UAS は通常 re-INVITE の受信時にユーザに知らせないので、UAC は Alert-Info ヘッダフィールドまたは「alert」という値の Content-Disposition を持つボディを追加しないことを選択してもよい[MAY]。

フォークすることができる INVITE とは異なり、re-INVITE は決してフォークしない。そのため、常に一つの最終レスポンス(Final response)を生成する。re-INVITE が決してフォークしないことの理由は、ダイアログを確立した相手 UA インスタンスとして Request-URI がターゲットを識別するからである(そのユーザの Address-of-Record(AOR)を識別するのではなく)。

どちらかの方向で他の INVITE トランザクションが進行している間、UAC はダイアログ内で新規 INVITE トランザクションを開始してはいけない[MUST NOT]ということに注意すること。

1. 進行中の INVITE クライアントトランザクションがある場合、TU は新規 INVITE を開始する前に、そのトランザクションが Completed または Terminated ステートになるまで待たなくてはならない[MUST]。

2. 進行中の INVITE サーバトランザクションがある場合、TU は新規 INVITE を開始する前に、そのトランザクションが confirmed または Terminated ステートになるまで待たなくてはならない[MUST]。

しかしながら、UA は INVITE トランザクションが進行している間に通常のトランザクションを開始してもよい[MAY]。また、UA は通常のトランザクションが進行している間に INVITE トランザクションを開始してもよい[MAY]。

UA が re-INVITE に対する非 2xx 最終レスポンス(Final response)を受け取る場合、あたかも re-INVITE が発行されなかったかのように、セッションパラメータは変更されずにそのままでなくてはならない[MUST]。12.2.1.2 節で述べられているように、非 2xx レスポンスが、481(Call/Transaction Does Not Exist)、408(Request Timeout)、または re-INVITE に対するレスポンスが何もない(すなわち、INVITE クライアントトランザクションによってタイムアウトが返却された)場合、UAC はダイアログを終了するということに注意すること。

UAC が re-INVITE に対する 491 レスポンスを受け取る場合、UAC は以下のようにして選ばれたタイマーT を開始するべきである[SHOULD]。

1. UAC がそのダイアログ ID の Call-ID の所有者である場合(つまり、その UAC がその値を生成した場合)、T は 2.1 秒から 4 秒の間で 10 ミリ秒単位でランダムに選択された値を持つ。

2. UAC がそのダイアログ ID の Call-ID の所有者でない場合、T は 0 秒から 2 秒の間で 10 ミリ秒単位でランダムに選択された値を持つ。

タイマーが切れるとき、そのセッション修正を行うことをまだ望むなら、UAC は re-INVITE をもう一度試みるべきである[SHOULD]。たとえば、呼が BYE で既に切断されていた場合、re-INVITE は行われないだろう。

re-INVITE を送信することと、re-INVITE への 2xx レスポンスに対する ACK の生成のためのルールは、最初の INVITE のためのもの(13.2.1 節)と同じである。

14.2 UAS の動作 (UAS Behavior)

13.3.1 節では、こちらにやってくる re-INVITE をこちらにやってくる最初の INVITE と区別するための手順、および既存のダイアログに対する re-INVITE をハンドリングするための手順を述べている。

2 番めの INVITE を、それよりも小さい CSeq シーケンス番号を持つ最初の INVITE に対する最終レスポンス(Final response)を送る前に、同じダイアログ上で受け取る UAS は、2 番めの INVITE に対して 500(Server Internal Error)レスポンスを返送しなくてはならず[MUST]、0 秒から 10 秒の間でランダムに選ばれた値を持つ Retry-After ヘッダフィールドを含まなくてはならない[MUST]。

あるダイアログ上で送った INVITE が進行している間に、そのダイアログ上でもう一つの INVITE を受け取る UAS は、受け取ったその INVITE に対して 491(Request Pending)レスポンスを返送しなくてはならない[MUST]。

UA が既存のダイアログに対する re-INVITE を受け取る場合、セッション記述中のバージョン識別子または(バージョン識別子がない場合は)セッション記述のコンテンツを、それが変更されていないかどうか確認するために、チェックしなくてはならない[MUST]。セッション記述が変更されていた場合、UAS は、おそらくはユーザに確認を取った後で、セッションパラメータをしかるべく調整しなくてはならない[MUST]。

> セッション記述のバージョン番号付けは、新しくカンファレンスにやってきた者の能力を受け入れるため、メディアを追加/削除するため、あるいはユニキャストカンファレンスからマルチキャストカンファレンス に変更するために使用できる。

新しいセッション記述が受け入れられない場合、UAS は re-INVITE に対して 488(Not Acceptable Here)レスポンスを返送することでそれを拒否できる。このレスポンスは Warning ヘッダフィールドを含むべきである[SHOULD]。

UAS が 2xx レスポンスを生成したのにいつまでも ACK を受け取らない場合、UAS はダイアログを終了するために BYE を生成するべきである[SHOULD]。

UAS は re-INVITE に対して 180(Ringing)レスポンスを生成しないことを選択してもよい[MAY]。これは、UAC が通常この情報をユーザに与えないためである。同じ理由から、UAS は re-INVITE に対するレスポンスで、Alert-Info ヘッダフィールドや”alert”という値の Content-Disposition を持つボディを使わないことを選択してもよい[MAY]。

(INVITE がオファーを含まなかったため)2xx でオファーを提供する UAS は、SDP の場合について参考文献[13]で述べられているように、既存のセッションをアップデートするオファーを送る際の制約にしたがって、あたかも UAS が新たに電話をかけているかのようにオファーを構築するべきである[SHOULD]。具体的には、それはその UA がサポートすることを望むメディアフォーマットとメディアタイプをできるだけ含むべきである[SHOULD]ことを意味する。UAS は、セッション記述が以前のセッション記述と、相手側のサポートを必要とするメディアフォーマット、トランスポート、または他のパラメータにおいて重複することを保証しなくてはならない[MUST]。これは、相手がセッション記述を拒否する必要性を避けるためである。しかしながら、それが UAC にとって受け入れられないものである場合、UAC は有効なセッション記述を持つアンサーを生成するべきであり[SHOULD]、次いでそのセッションを終了するために BYE を送る。

１５．セッションの終了 (Terminating a Session)

本節では、SIP で確立されたセッションを終了するための手順について述べる。セッションの状態とダイアログの状態は非常に密接に関連している。セッションが INVITE で開始されるとき、個々の UAS からのそれぞれの 1xx または 2xx レスポンスはダイアログを生成し、そのレスポンスがオファー/アンサー交換を完了する場合、それはセッションも生成する。結果として、それぞれのセッションは(それを生成する結果を招いた)一つのダイアログと「関連付け」られる。最初の INVITE が非 2xx 最終レスポンス(Final response)を生成する場合、それはリクエストに対するレスポンスで生成された(もしあれば)すべてのセッションと(もしあれば)すべてのダイアログを終了する。トランザクションを完了することで、非 2xx 最終レスポンス(Final response)は更なるセッションが INVITE の結果として生成されることを防ぐ。特定のセッションやまだ確立していないセッションを終了するために BYE リクエストが使用される。この場合の特定のセッションとは、ダイアログの向こう側に相手 UA がいるセッションのことである。ダイアログ上で BYE を受け取るときは、そのダイアログに関連付けられたすべてのセッションを終了するべきである[SHOULD]。UA はダイアログ外で BYE を送ってはいけない[MUST NOT]。発信者の UA は confirmed ダイアログあるいは early ダイアログのいずれに対しても BYE を送ってもよい[MAY]。着信者の UA は confirmed ダイアログ上で BYE を送ってもよい[MAY]が、early ダイアログ上で BYE を送ってはいけない[MUST NOT]。

しかしながら、着信者の UA は、それが送った 2xx レスポンスに対する ACK を受け取るまで、あるいはサーバトランザクションがタイムアウトするまでは confirmed ダイアログ上で BYE を送ってはいけない[MUST NOT]。どの SIP 拡張もダイアログに関連付けられた他のアプリケーションレイヤステートを定義していなかった場合、BYE はダイアログも終了する。

INVITE に対する非 2xx 最終レスポンス(Final response)がダイアログおよびセッションに与える強い影響は CANCEL の使用を魅力のあるものにする。CANCEL は INVITE に対する非 2xx レスポンス(特に、487)を強いる。したがって、UAC が呼び出しの試み(call attempt)を完全にやめたいと望む場合、UAC は CANCEL を送ることができる。INVITE がそれに対する 2xx 最終レスポンス(Final response)(一つまたは複数)を生じる結果になる場合は、CANCEL が進行している間に UAS が招待を受け入れたことを意味する。UAC は 2xx レスポンスで確立されたセッションを継続してもよい[MAY]し、またはそれらを BYE で終了してもよい[MAY]。

「電話を切る(haging up)」という考えは、SIP ではしっかりと定義されていない。たとえ一般的な考えであるとしても、それはある特定のユーザインターフェースに固有のものである。通常、ユーザが電話を切るときには、セッションを確立する試みを終了すること、およびすでに生成されたすべてのセッションを終了することを望むことを示す。発信者の UA にとってこれは、最初の INVITE が最終レスポンス

(Final response)を生成していない場合には CANCEL リクエストを、そして最終レスポンス(Final response)後のすべての confirmed ダイアログに対しては BYE を意味する。着信者の UA にとっては、これは通常 BYE を意味する。つまり、推定されるように、ユーザが電話に出るとき 2xx が生成されるので、電話を切ることは ACK を受信した後に BYE を生成することになる。これは、ユーザが ACK の受信前に電話を切ることができないという意味ではなく、ユーザの電話の中のソフトウェアが適切にクリーンアップを行うために短時間だけ状態を保持する必要があることを単に意味する。ユーザが電話に出る前にそれを拒否することを特定の UI が可能にする場合、403(Forbidden)がそれを表現するための良い方法である。上記のルールにより、BYE を送ることはできない。

15.1 BYE リクエストによるセッションの終了 (Terminating a Session with a BYE Request)

15.1.1 UAC の動作 (UAC Behavior)

BYE リクエストはダイアログ内の他のいかなるリクエストとも同様に、12 節で述べられているように構築される。

BYE が構築されるとすぐに、UAC コアは新規の非 INVITE クライアントトランザクションを生成し、それに BYE リクエストを渡す。UAC は BYE リクエストがクライアントトランザクションに渡されるとすぐにセッションが終了した(したがって、メディアの送信または受信を停止する)とみなさなくてはならない[MUST]。BYE に対するレスポンスが 481(Call/Transaction Does Not Exist)または 408(Request Timeout)の場合、あるいは BYE に対するレスポンスを何も受け取らない場合(すなわち、クライアントトランザクションからタイムアウトが返却された)、UAC はセッションとダイアログが終了したとみなさなくてはならない[MUST]。

15.1.2 UAS の動作 (UAS Behavior)

UAS は最初に 8.2 節で述べられている一般的な UAS の処理にしたがって BYE リクエストを処理する。BYE リクエストを受け取る UAS コアは、それが既存のダイアログにマッチするかどうかを確認する。BYE が既存のダイアログにマッチしない場合、UAS コアは 481(Call/Transaction Does Not Exist)レスポンスを生成してそれをサーバトランザクションに渡すべきである[SHOULD]。

このルールは、UAC が送った tag を持たない BYE は拒否されることを示す。これは tag なしの BYE を許可していた RFC2543 からの変更である。

既存のダイアログに対して BYE リクエストを受け取る UAS コアは、そのリクエストを処理するために 12.2.2 節の手順に従わなくてはならない[MUST]。処理が終了したらすぐに、UAS はセッションを終了するべきである[SHOULD](それゆえ、メディアの送信と受信を中止する)。セッションの終了を選択できない唯一のケースは、たとえダイアログ内の他の参加者がセッションへの関わりを終了しても参加が可能な、マルチキャストセッションの場合である。セッションへの参加を終えるかどうかに関わらず、UAS コアは BYE に対して 2xx レスポンスを生成しなくてはならず[MUST]、送信するためにそれをサーバトランザクションに渡さなくてはならない[MUST]。

UAS は、そのダイアログに対して受け取っているいかなる保留中のリクエストに対しても依然としてレスポンスしなくてはならない[MUST]。それらの保留中のリクエストに対して 487(Request Terminated)レスポンスを生成することが推奨される[RECOMMENDED]。

１６．プロキシの動作 (Proxy Behavior)

16.1　概要 (Overview)

SIP プロキシは、SIP リクエストをユーザエージェントサーバに、SIP レスポンスをユーザエージェントクライアントにルーティングするエレメントである。リクエストは UAS にたどり着くまでにいくつかのプロキシをトラバースするかもしれない。各プロキシはルーティングの決定を行い、リクエストを次のエレメントにフォワードする前にそれを修正する。レスポンスは、リクエストがトラバースしたプロキシの組を逆にたどってルーティングされる。

SIP エレメントにとって、プロキシになるということは論理的な役割を果たすことである。リクエストが到着するとき、プロキシの役割を果たすことができるエレメントは、それ自身がそのリクエストにレスポンスする必要があるかどうかを最初に確定する。たとえば、リクエストが不正な形式になっているかもしれないし、そのエレメントがプロキシとして動作する前にクライアントからの信用証明書を必要とするかもしれない。エレメントは、あらゆる適切なエラーコードでレスポンスしてもよい[MAY]。リクエストに直接レスポンスするとき、エレメントは UAS の役割を演じており、8.2 節で述べられているように動作しなくてはならない[MUST]。

プロキシは、それぞれの新規リクエストに対して、ステートフルまたはステートレスのいずれかのモードで動作できる。ステートレスで動作するとき、プロキシは単にフォワードを行うエレメントとして動作する。それは各リクエストをダウンストリームに、そのリクエストに基づいてターゲットの決定とルーティングの決定を行うことで確定された一つのエレメントにフォワードする。それは受け取ったすべてのレスポンスを単にアップストリームにフォワードする。ステートレスプロキシは、メッセージをフォワードするとすぐにそれについての情報を破棄する。ステートフルプロキシは、送られてくる各リクエストと送られてくるリクエストを処理した結果として送るすべてのリクエストについての情報(具体的には、トランザクションステート)を憶えている。ステートフルプロキシはこの情報を、そのリクエストに関連する以降のメッセージの処理に影響を与えるために使用する。ステートフルプロキシは、リクエストをフォークしたり、複数のデスティネーションにルートすることを選択してもよい[MAY]。一つ以上のロケーションにフォワードされるすべてのリクエストはステートフルでハンドリングされなくてはならない[MUST]。

ある状況下では、プロキシは、トランザクションがステートフルにならずに(TCP のような)ステートフルトランスポートを利用してリクエストをフォワードしてもよい[MAY]。たとえば、プロキシは、一つの TCP コネクションからのリクエストを他のトランザクションにステートレスにフォワードしてもよい[MAY](そのリクエストが到着したのと同じコネクションを逆方向に、レスポンスをフォワードするのに十分な情報をメッセージ中に置きさえすれば)。異なるトランスポートタイプ(プロキシの TU は、そのどちらかで確実な配送が行われることを保証するための積極的な役割を果たさなければならない)の間でフォワードされるリクエストは、トランザクションがステートフルな状態でフォワードされなくてはならない[MUST]。

ステートフルプロキシは、ステートレスになることを妨げるようなことを最初に何もしていなければ(たとえば、フォークや 100 レスポンスの生成)、リクエストの処理中にいつでもステートレスオペレーションに移行してもよい[MAY]。そのような移行を行うとき、すべての状態は単純に破棄される。そのプロキシは CANCEL リクエストを開始するべきではない[SHOULD NOT]。

リクエストに対してステートレスまたはステートフルに動作するときに関わる処理の多くは同じである。これ以降のいくつかのサブ節は、ステートフルプロキシの視点から書かれている。最後のサブ節は、ステー

トレスプロキシが異なった動作をする点について述べている。

16.2 ステートフルプロキシ (Stateful Proxy)

ステートフルで動作するとき、プロキシは純粋にSIPトランザクション処理エンジンである。その動作は、ここでは、17節で定義されるサーバトランザクションおよびクライアントトランザクションの観点からモデル化する。ステートフルプロキシは、プロキシコアとして知られる上位レイヤのプロキシ処理コンポーネント(図3参照)によって一つ以上のクライアントトランザクションに関連付けられた、サーバトランザクションを持つ。送られてくるリクエストはサーバトランザクションによって処理される。サーバトランザクションからのリクエストはプロキシコアに渡される。プロキシコアは、一つ以上のネクストホップのロケーションを選択して、そのリクエストをどこにルーティングするか決定する。各ネクストホップのロケーションに対して送り出すリクエストは、それら自身に関連付けられたクライアントトランザクションによって処理される。プロキシコアはクライアントトランザクションからのレスポンスを収集し、それらをサーバトランザクションにレスポンスを送るために使用する。

ステートフルプロキシは、受け取った各新規リクエストに対する新規サーバトランザクションを生成する。リクエストのすべての再送はその後、17節で述べられているように、そのサーバトランザクションによってハンドリングされる。プロキシコアは、8.2.6節で述べられているように、そのサーバトランザクション上で即時暫定レスポンスを送ることに関しては UAS として動作しなくてはならない[MUST](たとえば 100 Trying)。したがって、ステートフルプロキシは非 INVITE リクエストに対して100(Trying)レスポンスを生成するべきではない[SHOULD NOT]。

これはプロキシの動作モデルであって、ソフトウェアの動作モデルではない。実装では、このモデルが定義する外部動作を再現するどのようなアプローチをとることもできる。

すべての新規リクエスト(未知のメソッドを持つものも含む)に対して、リクエストをプロキシしようとするエレメントは、必ず以下の事をしなくてはならない[MUST]。

1. リクエストの有効性を検証する(16.3節)
2. ルーティング情報の前処理を行う(16.4節)
3. リクエストのターゲット(一つまたは複数)を確定する(16.5節)

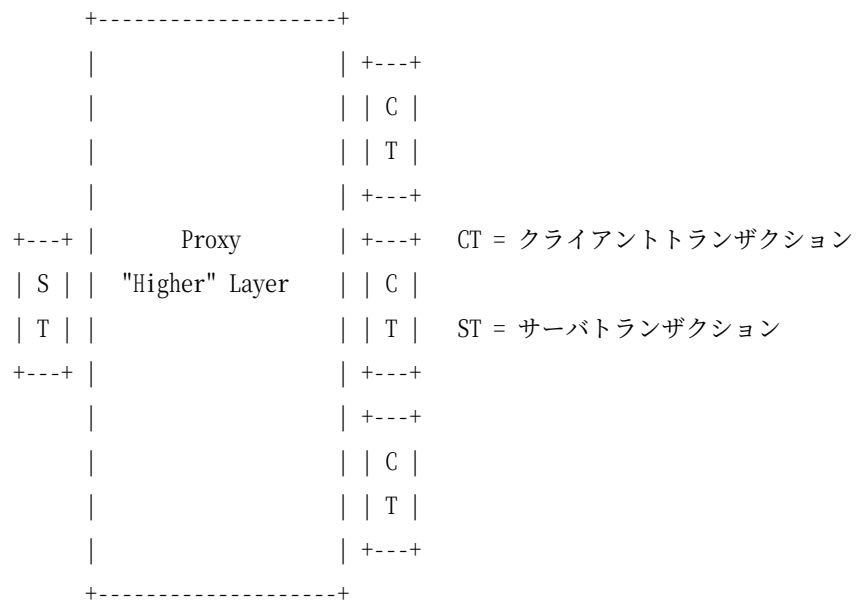

図 3: ステートフルプロキシのモデル

4. 各ターゲットにリクエストをフォワードする(16.6 節)

5. すべてのレスポンスを処理する(16.7 節)

16.3 **リクエストの有効性検証** (Request Validation)

エレメントはリクエストをプロキシする前に、メッセージの有効性を検証しなくてはならない[MUST]。有効なメッセージは以下のチェックをパスしなければならない。

1. 構文の合理性
2. URI スキーム
3. Max-Forwards
4. (任意の)ループ検知
5. Proxy-Require
6. Proxy-Authorization

これらのチェックのうちどれかが失敗する場合、エレメントはユーザエージェントサーバとして動作して(8.2 節参照)エラーコードでレスポンスしなくてはならない[MUST]。

プロキシはマージされたリクエストを検知する必要はないことに注意すること。そして、マージされたリクエストをエラー状態として扱ってはいけない[MUST NOT]ことにも注意すること。リクエストを受け取るエンドポイントが、8.2.2.2 節で述べられているようにマージを解決する。

1. 構文の合理性チェック

リクエストは、サーバトランザクションでハンドリングされるために、十分に well-formed でなくてはならない[MUST]。これらの「リクエストの有効性検証」ステップあるいは「リクエストの処理」セク

ションの結果として残った部分から成るコンポーネントは、well-formed でなくてはならない[MUST]。それ以外のすべてのコンポーネントは、well-formed かそうでないかに関わらず、無視されてメッセージがフォワードされるときには変更されずにそのままであるべきである[SHOULD]。たとえば、エレメントは不正な形式の Date ヘッダフィールドを理由にしてリクエストを拒否しない。同様に、プロキシはリクエストをフォワードする前に不正な形式の Date ヘッダフィールドを削除しない。

このプロトコルは拡張されることを考慮してデザインされている。 将来の拡張ではいつでも新しいメソッドやヘッダフィールドを定義できる。エレメントは、それが知らないメソッドやヘッダフィールドをリクエストが含むことを理由にして、そのリクエストをプロキシすることを拒否してはいけない[MUST NOT]。

2. URI スキームチェック

Request-URI がプロキシの理解できないスキームの URI を持っている場合、プロキシは 416(Unsupported URI Scheme)レスポンスでそのリクエストを拒否するべきである[SHOULD]。

3. Max-Forwards チェック

Max-Forwards ヘッダフィールド(20.22 節)は、SIP リクエストがトラバースできるエレメントの数を制限するために使用される。

リクエストが Max-Forwards ヘッダフィールドを含まない場合は、このチェックをパスする。

リクエストが、0 よりも大きいフィールド値を持つ Max-Forwards ヘッダフィールドを含む場合は、このチェックをパスする。

リクエストが、フィールド値ゼロ(0)を持つ Max-Forwards ヘッダフィールドを含む場合、エレメントはそのリクエストをフォワードしてはいけない[MUST NOT]。リクエストが OPTIONS のためのものだった場合、エレメントは最終受信者として動作して 11 節に基づいてレスポンスしてもよい[MAY]。そうでなければ、エレメントは 483(Too many hops)レスポンスを返送しなくてはならない[MUST]。

4. 任意のループ検知チェック

エレメントはリクエストをフォワードする前に、ループ転送をチェックしてもよい[MAY]。リクエストが、そのプロキシによってそれ以前のリクエストに入れられたのと同じ sent-by 値を持つ Via ヘッダフィールドを含む場合、そのリクエストは、以前、このエレメントによってフォワードされている。そのリクエストはループしているか、あるいはそのエレメントを通って正規にスパイラルしている。リクエストがループしているかどうか確定するために、エレメントは、16.6 節のステップ 8 に述べられている branch パラメータの計算をこのメッセージで実行して、それを Via ヘッダフィールド値で受け取ったパラメータと比較してもよい[MAY]。パラメータがマッチした場合、そのリクエストはループしている。異なっていた場合は、そのリクエストはスパイラルしており、処理は継続する。ループが検知された場合、エレメントは 482(Loop Detected)レスポンスを返送してもよい[MAY]。

5. Proxy-Require チェック

将来のこのプロトコルの拡張では、プロキシによる特別なハンドリングを要求する機能を導入するかもしれない。エンドポイントは、これらの機能を使用するリクエストに、プロキシがその機能を理解で

きなければリクエストを処理しないことを伝えるための、Proxy-Require ヘッダフィールドを含めるだろう。

リクエストが、このエレメントが理解できない一つ以上のオプションタグを持つ Proxy-Require ヘッダフィールド(20.29 節)を含む場合、そのエレメントは 420(Bad Extension)レスポンスを返送しなくてはならない[MUST]。そのレスポンスはエレメントが理解できなかったオプションタグをリストした Unsupported(20.40 節)ヘッダフィールドを含まなくてはならない[MUST]。

6. Proxy-Authorization チェック

エレメントがリクエストをフォワードする前に信用証明書を必要とする場合、そのリクエストは 22.3 節で述べられているように検査されなくてはならない[MUST]。22.3 節では、検査が失敗した場合にエレメントが何をしなければならないかも定義している。

16.4 ルート情報の前処理 (Route Information Preprocessing)

プロキシはリクエストの Request-URI を検査しなくてはならない[MUST]。リクエストの Request-URI が、以前にこのプロキシが Record-Route ヘッダフィールドに置いた値を含んでいる場合(16.6 節の項目 4 参照)、プロキシはリクエストの Request-URI を Route ヘッダフィールドの最後の値に置き換えて、Route ヘッダフィールドからその値を削除しなくてはならない[MUST]。それからプロキシは、この修正されたリクエストを受け取ったかのように処理を進めなくてはならない[MUST]。

これは、リクエストをプロキシ(エンドポイントかもしれない)に送るエレメントがストリクトルータであるときにのみ起こる。受信時のこの書き換えは、それらのエレメントとの下位互換を可能にするために必要である。これはまた、この仕様に従うエレメントがストリクトルーティングを行うプロキシを経ても Request-URI を維持することを可能にする(12.2.1.1 節参照)。

この要求は、プロキシが以前に Record-Route ヘッダフィールドに置いた URI を検知するために状態を保持することを、プロキシに義務付けるものではない。そうではなくてプロキシは、それらの URI がのちほど現れたときに、それらがそのプロキシが提供した値であることを認識するのに十分な情報だけをそれらの URI に置くことが必要である。

Request-URI が maddr パラメータを含む場合、プロキシはその値が、プロキシが責任を負うように設定されたアドレスまたはドメインのセットの中にあるかどうか確認しなくてはならない[MUST]。Request-URI がプロキシが責任を負う値の maddr パラメータを持ち、リクエストがその Request-URI で(明示的あるいはデフォルトで)示されるポートとトランスポートを使用して受信された場合、プロキシは maddr およびすべての非デフォルトのポートまたはトランスポートパラメータを取り去ってリクエストにそれらの値が存在しなかったかのように処理を継続しなくてはならない[MUST]。

そのプロキシにマッチする maddr を持つが、その URI で示されるのとは違うポートとトランスポートで、リクエストは到着するかもしれない。そのようなリクエストは、示されたポートとトランスポートを使用してそのプロキシにフォワードされる必要がある。

最初の Route ヘッダフィールド値がこのプロキシを示す場合、プロキシはリクエストからその値を削除しなくてはならない[MUST]。

16.5 リクエストのターゲットの決定 (Determining Request Targets)

次に、プロキシはリクエストのターゲット(一つまたは複数)を計算する。ターゲットのセットは、リクエストのコンテンツであらかじめ決定されているか、あるいは抽象ロケーションサービスから取得されるかのいずれかである。セット中のそれぞれのターゲットは URI で表される。

リクエストの Request-URI が maddr パラメータを含む場合、ターゲットセットに唯一のターゲットとしてその Request-URI が置かれなくてはならず[MUST]、プロキシは 16.6 節の処理に進まなくてはならない[MUST]。

Request-URI のドメインがこのエレメントが責任を負わないドメインを示す場合、ターゲットセットに唯一のターゲットとしてその Request-URI が置かれなくてはならず[MUST]、エレメントはリクエストのフォワードのタスク(16.6 節)に進まなくてはならない[MUST]。

> プロキシが、それが責任を負わないドメインに対するリクエストを受け取るかもしれない、多くの状況がある。送り出される呼をハンドリングするファイアウォールプロキシ(HTTP プロキシが送り出されるリクエストをハンドリングする方法で)は、これが起こる可能性が高い場合の一つの例である。

リクエストのためのターゲットセットが上述のようにあらかじめ決定されていない場合は、Request-URI のドメインに対する責任をそのエレメントが負うことを意味し、エレメントはリクエストをどこに送るか決定するために、それが望むどのようなメカニズムでも使用してもよい[MAY]。これらのどのメカニズムも、抽象ロケーションサービスにアクセスすることとしてモデル化できる。これは、SIP レジストラサーバによって作成されたロケーションサービスから情報を取得すること、データベースから読み出すこと、プレゼンスサーバを検索すること、他のプロトコルを使用することからなるか、あるいは単純に、それらを置き換えるアルゴリズム的な処理を Request-URI に対して実行することからなる。レジストラサーバによって構築されたロケーションサービスにアクセスするとき、Request-URI はインデックスとして使用される前に、最初に 10.3 節で述べられているように正規化(canonicalized)されなくてはならない[MUST]。これらのメカニズムの出力は、ターゲットセットを構築するために使用される。

プロキシがターゲットセットを決定するために十分な情報を、Request-URI が提供しない場合、プロキシは 485(Ambiguous)レスポンスを返送するべきである[SHOULD]。このレスポンスは、試行する新しいアドレスの URI を含む Contact ヘッダフィールドを含むべきである[SHOULD]。たとえば、sip:John.Smith@company.com に対する INVITE は、複数の John Smith をリストしているロケーションサービスを参照するプロキシでは、あいまい(ambiguous)かもしれない。詳細は 21.4.23 節参照。

リクエスト中の情報またはリクエストに関する情報、あるいはエレメントの現在の環境が、ターゲットセットの構築に使用されてもよい[MAY]。たとえば、以下の条件によって、異なるセットが構築されるかもしれない。

ヘッダフィールドまたはボディの内容あるいはその存在
リクエストの到着する時間
リクエストが到着したインターフェース
前回のリクエストの失敗
エレメントの現在の稼動レベル

可能性のあるターゲットはこれらのサービスを介して場所を特定されるので、それらのURIはターゲットセットに追加される。ターゲットはターゲットセットに一度だけ置くことができる。ターゲットのURIが(URIタイプの等価性の定義に基づいて)すでにターゲットセットに存在する場合、それは再度追加されてはいけない[MUST NOT]。

プロキシは、オリジナルリクエストのRequest-URIがこのプロキシが責任を負うリソースを示さない場合に、ターゲットセットに付加的なターゲットを追加してはいけない[MUST NOT]。

プロキシは、そのURIに対して責任を負う場合にのみ、フォワードを行う間にリクエストのRequest-URIを変更できる。プロキシがそのURIに対して責任を負わない場合、以下に述べるように、それは3xxまたは416レスポンスで再帰を行わない。

オリジナルリクエストのRequest-URIがこのプロキシにおいて責任を負うリソースを示す場合、プロキシはリクエストのフォワード処理を始めた後にターゲットセットへのターゲット追加を継続してもよい[MAY]。プロキシは新しいターゲットを確定するために、その処理中に取得したどのような情報でも使用してもよい[MAY]。たとえば、プロキシはターゲットセットにリダイレクトレスポンス(3xx)で取得したコンタクト先を組み込むことを選択できる。プロキシが、ターゲットセットを構築している間に情報の動的ソースを使用する場合(たとえば、SIPレジストラサーバを検索する場合)、プロキシはリクエストを処理する間そのソースを監視するべきである[SHOULD]。新しいロケーションが有効になったら、それをターゲットセットに追加するべきである[SHOULD]。上述のように、与えられたいかなるURIも2回以上ターゲットセットに追加してはいけない[MUST NOT]。

URIをターゲットセットに1回だけ追加することを認めることは、不要なネットワークトラフィックを減らし、また、リダイレクトリクエストからコンタクト先を組み込む場合には無限回の再帰を防止する。

たとえば、普通のロケーションサービスは、ターゲットURIと送られてくるリクエストのURI(request URI)が等しい、「no-op」である。リクエストは更なる処理のために特定のネクストホップに送られる。16.6節の項目6で述べられているリクエストのフォワード処理の間、SIPまたはSIPS URIで表現されているそのネクストホップのアイデンティティがRouteヘッダフィールドの先頭の値としてリクエストに挿入される。

Request-URIがこのプロキシの存在しないリソースを示す場合、プロキシは404(Not Found)レスポンスを返送しなくてはならない[MUST]。

上記のすべてを適用した後でもターゲットセットが空のままの場合、プロキシはエラーレスポンスを返送しなくてはならない[MUST]。それは480(Temporarily Unavailable)レスポンスであるべきである[SHOULD]。

16.6 リクエストのフォワード (Request Forwarding)

ターゲットセットが空でなくなるとすぐに、プロキシはリクエストのフォワードを開始できる。ステートフルプロキシはターゲットセットをどのような順番でも処理してもよい[MAY]。ステートフルプロキシは複数のターゲットを順次に処理してもよい[MAY]。これは、各クライアントトランザクションを、次を開始する前に完了することを可能にする。ステートフルプロキシはすべてのターゲットのクライアントトランザクションを並行して開始してもよい[MAY]。ステートフルプロキシはターゲットセットを任意にグループ分けしてもよい[MAY]。そうして、グループを順次に、各グループ中のターゲットを並行して処理する。

一般的な順番付けメカニズムは、Contact ヘッダフィールドから取得したターゲットの qvalue パラメータを使用する(20.10 節参照)。ターゲットは qvalue の最も大きいものから最も小さいものの順で処理される。同じ qvalue を持つターゲットは並行して処理されるかもしれない。

ステートフルプロキシは、レスポンスを受け取ったときにターゲットセットを保持し、フォワードされた各リクエストに対するレスポンスをオリジナルリクエストと関連付けるためのメカニズムを持たなければならない。このモデルにとって、このメカニズムは、最初のリクエストをフォワードする前にプロキシレイヤで生成された「レスポンスコンテキスト(response context)」である。

各ターゲットに対して、プロキシは以下の手順に従ってリクエストをフォワードする。

1. 受け取ったリクエストのコピーを作成する。
2. Request-URI をアップデートする。
3. Max-Forwards ヘッダフィールドをアップデートする。
4. 任意に Record-route ヘッダフィールド値を追加する。
5. 任意に付加的なヘッダフィールドを追加する。
6. ルーティング情報の後処理をする。
7. ネクストホップのアドレス、ポート、およびトランスポートを確定する。
8. Via ヘッダフィールド値を追加する。
9. 必要なら、Content-Length ヘッダフィールドを追加する。
10. 新しいリクエストをフォワードする。
11. タイマーC を設定する。

これらの各手順を以下に詳述する。

1. リクエストをコピーする

   プロキシは受け取ったリクエストをコピーすることから始める。コピーは最初の状態では、受け取ったリクエストのすべてのヘッダフィールドを含まなくてはならない[MUST]。以下で述べられる処理で詳述されていないフィールドは取り除いてはいけない[MUST NOT]。コピーはヘッダフィールドの順番を受け取ったリクエストのとおり維持するべきである[SHOULD]。プロキシは一般的なフィールド名を持つフィールド値を並べ替えてはいけない[MUST NOT](7.3.1 節参照)。プロキシはメッセージボディを追加、修正、削除してはいけない[MUST NOT]。

   実際の実装ではコピーを実行する必要はない。主要な要件は、各ネクストホップの処理が同じリクエストで開始されるということである。

2. Request-URI

   コピーのスタートライン中の Request-URI は、このターゲットの URI で置き換えられなくてはならない[MUST]。URI が Request-URI で許可されていない何らかのパラメータを含む場合、それらは取り除かれなくてはならない[MUST]。

   これはプロキシの役割の本質である。これはプロキシがリクエストをそれのデスティネーション

にルーティングするメカニズムである。ある状況下では、受け取った Request-URI は修正されることなしにターゲットセットに置かれる。そのターゲットに対しては、上記の置き換え処理では事実上何も行われない(no-op)。

3. Max-Forwards

コピーが Max-Forwards ヘッダフィールドを含む場合、プロキシはその値を 1 だけ減少させなくてはならない[MUST]。

コピーが Max-Forwards ヘッダフィールドを含まない場合、プロキシは、1つのフィールド値と共にそれを追加しなくてはならず[MUST]そのフィールド値は 70 であるべきである[SHOULD]。

いくつかの既存の UA は、リクエストで Max-Forwards ヘッダフィールドを提供しない。

4. Record-Route

このプロキシが、このリクエストで生成されたダイアログ(リクエストがダイアログを生成したと仮定している)の以降のリクエストのパスに残ることを望む場合は、たとえ Route ヘッダフィールドが既に存在していたとしても、Record-Route ヘッダフィールド値をこのコピーの、既存のすべての Record-Route ヘッダフィールド値の前に挿入しなくてはならない[MUST]。

ダイアログを確立するリクエストはプリロードされた Route ヘッダフィールドを含むかもしれない。

このリクエストがすでにダイアログの一部である場合、プロキシは、ダイアログの以降のリクエストのパスに残ることを望む場合、Record-Route ヘッダフィールド値を挿入するべきである[SHOULD]。12 節で述べられているように通常のエンドポイントオペレーションにおいては、これらの Record-Route ヘッダフィールド値は、エンドポイントが使用するルートセットに何の影響も及ぼさない。

プロキシが、既にダイアログの一部であるリクエストに Record-Route ヘッダフィールド値を挿入しないことを選択する場合、プロキシはパスに残ることができる。しかしながら、エンドポイントがダイアログの再構成に失敗したときにプロキシはパスから取り除かれる。

プロキシはいかなるリクエストにも Record-Route ヘッダフィールド値を挿入してもよい[MAY]。リクエストがダイアログを開始しない場合、エンドポイントはその値を無視する。エンドポイントが Route ヘッダフィールドを構築するために Record-Route ヘッダフィールド値をどのように使用するかについての詳細は、12 節を参照のこと。

リクエストのパス中の各プロキシは、Record-Route ヘッダフィールド値を追加するかどうかを独自に選択する。リクエストに Record-Route ヘッダフィールドが存在することは、プロキシが値を追加することを義務化するものではない。

Record-Route ヘッダフィールド値に置かれた URI は、SIP URI または SIPS URI でなくてはならない[MUST]。この URI は lr パラメータを含まなくてはならない[MUST](19.1.1 節参照)。この URI

はリクエストがフォワードされる各デスティネーションごとに異なってもよい[MAY]。プロキシが、これ以降のリクエストのパスにあるネクストダウンストリームエレメントがそのトランスポートをサポートするという知識(たとえばプライベートネットワーク内で)を持つのでなければ、URI はトランスポートパラメータを含むべきではない[SHOULD NOT]。

このプロキシが提供する URI は、ルーティングの決定を行うために他のエレメントによって使用される。一般的に、このプロキシはそのエレメントがどのような能力を持つのか知るすべはない。そのため、それはそれ自身を SIP 実装の必須エレメントに限定しなければならない。すなわち、SIP URI、および、TCP または UDP トランスポートのいずれかである。

Record-Route ヘッダフィールドに置かれた URI は、参考文献[4]のサーバの位置特定手順が適用されたときに、それ以降のリクエストが同じ SIP エレメントに到着するように、それを挿入したエレメント(または適切な代理)になるように解決されなくてはならない[MUST]。Request-URI が SIPS URI を含むか、または Route ヘッダフィールドの先頭の値が(項目 6 の後処理の後に)SIPS URI を含む場合、Record-Route ヘッダフィールドに置かれた URI は SIPS URI でなくてはならない[MUST]。さらに、リクエストが TLS 上で受信されたのでなければ、プロキシは Record-Route ヘッダフィールドを挿入しなくてはならない[MUST]。同様に、TLS 上でリクエストを受け取るが、Request-URI または(項目 6 の後処理の後に)Route ヘッダフィールドの先頭の値に、SIPS URI を持たないリクエストを生成するプロキシは、SIPS URI でない Record-Route ヘッダフィー ルドを挿入しなくてはならない[MUST]。

セキュリティの境界にあるプロキシは、ダイアログ全体を通してその境界にい続けなければならない。

Record-Route ヘッダフィールドに置かれた URI が、レスポンスで送り返される過程で書き換えられる必要がある場合、その URI はその時点でロケーションを特定するのに十分明確でなくてはならない[MUST]。(リクエストはこのプロキシを通ってスパイラルするかもしれず、結果的に一つ以上の Record-Route ヘッダフィールド値が追加されることになる)16.7 節の項目 8 で URI を十分に明確にするメカニズムを推奨する。

プロキシは Record-Route ヘッダフィールド値にパラメータを含めてもよい[MAY]。これらは、INVITE に対する 200(OK)レスポンスのような、リクエストに対するある種のレスポンスで返される。このようなパラメータは、プロキシではなくメッセージに状態を保持するために有用であるかもしれない。

プロキシがどんなタイプのダイアログのパス上にもいることが必要な場合(たとえば、ファイアウォールをまたぐダイアログ)、プロキシが理解できないメソッドを持つすべてのリクエストに(そのメソッドがダイアログを解釈する処理(dialog semantics)を持つかもしれないので)Record-Route ヘッダフィールド値を追加するべきである[SHOULD]。

プロキシが Record-Route ヘッダフィールドに置く URI は、それが発生したトランザクションで生成されたすべてのダイアログの生存期間の間のみ有効である。たとえばダイアログステートフルプロキシは、Request-URI にその値を持つ以降のリクエストの受け入れを、ダイアログが終了した

後は、拒否してもよい[MAY]。当然ながら、非ダイアログステートフルプロキシはダイアログがいつ終了したかについての概念を持たないが、非ダイアログステートフルプロキシは以降のリクエストのダイアログ識別子と比較するために十分な情報をその値にエンコードしてもよく[MAY]、その情報にマッチしないリクエストを拒否してもよい[MAY]。エンドポイントはダイアログ外で提供された Record-Route ヘッダフィールドから取得した URI を使用してはいけない[MUST NOT]。エンドポイントの Record-Route ヘッダフィールドの用法についての詳細は 12 節参照のこと。

プロキシがダイアログ中のすべてのメッセージを観察する必要がある特定のサービスでは、Record-route することが必要とされるかもしれない。しかしながら、それは処理を遅くし、スケーラビリティを損なうので、プロキシは特定のサービスで必要とされた場合にのみ record-route するべきである。

Record-Route 処理は、ダイアログを開始するすべての SIP リクエストで動作するように設計されている。この仕様においては、INVITE がそのような唯一のリクエストであるが、プロトコルの拡張はその他のものを定義してもよい[MAY]。

5. 付加的なヘッダフィールドを追加する

プロキシは、この時点で、他のすべての適切なヘッダフィールドをコピーに追加してもよい[MAY]。

6. ルーティング情報の後処理をする

プロキシは、リクエストがデスティネーションに配送される前に特定のプロキシのセットを通ることを義務化する、ローカルポリシーを持ってもよい[MAY]。プロキシは、そのようなプロキシはすべてルースルータであることを保証しなくてはならない[MUST]。一般的に、これは、プロキシが同じ管理ドメイン内にいる場合にだけ確実に知ることができる。このプロキシのセットは URI のセットで表される(それぞれの URI は、lr パラメータを含む)。このセットは、コピーの Route ヘッダフィールドの(存在すれば)すべての既存の値の前に挿入されなくてはならない[MUST]。Route ヘッダフィールドがない場合には、URI のリストを含めて、Route ヘッダフィールドが追加されなくてはならない[MUST]。

プロキシが、リクエストがひとつの特定のプロキシを通ることを義務付けるローカルポリシーを持つ場合、Route ヘッダフィールドに Route 値を挿入することの代替手段は、下記の項目 10 のフォワードロジックをバイパスして、そのかわりにその特定のプロキシのアドレス、ポート、トランスポートに単純にリクエストを送ることになる。リクエストが Route ヘッダフィールドを持つ場合、ネクストホップのプロキシがルースルータであると知っているのでなければ、この代替手段を使用してはいけない[MUST NOT]。そうでなければ、このアプローチは使用してもよい[MAY]が、上記の Route 挿入メカニズムがそれのオペレーションの堅牢性、フレキシビリティ、普遍性と一貫性のために好ましい。さらに、Request-URI が SIPS URI を含む場合、そのプロキシと通信するために TLS を使用しなくてはならない[MUST]。

コピーが Route ヘッダフィールドを含む場合、プロキシはそれの最初の値に含まれる URI を検査しなくてはならない[MUST]。その URI が lr パラメータを含まない場合、プロキシはコピーを以下のように修正しなくてはならない[MUST]。

– プロキシはRouteヘッダフィールドに、それの最後の値として、Request-URIを置かなくてはならない[MUST]。

– それからプロキシは、最初のRouteヘッダフィールド値をRequest-URIに置いて、Routeヘッダフィールドからその値を削除しなくてはならない[MUST]。Routeヘッダフィールドに Request-URIを追加することは、ストリクトルーティングを行うエレメントを通してその Request-URI の情報を渡すために使用されるメカニズムの一部である。最初の Routeヘッダフィールド値を Request-URIに移動することは、ストリクトルーティングを行うエレメントが受け取ることを期待している形に(それ自身の URIを Request-URIに、次に通るロケーションを最初の Routeヘッダフィールド値に)そのメッセージを形成することになる。

7. ネクストホップのアドレス、ポート、およびトランスポートを確定する

プロキシは、RouteおよびRequest-URIの値に依存せずに特定の IPアドレス、ポート、トランスポートにリクエストを送るためのローカルポリシーを持ってもよい[MAY]。そのようなポリシーは、ルースルータであるサーバに対応する IP アドレス、ポート、トランスポートをプロキシが確実に知らない場合は、使用してはいけない[MUST NOT]。しかしながら、特定のネクストホップを通してリクエストを送るためのこのメカニズムは推奨されない[NOT RECOMMENDED]。そのかわりに上述したようにこの目的のためにはRouteヘッダフィールドが使用されるべきである。

このようなオーバーライドメカニズムがない場合には、リクエストをどこに送るか決定するために、以下のように参考文献[4]にリストされている手順を、プロキシは適用する。プロキシが、上記のステップ6で述べられたようにストリクトルーティングを行うエレメントに送るリクエストを再形成した場合、プロキシはリクエストの Request-URIにそれらの手順を適用しなくてはならない[MUST]。そうでなければ、プロキシはその手順を最初の Routeヘッダフィールド値に(存在すれば)、さもなければ Request-URIに適用しなくてはならない[MUST]。この手順は順番に並べられたタプル(tuples)(アドレス、ポート、トランスポート)のセットを生成する。参考文献[4]の手順の入力としてどの URIが使用されるかに関係なく、Request-URIが SIPSリソースを明示する場合、プロキシは入力された URI が SIPS URI であるかのように参考文献[4]の手順にしたがわなくてはならない[MUST]。

参考文献[4]で述べられているように、プロキシはそのセットの最初のタプルにメッセージを配送することを試み、そして配送の試みが成功するまでセットの中を順番に処理していかなくてはならない[MUST]。

それぞれのタプルの試行では、プロキシはそのタプルに適切なようにメッセージを形成してステップ 8 から 10 で述べられているように新規クライアントトランザクションを使用してそのリクエストを送らなくてはならない[MUST]。

各試行は新規クライアントトランザクションを使用するので、それは新しい branch を意味することになる。したがって、ステップ 8 で挿入される Via ヘッダフィールドで提供される branch パラメータは、各試行で異ならなくてはならない[MUST]。

クライアントトランザクションがそれのステートマシーンから、リクエスト送信の失敗またはタイムアウトの報告をする場合、プロキシは、順番に並べられたセット中の次のアドレスで処理を継続する。順番に並べられたセットの最後までいった場合、リクエストはターゲットセットのこのエレメントにフォワードできない。プロキシはレスポンスコンテキストに何も置く必要はないが、ターゲットセットのこのエレメントがあたかも 408(Request Timeout)最終レスポンス(Final response)を返送したかのように動作する。

8. Via ヘッダフィールド値を追加する

プロキシは、コピーの既存の Via ヘッダフィールド値の前に Via ヘッダフィールド値を挿入しなくてはならない[MUST]。この値の構築は 8.1.1.7 節と同じガイドラインにしたがう。これは、プロキシが、その branch のためのグローバルに一意で必須のマジッククッキーを含む、それ自身の branch パラメータを計算することを意味する。このことは、プロキシ経由のスパイラルまたはループしたリクエストの異なるインスタンスでは、branch パラメータが異なることを暗示する、ということに注意。

ループを検知することを選択するプロキシは、branch パラメータの構築に使用する値に関して付加的な制約を持つ。ループを検知することを選択するプロキシは、実装によって 2 つの部分に分離可能な branch パラメータを作成するべきである[SHOULD]。最初の部分は上述のように 8.1.1.7 節の制約を満たさなくてはならない[MUST]。2 つめの部分はループ検知を実行するため、およびループをスパイラルと区別するために使用される。

ループの検知は、リクエストがプロキシに返送されたときに、リクエストの処理に大きな影響を及ぼすフィールドが変更されていないことを検証をすることによって実行される。branch パラメータのこの部分に置かれる値は、それらのすべてのフィールド(すべての Route、Proxy-Require、および Proxy-Authorization ヘッダフィールドを含む)を反映するべきである[SHOULD]。これは、リクエストがプロキシにルーティングされて戻ってきて、それらのフィールドの一つが変更されていた場合に、それがループではなくスパイラルとして扱われること(16.3 節参照)を保証するためである。この値を生成するための一般的な方法は、(存在するかもしれないすべての Proxy-Require と Proxy-Authorization ヘッダフィールドに加えて)To tag、From tag、Call-ID ヘッダフィールド、受け取ったリクエストの(変換前の)Request-URI、先頭の Via ヘッダ、および CSeq ヘッダフィールドのシーケンス番号の暗号ハッシュを計算することである。ハッシュを計算するために使用されるアルゴリズムは実装に依存するが、16 進数で表現された MD5 (RFC 1321 [35])が妥当な選択である。(Base64 はトークンとして許可されない。)

プロキシがループを検知することを望む場合、それが供給する branch パラメータは、送られてくる Request-URI とリクエストの許可(admission)とルーティング決定に影響を及ぼすすべてのヘッダフィールドを含めて、リクエスト処理に影響を及ぼすすべての情報に依存しなくてはならない[MUST]。これは、ループしたリクエストとこのサーバに返送される前にルーティングパラメータが変更されたリクエストを区別するために必要である。

リクエストのメソッドは branch パラメータの計算に含めてはいけない[MUST NOT]。特に、CANCEL リクエストと ACK リクエスト(非 2xx レスポンスに対する)は、それがキャンセルまたは同意する対応するリクエストのものと同じ branch 値を持たなくてはならない[MUST]。branch パラメータは、リクエストをハンドリングするサーバで、リクエストを関連付けるために使用される。(17.2.3 節および 9.2 節参照)。

9. 必要であれば、Content-Length ヘッダフィールドを追加する

リクエストがストリームベースのトランスポートを使用して送られることになり、かつ、コピーが Content-Length ヘッダフィールドを含まない場合、プロキシはリクエストのボディに対して正しい値を持つ Content-Length ヘッダフィールドを挿入しなくてはならない[MUST](20.14 節参照)。

10. リクエストをフォワードする

ステートフルプロキシは、17.1 節に述べられているようにこのリクエストのために新規クライアントトランザクションを生成してそのトランザクションにステップ 7 で決定されたアドレス、ポート、トランスポートを使用してリクエストを送るように指示しなくてはならない[MUST]。

11. タイマーC を設定する

INVITE リクエストがいつまでも最終レスポンス(Final response)を生成しない場合をハンドリングするために、TU はタイマーC と呼ばれるタイマーを使用する。タイマーC は、INVITE リクエストがプロキシされたときにそれぞれのクライアントトランザクションに対して設定されなくてはならない[MUST]。タイマーC は 3 分よりも長くなくてはならない[MUST]。16.7 節の項目 2 で、このタイマーを暫定レスポンス(provisional response)でどのようにアップデートするかについて議論する。また、16.8 節で、タイマーが切れたときの処理について議論する。

16.7 **レスポンスの処理** (Response Processing)

エレメントがレスポンスを受け取るとき、エレメントはそのレスポンスにマッチするクライアントトランザクションを見つけることを最初に試みる(セクショ 17.1.3)。一つも見つからない場合、エレメントはそのレスポンスを(たとえそれが通知レスポンスだとしても)ステートレスプロキシ(以下に記述されている)として処理しなくてはならない[MUST]。マッチするものが見つかった場合、レスポンスはクライアントトランザクションに渡される。

対応するクライアントトランザクション(あるいはより一般的には、関連付けられたリクエストを送ったという知識)が見つからないレスポンスをフォワードすることは堅牢性を高める。とりわけそれは、INVITE リクエストに対する「遅れた」2xx レスポンスが適切にフォワードされることを保証する。

クライアントトランザクションがレスポンスをプロキシレイヤに渡すときは、以下の処理が行われなくてはならない[MUST]。

1. 適切なレスポンスコンテキストを探し出す。
2. 暫定レスポンス(provisional response)に対してタイマーC をアップデートする。
3. 最初の Via を取り除く。
4. レスポンスコンテキストにレスポンスを追加する。
5. このレスポンスが直ちにフォワードされるべきかどうか確認する。
6. 必要なときは、レスポンスコンテキストから最適な最終レスポンス(Final response)を選択する。

レスポンスコンテキストに関連付けられたすべてのクライアントトランザクションが終了した後に、最終レスポンス(Final response)が一つもフォワードされていない場合、プロキシは、これまでに見たそれらの

なかから「最適の」レスポンスを選択してフォワードしなければならない。

フォワードされる各レスポンスにおいて、以下の処理が実行されなくてはならない[MUST]。各リクエストに対して 2 つ以上のレスポンスがフォワードされることがあり得る。少なくとも各暫定レスポンス(provisional response)と一つの最終レスポンス(Final response)である。

7. 必要であれば認可のためのヘッダフィールド値を集約する
8. 任意で Record-Route ヘッダフィールド値を書き換える
9. レスポンスをフォワードする
10. 必要なすべての CANCEL リクエストを生成する

上記の各手順を以下に詳述する。

1. コンテキストを見つける

   プロキシは、オリジナルリクエストをフォワードする前に、16.6 節述べられているキーを使用して、それが生成した「レスポンスコンテキスト」を探し出す。残りの処理手順はこのコンテキストで行われる。

2. 暫定レスポンス(provisional response)に対してタイマーC をアップデートする

   INVITE トランザクションにおいて、レスポンスがステータスコード 101～199(すなわち、100 以外)の暫定レスポンス(provisionalresponse)である場合、プロキシはそのクライアントトランザクションのタイマーC をリセットしなくてはならない[MUST]。タイマーC は別の値にリセットしてもよい[MAY]が、この値は 3 分よりも長くなくてはならない[MUST]。

3. Via

   プロキシはレスポンスから最初の Via ヘッダフィールド値を取り除く。

   レスポンスに Via ヘッダフィールド値が一つも残らない場合、そのレスポンスはこのエレメントに宛てられたものであるので、フォワードしてはいけない[MUST NOT]。本節で述べられている残りの処理はこのメッセージには実行されない。そのかわりに、8.1.3 節で述べられている UAC の処理ルールにしたがう(トランスポートレイヤの処理は既に行われている)。

   たとえば、10 節で述べられているようにエレメントが CANCEL リクエストを生成するときにこれが起こる。

4. コンテキストにレスポンスを追加する

   受け取った最終レスポンス(Final response)は、このコンテキストに関連付けられたサーバトランザクションで最終レスポンス(Final response)が生成されるまで、レスポンスコンテキストに保存される。そのレスポンスは、そのサーバトランザクションで返却される最適な最終レスポンス(Final response)の候補になるかもしれない。このレスポンスからの情報は、このレスポンスが選択されなかったとしても、最適なレスポンスを形成するときに必要になるかもしれない。

   プロキシが、3xx レスポンスのコンタクト先をターゲットセットに追加することで、それらのう

ちのどれかに再帰(recurse)することを選択した場合、プロキシはレスポンスをレスポンスコンテキストに追加する前にそのレスポンスからそれらを取り除かなくてはならない[MUST]。しかしながら、プロキシは、オリジナルリクエストの Request-URI が SIPS URI の場合に非 SIPS URI に再帰するべきではない[SHOULD NOT]。プロキシが 3xx レスポンスのすべてのコンタクト先に再帰する場合、プロキシはその結果としてのコンタクト先を持たないレスポンスをレスポンスコンテキストに追加するべきではない[SHOULD NOT]。

レスポンスをレスポンスコンテキストに追加する前にコンタクト先を取り除くことは、アップストリームの次のエレメントがこのプロキシがすでに試行したロケーションを再試行することを防止する。

3xx レスポンスは SIP URI、SIPS URI、および非 SIP URI が混ざったものを含むかもしれない。プロキシは、SIP URI および SIPS URI に再帰することを選択し、残りを最終レスポンス(Final response)で返送される可能性のあるレスポンスコンテキストに置くかもしれない。

プロキシが、Request-URI のスキームが SIP でなかった(しかしオリジナルに受け取ったリクエストのスキームは SIP または SIPS であった)(すなわち、プロキシするときにプロキシがスキームを SIP または SIPS からそれ以外のものに変更した)リクエストに対して 416(Unsupported URI Scheme)レスポンスを受け取る場合、プロキシはターゲットセットに新しい URI を追加するべきである[SHOULD]。この URI は、今試行したばかりの非 SIP URI に対応する SIP URI であるべきある[SHOULD]。tel URL の場合は、tel URL の telephone-subscriber 部分を SIP URI のユーザ部分に、そしてホスト部分に前回のリクエストを送ったドメインを設定することで、これが実現できる。tel URL から SIP URI を形成することについての更なる詳細は、19.1.6 節参照のこと。

3xx レスポンスと同様に、プロキシが SIP URI または SIPS URI で試行することによって 416 で「再帰」する場合、416 レスポンスはレスポンスコンテキストに追加されるべきではない[SHOULD NOT]。

5. フォワードするレスポンスを確認する

最終レスポンス(Final response)がサーバトランザクションで送られてしまうまで、以下のレスポンスは直ちにフォワードされなくてはならない[MUST]。

- 100(Trying)以外のすべての暫定レスポンス(provisional response)
- すべての 2xx レスポンス

6xx レスポンスを受け取る場合、それは直ちにフォワードされないが、ステートフルプロキシは保留されているすべてのクライアントトランザクションを 10 節で述べられているようにキャンセルするべきであり[SHOULD]、このコンテキストでいかなる新規 branch も生成してはいけない[MUST NOT]。

これは、6xx を直ちにフォワードすることを義務付けていた RFC2543 からの変更である。INVITE トランザクションにおいては、このアプローチは 2xx が別の branch で到着する可能性があるという問題をかかえていた。その場合、プロキシはその 2xx をフォワードしなければならないだろう。結果として、決して起こることが認められるべきでない、2xx レスポンスが後に続く 6xx レスポン

スを UAC が受け取ることがありうる。新しいルールの下では、6xx を受け取ったときに、プロキシは通常すべてのクライアントトランザクションから 487 レスポンスを受け取るという結果になる CANCEL を発行し、それからその時点で 6xx をアップストリームにフォワードする。

サーバトランザクションで最終レスポンス(Final response)が送られた後で、以下のレスポンスが直ちにフォワードされなくてはならない[MUST]。

- INVITE リクエストに対するすべての 2xx レスポンス

ステートフルプロキシはそれ以外のレスポンスを直ちにフォワードしてはいけない[MUST NOT]。特に、ステートフルプロキシはすべての 100(Trying)レスポンスをフォワードしてはいけない[MUST NOT]。「最適な」レスポンスとして後にフォワードするための候補であるレスポンスが、「コンテキストにレスポンスを追加する」手順で述べられているように収集された。

直ちにフォワードするために選ばれたすべてのレスポンスは、「認可のためのヘッダフィールド値を集約する」から「Record-Route」の手順で述べられているように処理されなくてはならない[MUST]。

この手順は、次の手順と合わせて、ステートフルプロキシが非 INVITE リクエストに対して正確に一つの最終レスポンス(Final response)をフォワードすることと、INVITE リクエストに対して正確に一つの非 2xx レスポンスまたは一つ以上の 2xx レスポンスをフォワードすることを保証する。

6．最適なレスポンスを選択する

ステートフルプロキシは、上記のルールによって最終レスポンス(Final response)が一つも直ちにフォワードされておらず、このレスポンスコンテキスト中のすべてのクライアントトランザクションが終了している場合、レスポンスコンテキストのサーバトランザクションに対して最終レスポンス(Final response)を送らなくてはならない[MUST]。

ステートフルプロキシは、受け取ってレスポンスコンテキストに保存されたものの中から「最適な」最終レスポンス(Final response)を選ばなくてはならない[MUST]。

レスポンスコンテキストの中に最終レスポンス(Final response)が一つもない場合、プロキシはサーバトランザクションに 408(Request Timeout)レスポンスを送らなくてはならない[MUST]。

さもなくば、プロキシはレスポンスコンテキストに保存されたレスポンスの中から一つのレスポンスをフォワードしなくてはならない[MUST]。プロキシはレスポンスコンテキストに 6xx クラスレスポンスがあればその中から選択しなくてはならない[MUST]。6xx クラスレスポンスが存在しない場合、プロキシはレスポンスコンテキストに保存されている最下位のレスポンスクラスから選択するべきである[SHOULD]。プロキシは選択したクラスからどのレスポンスを選択してもよい[MAY]。プロキシは、このリクエストの再提出に影響を及ぼす情報を提供するレスポンスにプリファレンスを与えるべきである[SHOULD]。たとえば、4xx クラスが選択された場合には、401、407、415、420、484 というように。503(Service Unavailable)レスポンスを受け取るプロキシは、今後プロキシするかもしれないすべてのリクエストも 503 を生成すると確定できなければ、それをアップストリー

ムにフォワードするべきではない[SHOUL NOT]。言い換えると、503をフォワードするということは、そのプロキシが、503 を生成したリクエスト中の Request-URI に対してだけでなく、どのリクエストにもサービスできないことをそれ自身が知っていることを意味する。受け取った唯一のレスポンスが 503 の場合、プロキシは 500 レスポンスを生成してそれをアップストリームにフォワードするべきである[SHOULD]。

フォワードされたレスポンスは、「認可のためのヘッダフィールド値をまとめる」から「Record-Route」の手順で述べられているように処理されなくてはならない[MUST]。

たとえば、プロキシが 4 つのロケーションにリクエストをフォワードし、503、407、501、および 404 レスポンスを受け取った場合、それは 407(Proxy Authentication Required)レスポンスをフォワードすることを選択できる。

ダイアログの確立には 1xx および 2xx レスポンスが関与するかもしれない。リクエストが To tag を含まないとき、レスポンス中の To tag はダイアログを生成するリクエストに対する複数のレスポンスを区別するために UAC が使用する。プロキシは、リクエストが To tag を含んでいなければ、1xx または 2xx レスポンスの To ヘッダフィールドに tag を挿入してはいけない[MUST NOT]。プロキシは 1xx または 2xx レスポンスの To ヘッダフィールドの tag を修正してはいけない[MUST NOT]。

プロキシは、To tag を含まなかったリクエストに対する 1xx レスポンスの To ヘッダフィールドに tag を挿入しないかもしれないので、プロキシはそれ自身で非 100 暫定レスポンス(provisional response)を発行できない。しかしながら、プロキシは、プロキシとして同じエレメントを共有する UAS にリクエストをブランチできる。この UAS は、そのリクエストの起動者と early ダイアログに入ることで、それ自身の暫定レスポンス(provisional response)を返送することができる。UAS はプロキシから分離したプロセスである必要はない。それはプロキシと同じコード空間に実装されたバーチャル UAS であり得る。

3-6xx レスポンスはホップバイホップで配送される。3-6xx レスポンスを発行するとき、エレメントは、通常ダウンストリームエレメントから受け取ったレスポンスに基づいてそれ自身のレスポンスを発行しながら、実質的に UAS として動作している。エレメントは To tag を含まなかったリクエストに対する 3-6xx レスポンスを単純にフォワードするときは、To tag を保つべきである[SHOULD]。

プロキシは To tag を含むリクエストに対するフォワードされたいかなるレスポンスの To tag も修正してはいけない[MUST NOT]。

プロキシがフォワードされた 3-6xx レスポンスの To tag を置き換えてもアップストリームエレメントに対して重大な影響はないが、オリジナルの tag を保つことはデバッグの役に立つかもしれない。

プロキシがいくつかのレスポンスから情報を集約するとき、それらの中から To tag を選択することは任意であり、新しい To tag を生成することはデバッグを容易にするかもしれない。これは、たとえば、401(Unauthorized)と 407(Proxy Authentication Required)チャレンジを結合するとき、

あるいは暗号化されておらず認証もされていない 3xx レスポンスの Contact 値を結合するときに、起こる。

7．認可のためのヘッダフィールド値を集約する

選択されたレスポンスが 401(Unauthorized)あるいは 407(Proxy Authenti cation Required)の場合、プロキシは、このレスポンスコンテキストでこれまでに受け取った他のすべての 401(Unauthorized) と 407(Proxy Authentication Required) レスポンスからすべての WWW-Authenticate ヘッダフィールドと Proxy-Authenticate ヘッダフィールドを収集し、フォワードする前にそれらをこのレスポンスに修正せずに追加しなくてはならない[MUST]。結果としての 401(Unauthorized) または 407(Proxy Authentication Required) レスポンスはいくつかの WWW-Authenticate および(AND)Proxy-Authenticate ヘッダフィールド値を持つことがあり得る。

リクエストがフォワードされたデスティネーションのどれかあるいはすべてが信用証明書を要求していたかもしれないので、これは必須である。クライアントは、そのリクエストをリトライするときに、それらすべてのチャレンジを受け取り、それらの各々に対して信用証明書を供給する必要がある。この動作の動機付けは 26 節で提供される。

8．Record-Route

選択したレスポンスがもともとこのプロキシが提供した Record-Route ヘッダフィールド値を含む場合、プロキシはそのレスポンスをフォワードする前にその値を書き換えることを選択してもよい[MAY]。これはそのプロキシが、次のアップストリームエレメントおよびダウンストリームエレメントに、それ自身の別の URL(複数)を提供することを可能にする。プロキシはどのような理由でもこのメカニズムを使用することを選択できる。たとえば、それはマルチホームホストに対して有用である。

プロキシが TLS 上でリクエストを受け取り、非 TLS コネクションでそれを送る場合、プロキシは Record-Route ヘッダフィールドの URI を SIPS URI に書き換えなくてはならない[MUST]。プロキシが非 TLS コネクションでリクエストを受け取り、TLS 上でそれを送る場合、プロキシは Record-Route ヘッダフィールドの URI を SIP URI に書き換えなくてはならない[MUST]。

プロキシによって提供された新しい URI は、リクエストの Record- Route ヘッダフィールドに置かれる URI に対するものと同じ制約(16.6 節のステップ 4 参照)を、以下の変更点と共に満たさなくてはならない[MUST]。

今後のリクエストのパスにある次のアップストリーム(ダウンストリームとは逆に)エレメントが、そのトランスポートをサポートするという知識を持っていない限り、URI はトランスポートパラメータを含むべきではない[SHOULD NOT]。

プロキシがレスポンスの Record-Route ヘッダフィールドの修正を決定するとき、それが実行するオペレーションのひとつは、それが挿入した Record- Route 値を見つけ出すことである。そのリクエストがスパイラルし、プロキシがスパイラルの各繰り返し(iteration)で Record-Route を挿入した場合、レスポンスの中に正しい値を見つけること(それは反対方向への適切な繰り返し(iteration)でなければならない)は厄介である。上記のルールは、Record-Route ヘッダフィールド

値の書き換えを望むプロキシが、書き換える正しい Record-Route ヘッダフィールドが選択されるように、Record-Route ヘッダフィールドに十分に明瞭な URI を挿入することを推奨する。これを実現するために推奨される[RECOMMENDED]メカニズムは、プロキシが URI のユーザ部分にそのプロキシのインスタンスのための一意の識別子を追加することである。

レスポンスが到着するとき、プロキシはプロキシのインスタンスにマッチする識別子を持つ最初の Record-Route を修正する。修正によって、その URI のユーザ部分に追加されたこのデータ片を持たない URI を得ることになる。次の繰り返し(iteration)時に、同じアルゴリズム(そのパラメータを持つ最初の Record-Route ヘッダフィールド値を見つける)が、そのプロキシが挿入した次の Record-Route ヘッダ値を適確に抽出する。

プロキシが Record-Route ヘッダフィールド値を追加したリクエストに対するすべてのレスポンスが Record-Route ヘッダフィールドを含むわけではない。レスポンスが Record-Route ヘッダフィールドを含む場合、それはプロキシが追加した値を含む。

9．レスポンスをフォワードする

「認可のためのヘッダフィールド値を集約する」から「Record-Route」で述べられている手順を実行した後で、プロキシは選択されたレスポンスに対して、プロキシの機能特有のどのような操作でも実行してもよい[MAY]。プロキシはメッセージボディを、追加、修正、あるいは削除してはいけない[MUST NOT]。特に規定されなければ、プロキシは、16.7 節の項目 3 で議論されている Via ヘッダフィールド値以外のいかなるヘッダフィールド値も取り除いてはいけない[MUST NOT]。特に、プロキシは、それがこのレスポンスに関連付けられたリクエストを処理するときに次の Via ヘッダフィールド値に追加したのかもしれないいかなる received パラメータも削除してはいけない[MUST NOT]。プロキシは、レスポンスコンテキストに関連付けられたサーバトランザクションにレスポンスを渡さなくてはならない[MUST]。このことで、今現在の最初の Via ヘッダフィールド値で示されるロケーションにレスポンスが送られることになる。サーバトランザクションが既にその送信をハンドリングするために有効でない場合、エレメントはそのレスポンスをサーバトランスポートに送ることによってステートレスにフォワードしなくてはならない[MUST]。サーバトランザクションは、レスポンス送信の失敗を示すか、あるいはそれのステートマシーン内でタイムアウトを知らせるかもしれない。これらのエラーは、必要に応じて診断目的でログに記録されるが、プロトコルはプロキシが是正措置を取ることを何ら要求しない。

最終レスポンス(Final response)をフォワードした後でも、プロキシはレスポンスコンテキストに関連付けられたすべてのトランザクションが終了するまで、そのレスポンスコンテキストを保持しなくてはならない[MUST]。

10．CANCEL を生成する

フォワードされたレスポンスが最終レスポンス(Final response)であった場合、プロキシはこのレスポンスコンテキストに関連付けられた保留中のすべてのクライアントトランザクションに対して CANCEL リクエストを生成しなくてはならない[MUST]。プロキシは 6xx レスポンスを受け取ったときにも、このレスポンスコンテキストに関連付けられた保留中のすべてのクライアントトランザクションに対して CANCEL リクエストを生成するべきである[SHOULD]。保留中のクライアントトランザクションとは、暫定レスポンス(provisional response)を受け取っているが最終レスポンス

(Final response)は受け取っておらず(それは進行中状態である)、それに対して関連付けられたCANCELも生成していないものである。CANCELリクエストの生成は9.1節で述べられている。

最終レスポンス(Final response)フォワード時に保留中のクライアントトランザクションをCANCELする要件は、エンドポイントがINVITEに対する複数の200(OK)レスポンスを受け取らないことを保証するものではない。CANCELリクエストが送られて処理される前に、2つ以上のブランチで200(OK)レスポンスが生成されるかもしれない。さらに、将来の拡張がCANCELリクエストを発行するためのこの要件をオーバーライドするかもしれないことを予期するのが妥当である。

16.8 タイマーCの処理 (Processing Timer C)

万一タイマーCが切れる場合、プロキシはそれが選んだ何らかの値でタイマーをリセットするか、あるいはクライアントトランザクションを終了するかしなくてはならない[MUST]。クライアントトランザクションが暫定レスポンス(provisional response)を受け取っている場合、プロキシはそのトランザクションにマッチするCANCELリクエストを生成しなくてはならない[MUST]。クライアントトランザクションが暫定レスポンス(provisional response)を受け取っていない場合、プロキシはそのトランザクションがあたかも408(Request Timeout)レスポンスを受け取ったかのように動作しなくてはならない[MUST]。

プロキシがタイマーをリセットすることを許可することは、タイマーが切れたときに現在の状態(たとえば稼動率)に基づいてトランザクションの生存期間をプロキシが動的に伸張する事を可能にする。

16.9 トランスポートエラーのハンドリング (Handling Transport Errors)

トランスポートレイヤがリクエストのフォワードを試みるときに(18.4節参照)、プロキシにエラーを通知する場合、プロキシはフォワードされたリクエストが503 (Service Unavailable)レスポンスを受け取ったかのように動作しなくてはならない[MUST]。

プロキシがレスポンスをフォワードするときにエラー通知を受けたときは、そのレスポンスをやめる。プロキシは、この通知が原因でこのレスポンスコンテキストに関連付けられたまだ残っているすべてのクライアントトランザクションをキャンセルするべきではない[SHOULD NOT]。

プロキシがそれのまだ残っているクライアントトランザクションをキャンセルする場合、一つの悪意のあるまたは誤った動作をするクライアントが、それのViaヘッダフィールドを通してすべてのトランザクションを失敗させる原因になることがありうる。

16.10 CANCELの処理 (CANCEL Processing)

ステートフルプロキシはいつでも、それが生成したCANCEL以外のどのようなリクエストに対してでもCANCELを生成してもよい[MAY](9.1節で述べられているように、そのリクエストに対する暫定レスポンス(provisional response)を受け取ることを前提として)。プロキシは、マッチするCANCELリクエストを受け取るとき、レスポンスコンテキストに関連付けられた保留中のどのようなクライアントトランザクションもキャンセルしなくてはならない[MUST]。

ステートフルプロキシは、INVITEのExpiresヘッダフィールドで指定された期間の経過に伴い、ペンディング中のINVITEクライアントトランザクションに対してCANCELを生成してもよい[MAY]。しかしながら、関係するエンドポイントがトランザクション終了の合図をすることを引き受けるので、通常これを行う必要

はない。

CANCEL リクエストが、ステートフルプロキシでそれ自身のサーバトランザクションによってハンドリングされるとはいえ、それのための新規レスポンスコンテキストは生成されない。その代わりに、プロキシレイヤは、この CANCEL に関連付けられたリクエストをハンドリングするサーバトランザクションのために、それの既存のレスポンスコンテキストを検索する。マッチするレスポンスコンテキストが見つかった場合、そのエレメントは CANCEL リクエストに対して直ちに 200(OK)レスポンスを返送しなくてはならない[MUST]。この場合、エレメントは 8.2 節で定義されているようにユーザエージェントサーバとして動作している。さらに、エレメントは 16.7 節のステップ 10 で述べられているように、そのコンテキストのすべてのペンディング中のクライアントトランザクションに対して CANCEL を生成しなくてはならない[MUST]。

レスポンスコンテキストが見つからない場合、エレメントは CANCEL を適用するリクエストに関する知識を何ら持っていない。それは CANCEL リクエストをステートレスでフォワードしなくてはならない[MUST](それは関連付けられたリクエストをこれより前にステートレスでフォワードしてしまっているのかもしれない)。

16.11 ステートレスプロキシ (Stateless Proxy)

ステートレスで動作するとき、プロキシは単にメッセージをフォワードするものである。ステートレスに動作するときに実行されるほとんどの処理は、ステートフルに動作するときと同じである。相違点をここで詳述する。

ステートレスプロキシはトランザクションに関するいかなる概念も、あるいはステートフルプロキシの動作を記述するために使用されるレスポンスコンテキストに関するいかなる概念も持たない。その代わりに、ステートレスプロキシは、リクエストとレスポンスの両方で、トランスポートレイヤ(18 節参照)から直接メッセージを取り込む。結果として、ステートレスプロキシはそれ自身でメッセージを再送しない。しかしながら、ステートレスプロキシはそれが受け取るすべての再送をフォワードする(ステートレスプロキシは再送をオリジナルメッセージと区別する能力を持たない)。更には、リクエストをステートレスでハンドリングするとき、エレメントはそれ自身の 100(Trying)レスポンスをまたは他のいかなる暫定レスポンス(provisional response)も生成してはいけない[MUST NOT]。

ステートレスプロキシは 16.3 節で述べられているようにリクエストを検証しなくてはならない[MUST]。

ステートレスプロキシは以下のことを例外として、16.4 節から 16.5 節で述べられているリクエスト処理手順にしたがわなくてはならない[MUST]。

- ステートレスプロキシは、ターゲットセットからただ一つのターゲットを選択しなくてはならない[MUST]。この選択は、メッセージ中のフィールドと時間によって変化しないサーバのプロパティのみを基にしなくてはならない[MUST]。特に、再送されたリクエストは、それが処理される毎に同じデスティネーションにフォワードされなくてはならない[MUST]。更に、CANCEL リクエストと Route をふくんでいない(non-Routed)ACK リクエストは、それらが関連付けられている INVITE と同じ選択を行わなくてはならない[MUST]。

ステートレスプロキシは下記のことを例外として、16.6 節で述べられているように、リクエスト処理手順

にしたがわなくてはならない[MUST]。

- あらゆる時間と空間に渡って一意である branch ID のための要件は、ステートレスプロキシにも同様に適用される。しかしながら、ステートレスプロキシは、16.6 節の項目 8 で述べられているように branch ID の最初のコンポーネントを計算するために単純に乱数発生器を使用できない。これは、リクエストの再送が同じ値を必要とするためであり、ステートレスプロキシはオリジナルリクエストと再送を区別できないためである。したがって、branch パラメータをユニークにするコンポーネントは、再送されたリクエストがフォワードされるたびに同じでなくてはならない[MUST]。よってステートレスプロキシでは、branch パラメータは、再送時に変化しないメッセージパラメータを組み合わせた関数で計算されなくてはならない[MUST]。

  ステートレスプロキシは、トランザクション全体を通してそれの branch ID の一意性を保証するために、それが望むどのようなテクニックでも使用してもよい[MAY]。しかしながら、以下の手順が推奨される[RECOMMENDED]。プロキシは受け取ったリクエストの最初の Via ヘッダフィールドの branch ID を調査する。それがマジッククッキーで始まる場合、送り出されるリクエストの branch ID の最初のコンポーネントは、受け取った branch ID のハッシュとして計算される。そうでなければ、branch ID の最初のコンポーネントは、受け取ったリクエストの最初の Via、To ヘッダフィールドの tag、From ヘッダフィールドの tag、Call-ID ヘッダフィールド、CSeq 番号(メソッドではない)、および Request- URI のハッシュとして計算される。これらのフィールドの一つは異なる 2 つのトランザクションにおいて常に異なる。

- 16.6 節で規定されている他のすべてのメッセージ変換は、再送されたリクエストの変換と同じ結果にならなくてはならない[MUST]。特に、プロキシが Record-Route 値を挿入するか、または Route ヘッダフィールドに URI を入れる場合には、プロキシはリクエストの再送にも同じ値を置かなくてはならない[MUST]。Via の branch パラメータでは、変換が、時間によって変化しない設定または再送によって変化しないリクエストのプロパティに基づかなくてはならない[MUST]ことを、これは意味する。

- ステートレスプロキシは、16.6 節の項目 10 でステートフルプロキシのために述べられているように、リクエストをどこにフォワードするか決定する。リクエストは、クライアントトランザクションを通してではなく直接トランスポートレイヤに送られる。

  ステートレスプロキシは再送されたリクエストを同じデスティネーションにフォワードしなければならず、それらに同一の branch パラメータを追加しなければならないので、メッセージ自身からの情報と時間によって変化しない設定データをそれらの計算に使用する。設定状態が時間によって変化する場合(たとえば、ルーティングテーブルがアップデートされる場合)、その変更に影響を受ける可能性のあるいかなるリクエストも、その変更の前後、トランザクションタイムアウト時間と同じ期間だけ、ステートレスにフォワードされないかもしれない。その期間に、影響を受けるリクエストを処理する方法は実装が決める。一般的な解決策は、それらをトランザクションステートフルにフォワードすることである。

ステートレスプロキシは、CANCEL リクエストに対して特別な処理を実行してはいけない[MUST NOT]。CANCEL

リクエストは他のいかなるリクエストとも同じように上記のルールによって処理される。特に、ステートレスプロキシは、他のいかなるリクエストに適用するのとも同じ Route ヘッダ処理を CANCEL リクエストにも適用する。

16.7 節で述べられているようなレスポンスの処理は、ステートレスに動作するプロキシには適用されない。レスポンスがステートレスプロキシに到着するとき、そのプロキシは最初の(先頭の)Via ヘッダ値の sent-by 値を検査しなくてはならない[MUST]。そのアドレスがそのプロキシにマッチする場合(このプロキシが以前のリクエストに挿入した値と等しい場合)、プロキシはレスポンスからそのヘッダフィールド値を取り除いて次の Via ヘッダフィールド値で示されるロケーションにその結果をフォワードしなくてはならない[MUST]。プロキシはメッセージボディを追加、修正、削除してはいけない[MUST NOT]。特に規定されていなければ、プロキシは他のいかなるヘッダフィールド値も取り除いてはいけない[MUST NOT]。アドレスがプロキシにマッチしない場合、メッセージはだまって破棄されなくてはならない[MUST]。

16.12 プロキシのルート処理のまとめ (Summary of Proxy Route Processing)

ローカルポリシーがない場合、プロキシが Route ヘッダフィールドを含むリクエストで実行する処理は以下のようにまとめることができる。

1. プロキシは Request-URI を検査する。それがこのプロキシが所持するリソースを示す場合、プロキシはロケーションサービスを実行した結果でそれを置き換える。そうでなければ、プロキシは Request-URI を変更しない。

2. プロキシは最初の Route ヘッダフィールド値の URI を検査する。それがこのプロキシを示す場合、プロキシはそれを Route ヘッダフィールドから取り除く(この経路(ルート)ノードに到着したので)。

3. プロキシはリクエストを最初の Route ヘッダフィールド値の URI、または、Route ヘッダフィールドが存在しない場合には Request-URI の URI で示されるリソースにフォワードする。プロキシはリクエストをフォワードするときに、その URI に参考文献[4]の手順を適用して、使用するアドレス、ポート、トランスポートを決定する。

リクエストのパス上でストリクトルーティングを行うエレメントに出会わない場合、Request-URI は常にリクエストのターゲットを示す。

16.12.1 例 (Examples)

16.12.1.1 基本的な SIP トラペゾイド (Basic SIP Trapezoid)

このシナリオは、双方のプロキシが record-route を行う、基本的な SIP トラペゾイド(台形)(U1 -> P1 -> P2 -> U2)のものである。以下がそのフローである。

U1 が、

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

を P1 に送る。P1 はアウトバウンドプロキシである。P1 は domain.com に対して責任を負わないので、DNS をルックアップし、そこに送る。P1 は Record-Route ヘッダフィールド値も追加する。

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

P2 がこれを受け取る。P2 は domain.com に対して責任を負うので、ロケーションサービスを実行して Request-URI を書き換える。P2 は Record-Route ヘッダフィールド値も追加する。Route ヘッダフィールドがないので、P2 はリクエストをどこに送るか決定するために新しい Request-URI を解決する。

```
INVITE sip:callee@u2.domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p2.domain.com;lr>
Record-Route: <sip:p1.example.com;lr>
```

u2.domain.com の着信者はこれを受け取り、200 OK でレスポンスする。

```
SIP/2.0 200 OK
Contact: sip:callee@u2.domain.com
Record-Route: <sip:p2.domain.com;lr>
Record-Route: <sip:p1.example.com;lr>
```

u2 の着信者はまた、それのダイアログ状態のリモートターゲット URI に sip:caller@u1.example.com を設定し、それのルートセットに以下の値を設定する。

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

これは P2 によって P1 にそれから u1 に通常どおりフォワードされる。今度は、U1 がそれのダイアログ状態のリモートターゲット URI に sip:callee@u2.domain.com を設定し、それのルートセットに以下の値を設定する。

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

すべてのルートセットエレメントが lr パラメータを含むので、U1 は以下の BYE リクエストを構築する。

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

(プロキシを含む)他のすべてのエレメントがするように、U1 は、リクエストをどこに送るか決定するために、最初の Route ヘッダフィールド値の URI を DNS を使用して解決する。これは P1 に行く。P1 は Request-URI で示されるリソースに対して責任を負わないことに気付くので、それに変更を加えない。P1 は自分が Route ヘッダフィールドの最初の値であることがわかるので、その値を取り除き、リクエストを P2 にフォワード

する。

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

P2 も Request-URI で示されるリソースに対して責任を負わないことに気付くので(P2 は u2.domain.com にではなく domain.com に対して責任を負う)、それに変更を加えない。P2 は最初の Route ヘッダフィールド値に自分自身がいるのに気付くので、それを取り除き、Request-URI に対する DNS ルックアップの結果に基づいて以下のものを u2.domain.com にフォワードする。

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

16.12.1.2 ストリクトルーティングを行うプロキシのトラバース (Traversing a Strict-Routing Proxy)

このシナリオでは、4 つのプロキシを横断してダイアログが確立する。それぞれのプロキシは Record-Route ヘッダフィールド値を追加する。3 つめのプロキシは RFC2543 および多くの検討段階の文書で規定されているストリクトルーティング手順を実装している。

U1->P1->P2->P3->P4->U2

U2 に到着する INVITE は以下のヘッダフィールドを含む

```
INVITE sip:callee@u2.domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p4.domain.com;lr>
Record-Route: <sip:p3.middle.com>
Record-Route: <sip:p2.example.com;lr>
Record-Route: <sip:p1.example.com;lr>
```

U2 はこれに対して 200 OK でレスポンスする。後ほど、U2 は最初の Route ヘッダフィールド値に基づいて以下の BYE リクエストを P4 に送る。

```
BYE sip:caller@u1.example.com SIP/2.0
Route: <sip:p4.domain.com;lr>
Route: <sip:p3.middle.com>
Route: <sip:p2.example.com;lr>
Route: <sip:p1.example.com;lr>
```

P4 は Request-URI で示されるリソースに対して責任を負わないので、これをそのままにしておく。P4 は自分が最初の Route ヘッダフィールド値のエレメントであることに気付くので、それを取り除く。それから、最初の Route ヘッダフィールド値になった sip:p3.middle.com に基づいてリクエストを送る準備をする。しかし P4 はこの URI が lr パラメータを含んでいないことに気付くので、送る前にリクエストを以下のようにリフォームする。

```
BYE sip:p3.middle.com SIP/2.0
Route: <sip:p2.example.com;lr>
Route: <sip:p1.example.com;lr>
Route: <sip:caller@u1.example.com>
```

P3 はストリクトルータなので、以下のものを P2 にフォワードする。

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

P2 は、request-URI が P2 が Record-Route ヘッダフィールドに置いた値であることに気付くので、更に処理を進める前に以下のようにリクエストを書き換える。

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

P2 は u1.example.com に対して責任を負わないので、Route ヘッダフィールド値の解決を行った結果に基づいてリクエストを P1 に送る。

P1 は自分が最初の Route ヘッダフィールド値にいることに気付くので、それを取り除き、以下の結果を得る。

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

P1 は u1.example.com に対して責任を負わず、また Route ヘッダフィールドが一つもないので、P1 は Request-URI に基づいてリクエストを u1.example.com にフォワードする。

16.12.1.3 Record-Route ヘッダフィールド値の書き換え (Rewriting Record-Route Header Field Values)

このシナリオでは、U1 と U2 は異なるプライベート名前空間にいる。そして、それらはプロキシ P1 を介してダイアログに入る。プロキシ P1 は名前空間の間のゲートウェイとして動作する。

```
U1->P1->U2
```

U1 は以下のものを送る。

```
INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0
Contact: <sip:caller@u1.leftprivatespace.com>
```

P1 はそれのロケーションサービスを使用し、以下のものを U2 に送る。

```
INVITE sip:callee@rightprivatespace.com SIP/2.0
Contact: <sip:caller@u1.leftprivatespace.com>
```

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

U2 は以下の 200 (OK)を P1 に送り返す。

```
SIP/2.0 200 OK
Contact: <sip:callee@u2.rightprivatespace.com>
Record-Route: <sip:gateway.rightprivatespace.com;lr>
```

P1 は、U1 が有用であると考える値を提供するために、それの Record-Route ヘッダパラメータを書き換えて、以下のものを U1 に送る。

```
SIP/2.0 200 OK
Contact: <sip:callee@u2.rightprivatespace.com>
Record-Route: <sip:gateway.leftprivatespace.com;lr>
```

後ほど、U1 は以下の BYE リクエストを P1 に送る。

```
BYE sip:callee@u2.rightprivatespace.com SIP/2.0
Route: <sip:gateway.leftprivatespace.com;lr>
```

これを P1 は、以下のものとして U2 にフォワードする。

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

１７．トランザクション (Transactions)

SIP はトランザクションプロトコルである。つまり、コンポーネント間の相互作用が、独立したメッセージ交換の連続で起こる。具体的には、SIP トランザクションは一つのリクエストとそのリクエストに対する何らかのレスポンス(ゼロ個以上の暫定レスポンス(provisional response)および一つ以上の最終レスポンス(Final response)を含む)から成る。リクエストが INVITE であったときのトランザクション(INVITE トランザクションとして知られている)の場合には、最終レスポンス(Final response)が 2xx レスポンスでなかった場合に限り、トランザクションは ACK も含む。レスポンスが 2xx だった場合、ACK はそのトランザクションの一部とはみなされない。

> この場合分けの理由は、INVITE に対するすべての 200(OK)レスポンスを UAC に配送することの重要性に根ざす。それらをすべて UAC に配送するために、UAS のみがそれらの再送の責任を負い(13.3.1.4 節参照)、UAC のみがそれらに ACK で同意する責任を負う(13.2.2.4 節参照)。この ACK は UAC によってのみ再送されるので、それは事実上それ自身のトランザクションであるとみなされる。

トランザクションはクライアントの側面とサーバの側面を持つ。クライアントの側面はクライアントトランザクションとして知られており、サーバの側面はサーバトランザクションとして知られている。クライアントトランザクションはリクエストを送り、サーバトランザクションはレスポンスを送る。クライアントトランザクションとサーバトランザクションはいくつのエレメントにでも組み込める論理的な機能である。具体的には、それらはユーザエージェントとステートフルプロキシサーバの中に存在する。4 節の例を考える。

この例では、UAC がクライアントトランザクションを実行し、それのアウトバウンドプロキシがサーバトランザクションを実行する。アウトバウンドプロキシは、インバウンドプロキシ中のサーバトランザクションにリクエストを送る、クライアントトランザクションも実行する。そのプロキシは、今度は、そのリクエストを UAS のサーバトランザクションに送る、クライアントトランザクションも実行する。これは図 4 に示されている。

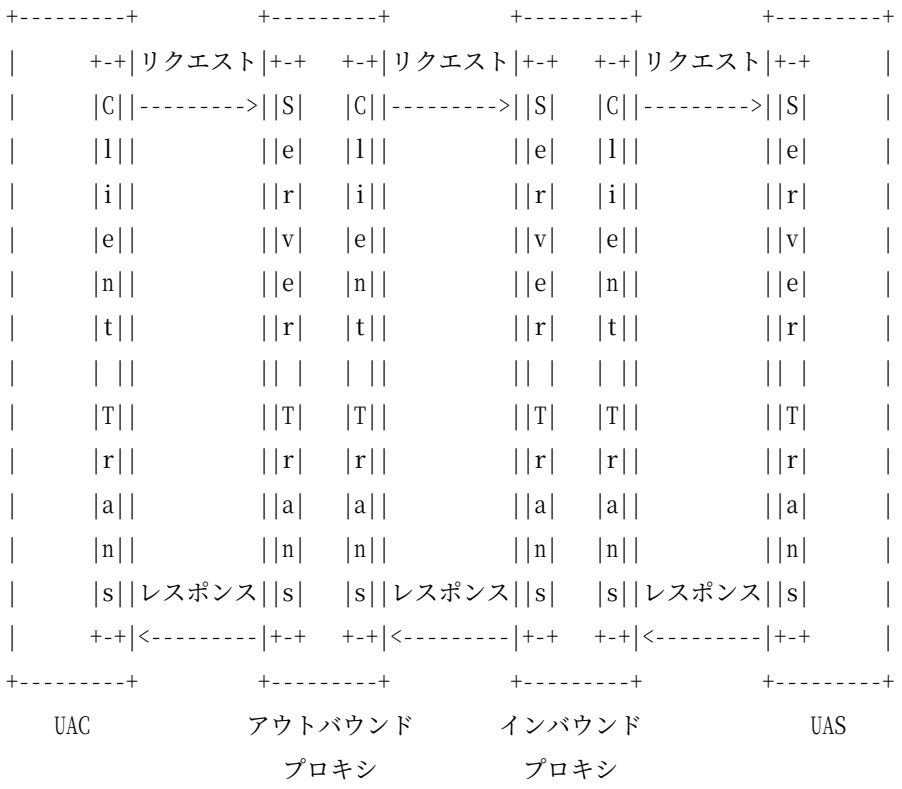

図 4: トランザクションの関係

ステートレスプロキシはクライアントトランザクションまたはサーバトランザクションを含まない。トランザクションは一方の側の UA またはステートフルプロキシともう一方の側の UA またはステートフルプロキシの間に存在する。SIP トランザクションに関して言えば、ステートレスプロキシは実質的にトランスペアレントである。クライアントトランザクションの目的は、クライアントが組み込まれているエレメント(このエレメントをトランザクションユーザまたは TU と呼ぶ。TU は UA またはステートフルプロキシである)からリクエストを受け取り、リクエストをサーバトランザクションに確実に配送することである。

クライアントトランザクションは、レスポンスを受け取って、レスポンスの再送や許可されないレスポンス(たとえば ACK に対するレスポンス)を除去しながらそれを TU に配送する責任も負う。それに加えて、INVITE リクエストの場合には、クライアントトランザクションは、2xx レスポンスを除く（※）すべての最終レスポンス(Final response)に対する ACK リクエストを生成する責任も負う。

[※訳注　：　原文では”accepting”となっているが、”excepting”の誤記と考えられる。]

同様に、サーバトランザクションの目的は、トランスポートレイヤからリクエストを受け取り、それを TU に配送することである。サーバトランザクションはネットワークからすべてのリクエストの再送をフィルタ

ーする。サーバトランザクションは TU からのレスポンスを受け入れ、ネットワーク上で送信するためにトランスポートレイヤにそれを配送する。INVITE トランザクションの場合には、サーバトランザクションは 2xx レスポンスを除くすべての最終レスポンス(Final response)に対する ACK を吸収する。

2xx レスポンスとそれの ACK は特別な扱いを受ける。このレスポンスは UAS のみが再送する。そしてそれの ACK は UAC のみが生成する。このエンド・トゥ・エンドの処置は、呼を受け入れた完全なユーザセットを発信者が知るために必要とされる。この特別なハンドリングのため、2xx の再送はトランザクションレイヤではなく UA コアでハンドリングされる。同様に、2xx に対する ACK の生成は UA コアでハンドリングされる。パス上の各プロキシは、INVITE に対する 2xx レスポンスとそれに対応する ACK を単にフォワードするだけである。

17.1 クライアントトランザクション (Client Transaction)

クライアントトランザクションはステートマシーンを保持することによってその機能を提供する。

TU は単純なインターフェースを介してクライアントトランザクションとコミュニケートする。TU が新規トランザクションの開始を望むときは、クライアントトランザクションを生成し、それに、送る SIP リクエスト、およびそれを送る先の IP アドレス、ポート、トランスポートを渡す。クライアントトランザクションはそれのステートマシーンの実行を開始する。有効なレスポンスはクライアントトランザクションから TU に渡される。

TU によって渡されたリクエストのメソッドによって、クライアントトランザクションのステートマシーンは 2 種類存在する。一つは INVITE リクエストのためのクライアントトランザクションをハンドリングする。このタイプのマシンは INVITE クライアントトランザクションと呼ばれる。もう一つのタイプは INVITE と ACK 以外のすべてのリクエストをハンドリングする。これは非 INVITE クライアントトランザクションと呼ばれる。ACK に対するクライアントトランザクションはない。TU が ACK を送ることを望む場合は、送信するために直接トランスポートレイヤに渡す。

INVITE トランザクションは、その拡張された持続時間のために他のメソッドのトランザクションと異なる。通常、INVITE にレスポンスするために人間による入力が求められる。レスポンスを送るために予期される長い遅延は 3 ウェイハンドシェイクを主張する。その一方、他のメソッドのリクエストは迅速に完了することが予期される。非 INVITE トランザクションは 2 ウェイハンドシェイクに依存するので、TU は非 INVITE リクエストに対して直ちにレスポンスするべきである[SHOULD]。

17.1.1 INVITE クライアントトランザクション (INVITE Client Transaction)

17.1.1.1 INVITE トランザクションの概要 (Overview of INVITE Transaction)

INVITE トランザクションは 3 ウェイハンドシェイクから成る。クライアントトランザクションが INVITE を送り、サーバトランザクションがレスポンスを送る。そしてクライアントトランザクションが ACK を送る。信頼性がないトランスポート(UDP など)では、クライアントトランザクションは、T1 秒で開始し再送のたびに倍になる時間間隔で、リクエストを再送する。T1 はラウンドトリップ時間(RTT)の予測値であり、初期値は 500ms である。ここで述べるほとんどすべてのトランザクションタイマーは T1 に対応し、T1 を変更することでそれらの値が調整される。リクエストは信頼性のあるトランスポート上では再送されない。1xx レスポンスを受け取った後、すべての再送は完全に停止し、クライアントはさらなるレスポンスを待つ。サーバトランザクションは、サーバトランザクションによって信頼性をもって送信されない付加的な 1xx レスポン

スを送ることができる。最終的に、サーバトランザクションは最終レスポンス(Final response)を送ることを決定する。信頼性のないトランスポートでは、そのレスポンスは定期的に再送され、信頼性のあるトランスポートでは、それは一度だけ送られる。クライアントトランザクションで受け取る各最終レスポンス(Final response)に対して、クライアントトランザクションは ACK を送る。その目的は、レスポンスの再送を抑えるためである。

17.1.1.2 **正式な説明** (Formal Description)

INVITE クライアントトランザクションのためのステートマシーンを図 5 に示す。TU が INVITE リクエストで新規クライアントトランザクションを開始するときは、最初のステートである「Calling」に入らなくてはならない[MUST]。クライアントトランザクションはリクエストを送信するためにリクエストをトランスポートレイヤに渡さなくてはならない[MUST](18 節参照)。信頼性がないトランスポートが使用されることになる場合、クライアントトランザクションは、値 T1 でタイマーA を開始しなくてはならない[MUST]。信頼性のあるトランスポートが使用されることになる場合、クライアントトランザクションはタイマーA を開始するべきではない[SHOULD NOT](タイマーA はリクエストの再送を制御する)。いかなるトランスポートにおいても、クライアントトランザクションは、値 64*T1 秒でタイマーB を開始しなくてはならない[MUST](タイマーB はトランザクションのタイムアウトを制御する)。

タイマーA が切れるとき、クライアントトランザクションはリクエストをトランスポートレイヤに渡すことによってそれを再送しなくてはならず[MUST]、値 2*T1 でタイマーをリセットしなくてはならない[MUST]。トランザクションレイヤのコンテキスト内での再送の正式な定義は、以前にトランスポートレイヤに送ったメッセージを取得してそれをトランスポートレイヤにもう一度渡すことである。

タイマーA が 2*T1 秒後に切れるとき、リクエストは再び再送されなくてはならない[MUST](クライアントトランザクションが依然としてこの状態にあると仮定している)。このプロセスは、各送信の後に 2 倍になる間隔でリクエストが再送されるように継続しなくてはならない[MUST]。これらの再送はクライアントトランザクションが「Calling」状態にある間だけ行われるべきである[SHOULD]。

T1 の初期値は 500ms である。T1 は、クライアントトランザクションとサーバトランザクションの間の RTT の推定値である。エレメントは(推奨されてはいないが[NOT RECOMMENDED])インターネット接続を許可しない閉じた、プライベートネットワーク内で、より小さい T1 の値を使用してもよい[MAY]。より大きな値を T1 に選択してもよく[MAY]、これは RTT がより大きいことがあらかじめわかっている場合(たとえば高遅延アクセスリンク上で)が推奨される[RECOMMENDED]。T1 の値がどのようなものであれ、本節で述べられている指数関数的に増加する再送のバックオフが使用されなくてはならない[MUST]。

タイマーB が切れるときにクライアントトランザクションが依然として「Calling」状態にある場合、クライアントトランザクションはタイムアウトが起こったことを TU に通知するべきである[SHOULD]。クライアントトランザクションは ACK を生成してはいけない[MUST NOT]。64*T1 という値は、信頼性のないトランスポートの場合に 7 つのリクエストを送るのに必要な時間に等しい。

クライアントトランザクションが「Calling」状態にある間に暫定レスポンス(provisional response)を受け取る場合、それは「Proceeding」状態に移行する。「Proceeding」状態では、クライアントトランザクションは、これ以上リクエストを再送するべきではない[SHOULD NOT]。さらに、暫定レスポンス(provisional response)は TU に渡されなくてはならない[MUST]。「Proceeding」状態にある間は、これ以上のいかなる暫

定レスポンス(provisional response)も TU に渡されなくてはならない[MUST]。

「Calling」または「Proceeding」状態にあるときは、ステータスコード 300 から 699 までのレスポンスの受信で、クライアントトランザクションを「Completed」に移行させなくてはならない[MUST]。クライアントトランザクションは受け取ったレスポンスを TU に渡さなくてはならず[MUST]、たとえトランスポートに信頼性があったとしてもクライアントトランザクションはACKリクエストを生成して(レスポンスからACKを構築するためのガイドラインは17.1.1.3節で与えられる)それから送信するためにそのACKをトランスポートレイヤに渡さなくてはならない[MUST]。ACK はオリジナルリクエストが送られたのと同じアドレス、ポート、およびトランスポートに送られなくてはならない[MUST]。クライアントトランザクションは「Completed」状態に入ったときに、信頼性のないトランスポートではすくなくとも 32 秒、信頼性のあるトランスポートではゼロ秒の値を持ったタイマーD を開始するべきである[SHOULD]。タイマーD は、信頼性のないトランスポートが使用されたときにサーバトランザクションが「Completed」状態にとどまることができる時間を反映する。これは、初期値が64*T1であるINVITEサーバトランザクションのタイマーHに等しい。しかしながら、クライアントトランザクションはサーバトランザクションが使用している T1 の値を知らないので、タイマーDをT1に基づいて決める代わりに、絶対最小値として32秒が使われる。

「Completed」状態にある間に受け取った最終レスポンス(Final response)のいかなる再送も、ACK を再送するためにトランスポートレイヤに ACK を再び渡すことの原因にならなくてはならない[MUST]が、新しく受け取ったレスポンスは TU に渡してはいけない[MUST NOT]。レスポンスの再送とは、17.1.3 節のルールに基づき、同じクライアントトランザクションにマッチするすべてのレスポンスであると定義されている。

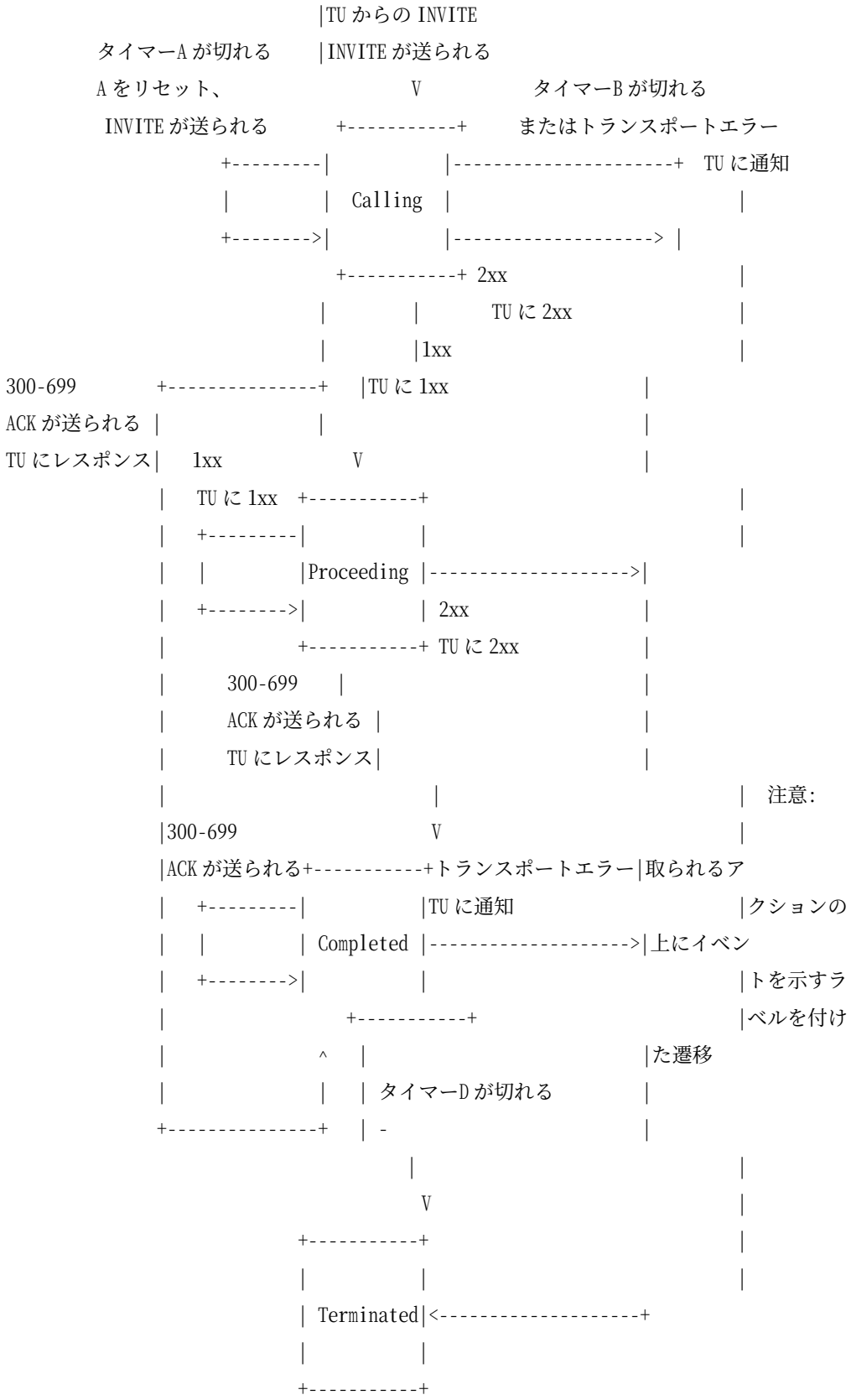

図 5: INVITE クライアントトランザクション

クライアントトランザクションが「Completed」状態にある間にタイマーD が切れる場合、クライアントトランザクションは「Terminated」状態に移らなくてはならない[MUST]。

「Calling」または「Proceeding」状態のいずれかにあるときに、2xx レスポンスを受信することは、クライアントトランザクションが「Terminated」状態に入る原因にならなくてはならず[MUST]、そのレスポンスは TU に渡されなくてはならない[MUST]。このレスポンスのハンドリングは、TU がプロキシコアか UAC コアかに依存する。UAC コアはこのレスポンスに対する ACK の生成をハンドリングし、その一方でプロキシコアは常にアップストリームに 200(OK)をフォワードする。プロキシと UAC の間の 200(OK)処理の違いが、そのハンドリングがトランザクションレイヤで行われないことの理由である。

クライアントトランザクションは、それが「Terminated」状態に入るとすぐに破棄されなくてはならない[MUST]。これは正しい動作を保証するために実際に必要である。その理由は、INVITE に対する 2xx レスポンスが異なる処理を施されるからである。つまり、各 2xx レスポンスはプロキシによってフォワードされ、UAC における ACK のハンドリングは異なる。したがって、各 2xx は(フォワードできるように)プロキシコアに渡される必要があり、また(同意できるように)UAC コアに渡される必要がある。トランザクションレイヤ処理は起こらない。トランスポートによってレスポンスが受け取られるといつでも、トランスポートレイヤが(17.1.3 節のルールを使用して)マッチするクライアントトランザクションを見つけない場合、レスポンスは直接コアに渡される。マッチするクライアントトランザクションは最初の 2xx によって破棄されているので、それ以降の 2xx はマッチするものを見つけられず、よってコアに渡される。

17.1.1.3 ACK リクエストの構築 (Construction of the ACK Request)

本節では、クライアントトランザクション内で送られる ACK リクエストの構築について規定する。2xx に対する ACK を生成する UAC コアは、これではなく、13 節で述べられているルールに従わなくてはならない[MUST]。

クライアントトランザクションによって構築される ACK リクエストは、クライアントトランザクションによってトランスポートに渡されたリクエスト(これをオリジナルリクエストと呼ぶ)に含まれるヘッダフィールドの値と同じ、Call-ID、From、および Request-URI の値を含まなくてはならない[MUST]。ACK の To ヘッダフィールドは承認するレスポンスの To ヘッダフィールドと同じでなくてはならず[MUST]、したがって、tag パラメータが追加されることによりオリジナルリクエストの To ヘッダフィールドとは通常異なることになる。ACK は一つの Via ヘッダフィールドを含まなくてはならず[MUST]、これはオリジナルリクエストの先頭の Via ヘッダフィールドと同じでなくてはならない[MUST]。ACK の CSeq ヘッダフィールドはシーケンス番号にオリジナルリクエストにあったのと同じ値を含まなくてはならない[MUST]が、メソッドパラメータは「ACK」でなくてはならない[MUST]。

それに対するレスポンスが承認されることになる INVITE が Route ヘッダフィールドを持っていた場合、それらのヘッダフィールドは ACK に現れなくてはならない[MUST]。これは、ダウンストリームのどのようなステートレスプロキシを通しても ACK が適切にルーティングされることを保証するためである。

どのようなリクエストでもボディを含むこともよい[MAY]とはいえ、ボディが理解されない場合でもリクエストが拒否されることができないので、ACK 中のボディは特別である。したがって、非 2xx に対する ACK にボディを置くことは推奨されない[NOT RECOMMENDED]が、置く場合には、INVITE に対するレスポンスが 415 でないとすると、ボディタイプは INVITE 中に現れたものに制限される。INVITE に対するレスポンスが 415

であった場合、ACK 中のボディは、415 の Accept ヘッダフィールドにリストされていたタイプのいずれかでもよい[MAY]。

例として、次のリクエストを考える。

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
```

このリクエストへの非 2xx 最終レスポンス(Final response)に対する ACK リクエストは以下のようになるかもしれない。

```
ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 ACK
```

17.1.2 非 INVITE クライアントトランザクション (Non-INVITE Client Transaction)

17.1.2.1 非 INVITE トランザクションの概要 (Overview of the non-INVITE Transaction)

非 INVITE トランザクションは ACK を使用しない。それは単純なリクエストとレスポンスの相互のやりとりである。信頼性のないトランスポートでは、T1 で開始し T2 になるまで倍になる時間間隔で、リクエストは再送される。暫定レスポンス(provisional response)を受け取る場合、信頼性のないトランスポートでは再送が継続するが、時間間隔は T2 で行われる。サーバトランザクションは、リクエストの再送を受け取ったときのみ、それが送った最後のレスポンス(暫定レスポンス(provisional response)または最終レスポンス(Final response)でありうる)を再送する。これが、暫定レスポンス(provisional response)後であってもリクエストの再送を継続する必要がある理由である。それは最終レスポンス(Final response)の確実な配送を保証するためである。

INVITE トランザクションとは異なり、非 INVITE トランザクションは 2xx レスポンスに対する特別なハンドリングがない。その結果、非 INVITE に対するただ一つの 2xx レスポンスが UAC に配送されることになる。

17.1.2.2 正式な説明 (Formal Description)

非 INVITE クライアントトランザクションのためのステートマシーンを図 6 に示す。それは INVITE のためのステートマシーンに非常に似ている。

TU がリクエストで新規クライアントトランザクションを開始するときに「Trying」状態に入る。この状態

に入るとき、クライアントトランザクションは 64*T1 秒後に切れるようにタイマーF をセットするべきである[SHOULD]。リクエストは送信するためにトランスポートレイヤに渡されなくてはならない[MUST]。信頼性のないトランスポートを使用している場合、クライアントトランザクションは T1 秒後に切れるようにタイマーE をセットしなくてはならない[MUST]。まだこの状態にある間にタイマーE が切れる場合、タイマーE はリセットされるが、今回は MIN(2*T1, T2)の値になる。タイマーが再び切れるときは、MIN(4*T1, T2)にリセットされる。この処理は、T2 を上限として指数関数的に増加する時間間隔で再送が起こるように継続する。T2 の初期値は 4 秒であり、非 INVITE サーバトランザクションがリクエストに即座にレスポンスしない場合に、それにレスポンスするために要する時間を表す。T1 と T2 の初期値に対して、これは、500 ミリ秒、1 秒、2 秒、4 秒、4 秒、4 秒、・・・という結果になる。

クライアントトランザクションが依然として「Trying」状態にある間にタイマーF が切れる場合、クライアントトランザクションは TU にタイムアウトを通知するべきであり[SHOULD]、その後「Terminated」状態に入るべきである[SHOULD]。「Trying」状態にある間に暫定レスポンス(provisional response)を受け取る場合、そのレスポンスは TU に渡されなくてはならず[MUST]、その後、そのクライアントトランザクションは「Processing」状態に移行するべきである[SHOULD]。「Trying」状態にある間に最終レスポンス(Final response)(ステータスコード 200 から 699)を受け取る場合、そのレスポンスは TU に渡されなくてはならず[MUST]、クライアントトランザクションは「Completed」状態に移行しなくてはならない[MUST]。

「Proceeding」状態にある間にタイマーE が切れる場合、リクエストは再送のためにトランスポートレイヤに渡されなくてはならず[MUST]、タイマーE は T2 秒でリセットされなくてはならない[MUST]。「Proceeding」状態にある間にタイマーF が切れる場合、TU はタイムアウトを通知されなくてはならず[MUST]、クライアントトランザクションは「Terminated」状態に移行しなくてはならない[MUST]。「Proceeding」状態にある間に最終レスポンス(Final response)(ステータスコード 200 から 699)を受け取る場合、そのレスポンスは TU に渡されなくてはならず[MUST]、クライアントトランザクションは「Completed」状態に移行しなくてはならない[MUST]。

クライアントトランザクションが「Completed」状態に入るとすぐに、それは、信頼性のないトランスポートに対しては T4 秒後に切れるように、信頼性のあるトランスポートに対してはゼロ秒後に切れるようにタイマーK をセットしなくてはならない[MUST]。「Completed」状態は、受け取るかもしれないレスポンスの、すべての追加の再送(これはクライアントトランザクションが信頼性のないトランスポートに対してのみ残ることの理由である)をバッファーするために存在する。T4 は、ネットワークがクライアントトランザクションとサーバトランザクションの間のメッセージをクリアするために要する時間を表す。T4 の初期値は 5 秒である。レスポンスが、17.1.3 節で規定されているルールを使用して同じトランザクションにマッチするとき、それは再送である。この状態にある間にタイマーK が切れる場合、クライアントトランザクションは「Terminated」状態に移行しなくてはならない[MUST]。

トランザクションが「Terminated」状態になったら、それはすぐに破棄されなくてはならない[MUST]。

17.1.3 **レスポンスをクライアントトランザクションにマッチングする** (Matching Responses to Client Transactions)

クライアントのトランスポートレイヤがレスポンスを受け取るとき、17.1.1 節と 17.1.2 節の処理を行うことができるように、それはどのクライアントトランザクションがレスポンスをハンドリングするか決定しなければならない。最初の Via ヘッダフィールドの branch パラメータがこのために使用される。レスポン

スは以下の 2 つの条件の下でクライアントトランザクションにマッチする。

1. レスポンスが、トランザクションを生成したリクエストの最初の Via ヘッダフィールドの branch パラメータと同じ branch パラメータ値を最初の Via ヘッダフィールドに持つ場合。

2. CSeq ヘッダフィールドのメソッドパラメータが、トランザクションを生成したリクエストのメソッドにマッチする場合。CANCEL リクエストは別のトランザクションを構成するが、同じ branch パラメータ値を共有するので、メソッドは必要になる。

リクエストがマルチキャストで送られる場合、異なるサーバによって複数のレスポンスを生成することがあり得る。これらのレスポンスはすべて最初の Via に同じ branch パラメータを持つが、To tag は異なる。上記のルールに基づいて最初に受け取ったレスポンスが使用され、他のものは再送とみなされる。それはエラーではない。マルチキャスト SIP は、一つのレスポンスを処理することに限定される、初歩的な「単一ホップ発見のような(single-hop-discovery-like)」サービスのみを提供する。詳細については 18.1.1 節参照のこと。

17.1.4 **トランスポートエラーのハンドリング** (Handling Transport Errors)

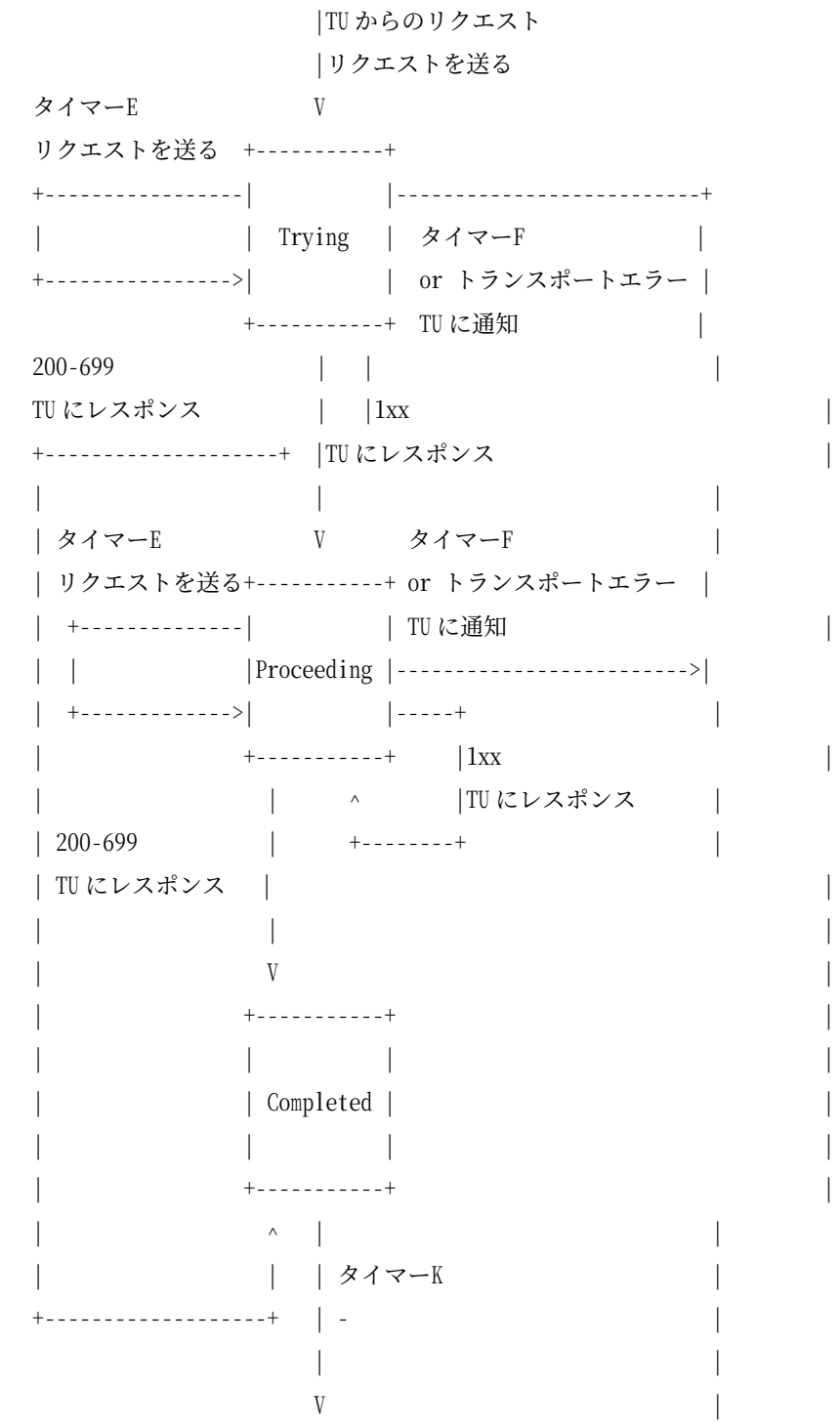

図 6: 非 INVITE クライアントトランザクション

クライアントトランザクションがリクエストを送るためにそれをトランスポートレイヤに送るときに、トランスポートレイヤが失敗を示す場合、以下の手順にしたがう。

クライアントトランザクションはトランスポートの失敗が起こったことを TU に通知するべきであり[SHOULD]、クライアントトランザクションは直接「Terminated」状態に遷移するべきである[SHOULD]。TU は参考文献[4]で述べられているフェールオーバーメカニズムをハンドリングする。

17.2 サーバトランザクション (Server Transaction)

サーバトランザクションは、TU にリクエストを配送することに対して、およびレスポンスの確実な送信に対しての責任を負う。サーバトランザクションはステートマシーンを介してこれを実現する。サーバトランザクションはリクエストを受け取ったときにコアによって生成され、そのリクエストに対するトランザクションハンドリングが求められる(常にこうであるとは限らない)。

クライアントトランザクションと同様に、ステートマシーンは受け取ったリクエストが INVITE かどうかに依存する。

17.2.1 INVITE サーバトランザクション (INVITE Server Transaction)

INVITE サーバトランザクションの状態図を図 7 に示す。

サーバトランザクションがリクエストのために構築されるとき、それは「Proceeding」状態に入る。サーバトランザクションは、TU が 200 ミリ秒以内に暫定レスポンス(provisional response)または最終レスポンス(Final response)を生成するということを知らない限り、100(Trying)レスポンスを生成しなくてはならない[MUST]。知っている場合は、100(Trying)レスポンスを生成してもよい[MAY]。この暫定レスポンス(provisional response)は、ネットワーク混雑を避けるために、リクエストの再送を迅速に抑えるために必要になる。100(Trying)レスポンスは、To ヘッダフィールドの tag の挿入が(リクエストに一つも存在しないときに)「してもよい[MAY]」から「するべきではない[SHOULD NOT]」にダウングレードされたことを除いて、8.2.6 節の手順にしたがって構築される。リクエストは TU に渡されなくてはならない[MUST]。

TU は暫定レスポンス(provisional response)をいくつでもサーバトランザクションに渡す。サーバトランザクションが「Proceeding」状態にある間は、これらは、送信するためにトランスポートレイヤに渡されなくてはならない[MUST]。それらはトランザクションレイヤによって信頼性を持って送られず(それらはトランザクションレイヤによって再送されない)、サーバトランザクションの状態を変更する原因にもならない。「Proceeding」状態にある間にリクエストの再送を受け取る場合、TU から受け取った最新の暫定レスポンス(provisional response)は再送するためにトランスポートレイヤに渡されなくてはならない[MUST]。17.2.3 節のルールに基づいて同じサーバトランザクションにマッチするリクエストは再送である。

「Proceeding」状態にある間に、TU がサーバトランザクションに 2xx レスポンスを渡す場合、サーバトランザクションはこのレスポンスを送信するためにトランスポートレイヤに渡さなくてはならない[MUST]。それはサーバトランザクションによって再送されない。2xx の再送は TU によってハンドリングされる。サーバトランザクションは、次いで「Terminated」状態に移行しなくてはならない[MUST]。

「Proceeding」状態にある間に、TU がサーバトランザクションに 300 から 699 までのステータスコードで

レスポンスを渡す場合、そのレスポンスは、送信するためにトランスポートレイヤに渡されなくてはならず[MUST]、ステートマシーンは「Completed」状態に入らなくてはならない[MUST]。信頼性のないトランスポートでは、T1秒で切れるようにタイマーGがセットされ、信頼性のあるトランスポートではセットされない。

これは、信頼性のあるトランスポートでも常にレスポンスが再送されていた、RFC2543からの変更である。

「Completed」状態に入ったとき、すべてのトランスポートにおいてタイマーHが64*T1秒で切れるようにセットされなくてはならない[MUST]。タイマーHは、サーバトランザクションがいつレスポンスの再送を止めるかを決定する。それの値は、クライアントトランザクションがリクエスト送信のリトライを継続する時間であるタイマーBと同じになるように選択される。タイマーGが切れる場合、レスポンスはもう一度再送のためにトランスポートレイヤに渡され、タイマーGはMIN(2*T1, T2)秒で切れるようにセットされる。それ以降、タイマーGが切れるときは、レスポンスは、送信するために再びトランスポートに渡され、タイマーGはT2を超えない限り倍の値でリセットされる。超える場合は、T2でリセットされる。これは、非INVITEクライアントトランザクションの「Trying」状態におけるリクエストの再送動作と同じである。さらに、「Completed」状態にある間にリクエストの再送を受け取る場合、サーバはレスポンスを再送するためにトランスポートに渡すべきである[SHOULD]。

サーバトランザクションが「Completed」状態にある間にACKを受け取る場合、サーバトランザクションは「Confirmed」状態に移行しなくてはならない[MUST]。この状態ではタイマーGが無視されるので、レスポンスのすべての再送は停止する。

「Completed」状態にある間にタイマーHが切れる場合は、ACKがいつまでも受け取れなかったことを示す。この場合、サーバトランザクションは「Terminated」状態に移行しなくてはならず[MUST]、トランザクションの失敗が起こったことをTUに通知しなくてはならない[MUST]。

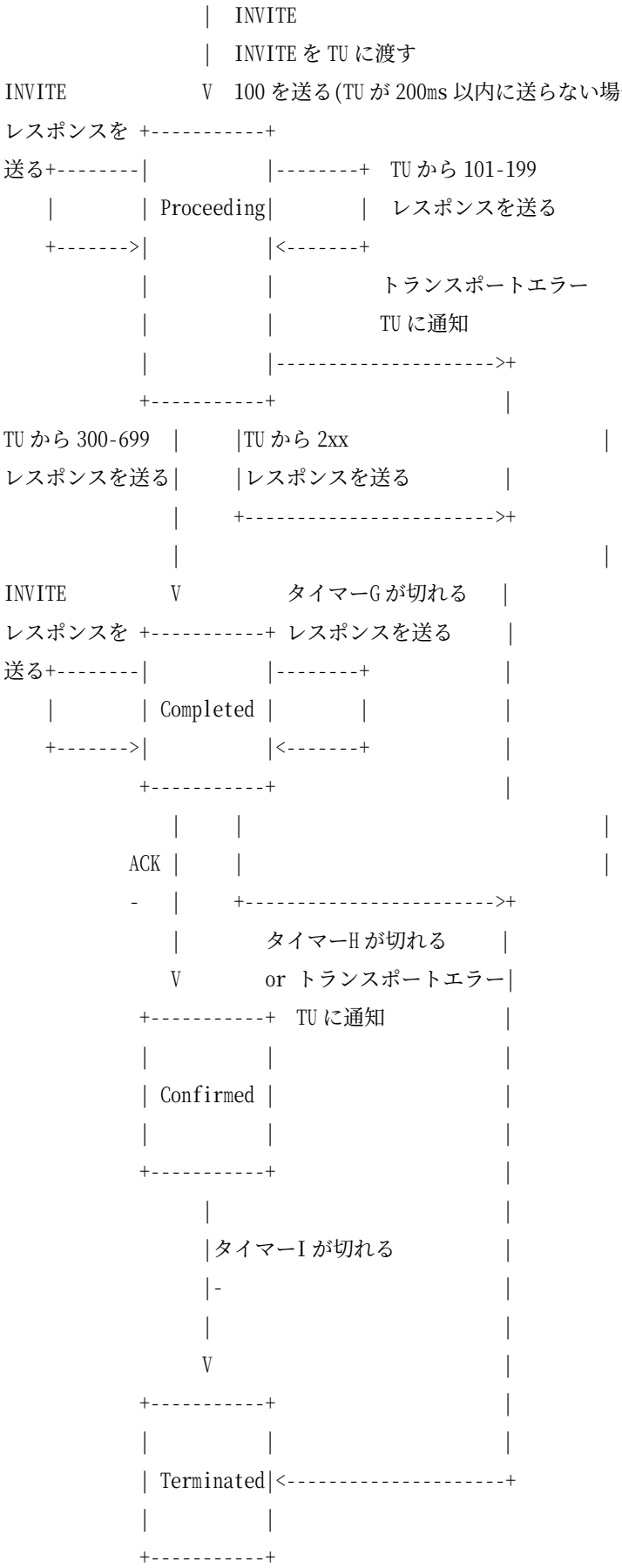

図 7: INVITE サーバトランザクション

「Confirmed」状態の目的は、最終レスポンス(Final response)の再送がトリガーとなって到着したすべての追加の ACK メッセージを吸収することである。この状態に入るとき、タイマーI が、信頼性のないトランスポートでは T4 秒で切れるように、信頼性のあるトランスポートではゼロ秒で切れるようにセットされる。タイマーI が切れるとすぐに、サーバは「Terminated」状態に移行しなくてはならない[MUST]。

トランザクションは、「Terminated」状態に入るとすぐに破棄されなくてはならない[MUST]。クライアントトランザクションと同様に、これは INVITE に対する 2xx レスポンスの信頼性を保証するために必要とされる。

17.2.2 非 INVITE サーバトランザクション (Non-INVITE Server Transaction)

非 INVITE サーバトランザクションのためのステートマシーンを図 8 に示す。

このステートマシーンは「Trying」状態の中で初期化され、初期化されるときに INVITE と ACK 以外のリクエストを渡される。このリクエストは TU に渡される。「Trying」状態に入るとすぐに、それ以降のどのリクエストの再送も捨てられる。17.2.3 節で規定されるルールを使用して同じサーバトランザクションにマッチするリクエストは再送である。

「Trying」状態にある間に、TU がサーバトランザクションに暫定レスポンス(provisional response)を渡す場合、サーバトランザクションは「Proceeding」状態に入らなくてはならない[MUST]。そのレスポンスは、送信するためにトランスポートレイヤに渡されなくてはならない[MUST]。「Proceeding」状態にある間に TU から受け取るそれ以降のすべての暫定レスポンス(provisional response)は、送信するためにトランスポートレイヤに渡されなくてはならない[MUST]。「Proceeding」状態にある間にリクエストの再送を受け取る場合、送られた最新の暫定レスポンス(provisional response)が、再送するためにトランスポートレイヤに渡されなくてはならない[MUST]。「Proceeding」状態にある間に、TU が最終レスポンス(Final response)(ステータスコード 200～699)をサーバに渡す場合、トランザクションは「Completed」状態に入らなくてはならず[MUST]、そのレスポンスは、送信するためにトランスポートレイヤに渡されなくてはならない[MUST]。

サーバトランザクションが「Completed」状態に入るときは、タイマーJ を、信頼性のないトランスポートでは 64*T1 秒で切れるように、信頼性のあるトランスポートではゼロ秒で切れるようにセットしなくてはならない[MUST]。「Completed」状態にある間に、サーバトランザクションは、リクエストの再送を受け取るといつでも、最終レスポンス(Final response)を、再送するためにトランスポートレイヤに渡さなくてはならない[MUST]。TU がサーバトランザクションに渡す他のすべての最終レスポンス(Final response)は、「Completed」状態にある間に捨てられなくてはならない[MUST]。サーバトランザクションはタイマーJ が切れるまでこの状態に留まる。タイマーJ が切れる時点で、サーバトランザクションは「Terminated」状態に移行しなくてはならない[MUST]。

サーバトランザクションは、「Terminated」状態に入るとすぐに破棄されなくてはならない[MUST]。

17.2.3 リクエストをサーバトランザクションにマッチングする (Matching Requests to Server Transactions)

サーバがネットワークからリクエストを受け取るとき、リクエストは既存のトランザクションにマッチされなければならない。これは以下の方法で実現される。

リクエストにおいて最初の Via ヘッダフィールドの branch パラメータが検査される。それが存在し、「z9hG4bK」というマジッククッキーで始まる場合、そのリクエストはこの仕様に準拠するクライアントトランザクションによって生成された。したがって、branch パラメータはそのクライアントが送ったすべてのトランザクションに渡って一意である。以下の場合、リクエストはトランザクションにマッチする。

1. リクエスト中のその branch パラメータが、トランザクションを生成したリクエストにおいて最初の Via ヘッダフィールドのものと同じであり、かつ、

2. リクエストの最初の Via にある sent-by の値が、トランザクションを生 成したリクエストのものと同じであり、かつ、

3. トランザクションを生成したリクエストのメソッドが INVITE となる ACK を除いて、リクエストのメソッドがトランザクションを生成したものとマッチする。

このマッチングルールは、INVITE トランザクションと非 INVITE トランザクションの双方に同じように適用される。

sent-by 値はマッチング処理の一部として使用される。なぜならば、異なるクライアントからの偶然あるいは悪意を持った branch パラメータの重複がおこりうるからである。

最初の Via ヘッダフィールドに branch パラメータが存在しない場合、またはマジッククッキーを含まない場合は、以下の手順が使用される。これは RFC2543 準拠の実装との下位互換をハンドリングするためにある。

INVITE リクエストは、Request-URI、To tag、From tag、Call-ID、CSeq、および最初の Via ヘッダフィールドがトランザクションを生成した INVITE リクエストのものとマッチする場合に、トランザクションにマッチする。この場合、その INVITE はトランザクションを生成したオリジナルの INVITE の再送である。ACK リクエストは、Request-URI、From tag、Call-ID、CSeq 番号(メソッドではない)、および最初の Via ヘッダフィールドがトランザクションを生成した INVITE のものとマッチし、ACK の To tag がサーバトランザクションが送ったレスポンスの To tag にマッチする場合に、トランザクションにマッチする。マッチングはそれら各々のヘッダフィールドに対して定義されているマッチングルールに基づいて行われる。ACK のマッチング処理に To ヘッダフィールドの tag を含めることは、プロキシで 2xx に対する ACK を他のレスポンスに対する ACK と明確に区別する役に立つ。そのプロキシは双方のレスポンスをフォワードしているかもしれない(これは特異の状態で起こる。具体的には、プロキシがリクエストをフォークしてそれからクラッシュしたとき、レスポンスは別のプロキシに配送されるかも知れず、そのプロキシは複数のレスポンスをアップストリームにフォワードすることになる)。以前の ACK にマッチした INVITE トランザクションにマッチする ACK リクエストは、その(以前の)ACK の再送とみなされる。

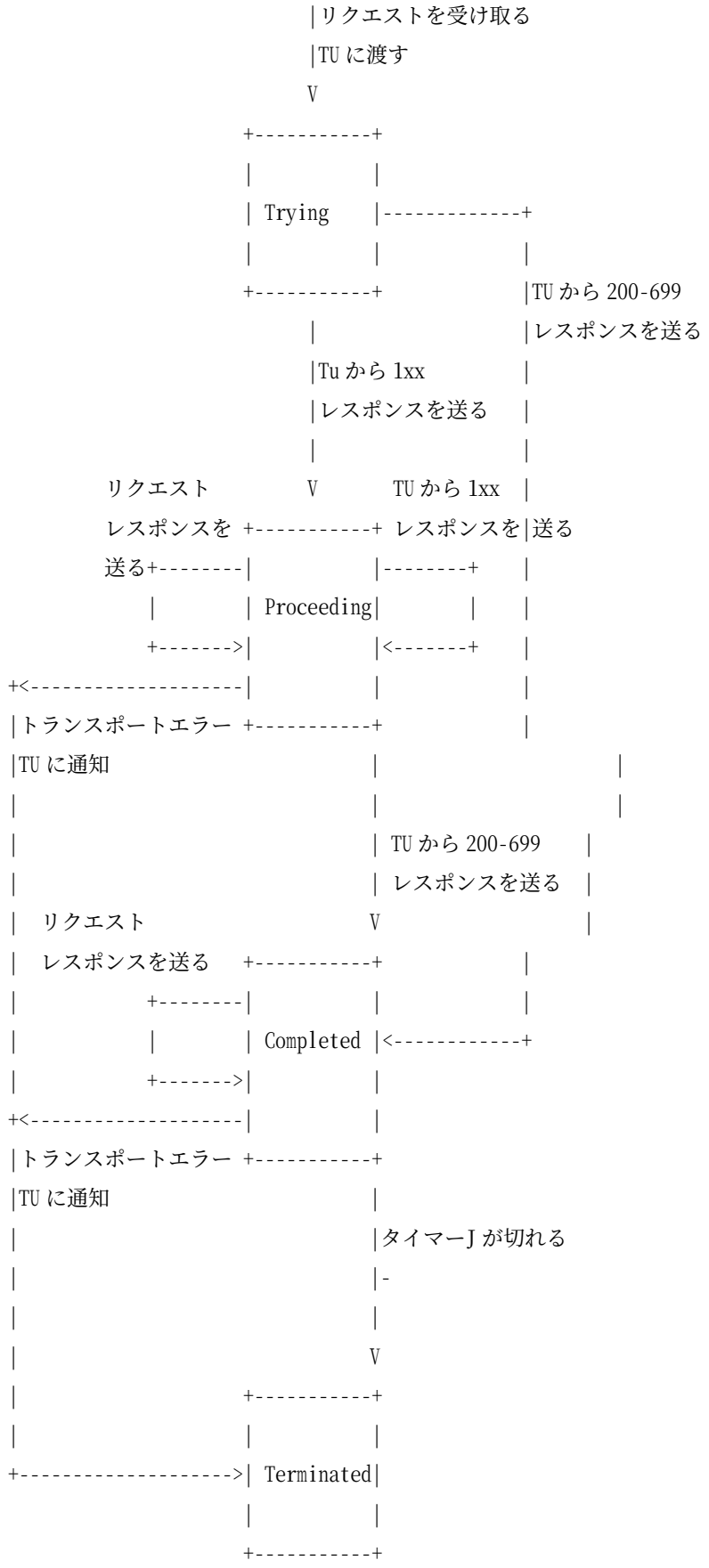

図 8: 非 INVITE サーバトランザクション

他のすべてのリクエストメソッドでは、Request-URI、To tag、From tag、Call-ID、Cseq(メソッドを含む)、および最初の Via ヘッダフィールドがトランザクションを生成したリクエストのものとマッチする場合に、リクエストはトランザクションにマッチする。マッチングは、それら各々のヘッダフィールドに対して定義されているマッチングルールに基づいて行われる。非 INVITE リクエストが既存のトランザクションにマッチするとき、それはそのトランザクションを生成したリクエストの再送である。

マッチングルールは Request-URI を含むので、サーバはレスポンスをトランザクションにマッチすることができない。TU がレスポンスをサーバトランザクションに渡すときは、そのレスポンスのターゲットにされた特定のサーバトランザクションに渡さなければならない。

17.2.4 **トランスポートエラーのハンドリング** (Handling Transport Errors)

サーバトランザクションがレスポンスを送るためにそれをトランスポートレイヤに送るときに、トランスポートレイヤが失敗を示す場合、以下の手順にしたがう。

最初に、レスポンスをバックアップに配送することを試みる参考文献[4]の手順に従う。参考文献[4]の失敗の定義に基づいて、万一それらがすべて失敗する場合、サーバトランザクションは失敗が起こったことを TU に通知するべきであり[SHOULD]、次いで「Terminated」状態に移行するべきである[SHOULD]。

１８．トランスポート (Transport)

トランスポートレイヤは、ネットワークトランスポート上でのリクエストとレスポンスの実際の送信に責任を負う。これは、コネクション指向のトランスポートの場合には、リクエストやレスポンスに使用するコネクションの決定を含む。

トランスポートレイヤは、TCP や SCTP、あるいはそれらより上の TLS のようなトランスポートプロトコルのための持続するコネクションを管理する責任を負う(トランスポートレイヤに対してオープンされたコネクションを含む)。これは、クライアントトランスポートやサーバトランスポートによって開かれたコネクションを含み、そのためコネクションはクライアントとサーバのトランスポート機能の間で共有される。これらのコネクションは、コネクションの遠端のアドレス、ポート、トランスポートプロトコルから形成されるタプルによってインデックス付けされる。トランスポートレイヤによってコネクションが開かれるとき、このインデックスがデスティネーションの IP、ポート、トランスポートに設定される。トランスポートレイヤがコネクションを受け入れるとき、このインデックスはソースの IP アドレス、ポート番号、トランスポートに設定される。ソースポートは短命(ephemeral)であることが多いが、実際に短命(ephemeral)であるのか参考文献[4]の手順で選択されたのか知ることができないので、トランスポートレイヤが受け入れたコネクションはほとんど再利用されることはない、ということに注意すること。その結果、コネクション指向のトランスポートを利用する「ピア」関係にある 2 つのプロキシは、使用中の 2 つのコネクションを多くの場合持つ。それぞれの方向で開始されたトランザクション用のものである。

コネクション上で最後のメッセージを送るか受け取ったあとも、実装で定義されたある時間の間、そのコネクションは開かれたままであることが推奨される[RECOMMENDED]。この期間は少なくとも、トランザクションを「instantiation」から「Terminated」状態に至らせるためにエレメントが必要とする時間の最大値と同じであるべきである[SHOULD]。これは、トランザクションが開始されたのと同じコネクション上で完了される可能性を高めるためである(たとえば、リクエスト、レスポンス、それと INVITE の場合には非 2xx レ

スポンスに対する ACK)。これは通常、少なくとも 64*T1 である(T1 の定義は 17.1.1.1 節参照)。しかしながら、たとえばタイマーC(16.6 節の項目 11)に大きな値を使用しているエレメントでは、この値はより大きくなるかもしれない。

すべての SIP エレメントは UDP と TCP を実装しなくてはならない[MUST]。SIP エレメントはその他のプロトコルを実装してもよい[MAY]。

UA に対して TCP を義務化することは RFC2543 からの大きな変更である。これは、以下で議論されるように、TCP を使わなくてはならない[MUST]大きなメッセージをハンドリングするための必要性から生じた。したがって、たとえエレメントが大きなメッセージを送ることがないとしても、それを受け取ることはあるかもしれず、それをハンドリングできる必要があるかもしれない。

18.1 クライアント (Clients)

18.1.1 リクエストの送信 (Sending Requests)

トランスポートレイヤのクライアント側はリクエストを送ることとレスポンスを受け取ることに対して責任を負う。トランスポートレイヤのユーザはクライアントトランスポートに、リクエスト、IP アドレス、ポート、トランスポートと、おそらくはマルチキャストデスティネーションのための TTL を渡す。

リクエストがあと 200 バイトかそれ未満でパスの MTU のサイズに達する場合、またはリクエストが 1300 バイトより大きくてパスの MTU が未知の場合、リクエストは、例えば TCP のような、RFC2914 [43]のコンジェスチョン制御プロトコルを使用して送られなくてはならない[MUST]。これによって先頭の Via で示したものからトランスポートプロトコルが変わる場合は、先頭の Via の値は変更されなくてはならない[MUST]。これは UDP 上でメッセージのフラグメンテーションを防ぎ、大きなメッセージのためのコンジェスチョン制御を提供する。しかしながら、実装はデータグラムの最大パケットサイズまでのメッセージをハンドリングできなくてはならない[MUST]。UDP においては、IP と UDP のヘッダサイズを含めて、このサイズは 65,535 バイトである。

メッセージサイズと MTU 間の 200 バイトの「バッファ」は、SIP のレスポンスがリクエストよりも大きくなることがあるという事実に対応する。これはたとえば INVITE に対するレスポンスへの Record-Route ヘッダフィールド値の追加のために起こる。余分なバッファのために、レスポンスはリクエストよりもおよそ 170 バイト大きくなることができ、それでも IPv4 でフラグメント化しない(約 30 バイトは IPSec がないとして、IP/UDP で消費される)。MTU が未知の時には、イーサネットの MTU が 1500 バイトであるという仮定に基づいて、1300 が選択される。

エレメントがこれらのメッセージサイズ制限のために TCP 上でリクエストを送り、そのリクエストが UDP 上で送られた場合に、コネクションを確立する試みが ICMP Protocol Not Supported を生成するか TCP のリセットを招く結果になるなら、エレメントは UDP を使ってリクエストを再試行するべきである[SHOULD]。これは、TCP をサポートしない RFC2543 準拠の実装との下位互換を提供するためだけである。この仕様の将来の改定において、この動作は反対されることが予想される。

マルチキャストアドレスにリクエストを送るクライアントは、デスティネーションのマルチキャストアドレスを含む maddr パラメータを Via ヘッダフィールド値に追加しなくてはならず[MUST]、IPv4 では、値 1 の ttl パラメータを追加するべきである[SHOULD]。IPv6 のマルチキャストの用法はこの仕様では定義されてお

らず、必要性が生じたときに将来の標準化の対象になるだろう。

これらのルールはSIPにおけるマルチキャストのしっかりとした目的のある制限をもたらす。それの主要な働きは、リクエストを同種(homogeneous)サーバのグループに配送して「単一ホップ発見のような(single-hop-discovery-like)」サービスを提供することである(いずれか一つのサーバからのレスポンスを処理することだけが必要とされる)。この機能は登録に対して最も有用である。実際、17.1.3節のトランザクション処理に基づいて、クライアントトランザクションは最初のレスポンスを受け入れ、その他のすべては同じViaのbranch識別子を含むので、それらを再送とみなす。

リクエストが送られる前に、クライアントトランスポートはViaヘッダフィールドにsent-byフィールドの値を挿入しなくてはならない[MUST]。このフィールドはIPアドレスまたはホスト名、およびポートを含む。FQDNの使用が推奨される[RECOMMENDED]。このフィールドは以下に述べられる特定の状況下でレスポンスを送るために使用される。ポートがない場合の初期値はトランスポートに依存する。UDP、TCP、SCTPでは5060、TLSでは5061である。

信頼性のあるトランスポートでは、レスポンスは通常、リクエストを受け取ったコネクション上で送られる。したがって、クライアントトランスポートはリクエストを送るために使用したのと同じコネクション上でレスポンスを受け取ることに備えなくてはならない[MUST]。エラー状況下では、サーバは、レスポンスを送るために新たなコネクションのオープンを試みるかもしれない。このケースをハンドリングするために、トランスポートレイヤは、リクエストを送ったソースIPアドレスとsent-byフィールド中のポート番号上でやってくるコネクションを受け取ることにも備えなくてはならない[MUST]。それはまた、参考文献[4]の5節で述べられている手順に基づいてサーバが選択するいかなるアドレスとポート上でやってくるコネクションでも受け取る用意をしていなくてはならない[MUST]。

信頼性のないユニキャストトランスポートでは、リクエストを送ったソースIPアドレス(レスポンスはソースアドレスに送り返されるので)とsent-byフィールドのポート番号でレスポンスを受け取るための用意を、クライアントトランスポートはしなくてはならない[MUST]。さらに、信頼性のあるトランスポートと同様に、特定の場合にはレスポンスは別の場所に送られる。クライアントは、参考文献[4]の5節で述べられている手順に基づいてサーバが選択するいかなるアドレスとポート上で受け取るレスポンスでも受け取る用意をしていなくてはならない[MUST]。

マルチキャストでは、リクエストが送られた先と同じマルチキャストグループとポートでレスポンスを受け取る用意を、クライアントトランスポートはしなくてはならない[MUST](すなわち、それはリクエストを送ったマルチキャストグループのメンバーになる必要がある)。

リクエストが、既存のコネクションがオープンしているあるIPアドレス、ポート、およびトランスポートに向けられた場合、そのリクエストを送るためにこのコネクションを使用することが推奨する[RECOMMENDED]が、別のコネクションをオープンして使用してもよい[MAY]。

リクエストがマルチキャストを使用して送られる場合、それはそのトランスポートユーザによって提供されるグループアドレス、ポート、およびTTLに送られる。リクエストがユニキャストの信頼性のないトランスポートを使用して送られる場合、それはそのトランスポートユーザによって提供されるIPアドレスおよびポートに送られる。

18.1.2 **レスポンスの受信** (Receiving Responses)

レスポンスを受け取るとき、クライアントトランスポートは最初の Via ヘッダフィールド値を検査する。そのヘッダフィールド値の sent-by パラメータの値が、リクエストに挿入するためにクライアントトランスポートに設定されている値に対応しない場合、そのレスポンスはだまって捨てられなくてはならない[MUST]。

存在するクライアントトランザクションがあれば、クライアントトランスポートはそのレスポンスを既存のトランザクションにマッチさせることを試みるために、17.1.3 節のマッチング手順を使用する。マッチするものがある場合、そのレスポンスはそのトランザクションに渡されなくてはならない[MUST]。そうでなければ、そのレスポンスは更なる処理のために(それがステートレスプロキシ、ステートフルプロキシ、あるいは UA であるか否かにかかわらず)コアに渡されなくてはならない[MUST]。これらの「コースから外れた」レスポンスのハンドリングは、コアに依存する(たとえば、プロキシがそれらをフォワードするのに対して、UA はそれを捨てる)。

18.2 **サーバ**(Servers)

18.2.1 **リクエストの受信** (Receiving Requests)

サーバは、そのサーバと通信する目的で「渡される」SIP URI または SIPS URI(参考文献[4])の DNS ルックアップの結果になる可能性があるどのような IP アドレス、ポート、トランスポートの組み合わせのリクエストを受け取ることにも備えるべきである[SHOULD]。このコンテキストにおいて、「渡される」ということには、ある REGISTER リクエストまたはリダイレクトレスポンスの Contact ヘッダフィールドに URI を置くこと、またはリクエストやレスポンスの Record-Route ヘッダフィールドに URI を置くことも含まれる。URI はまた、それを Web ページや名刺に掲載することによっても「渡す」ことができる。サーバはすべての公開インターフェースのデフォルト SIP ポート(TCP と UDP では 5060、TCP 上の TLS では 5061)上のリクエストを待ち受けすることが推奨される[RECOMMENDED]。典型的な例外はプライベートネットワークのときや、同一ホスト上で複数のサーバインスタンスが走っているときである。サーバが UDP のために待ち受けするいかなるポートとインターフェースも、サーバは TCP に対してもそれと同じポート、インターフェースを待ち受けしなくてはならない[MUST]。これは、メッセージが大きすぎる場合は、メッセージは UDP ではなくて TCP で送られる必要があるかもしれないからである。結果として、その逆は正しくない。サーバは、それが TCP のために特定のアドレスとポートを待ち受け中というだけで、UDP のためにそれと同じアドレスとポートを待ち受けする必要はない。もちろんその他にサーバが特定のアドレスとポートを待ち受けする必要がある理由があるかもしれない。

サーバトランスポートが何らかのトランスポート上でリクエストを受け取るとき、それは先頭にある Via ヘッダフィールド値の sent-by パラメータの値を検査しなくてはならない[MUST]。sent-by パラメータのホスト部分がドメイン名を含む場合、あるいはそれがパケットのソースアドレスと異なる IP アドレスを含む場合、サーバはその Via ヘッダフィールド値に received パラメータを追加しなくてはならない[MUST]。このパラメータはソースアドレス(このソースアドレスからパケットを受け取った)を含まなくてはならない[MUST]。これは、レスポンスがリクエストがやって来たソース IP アドレスに送られなければならないので、サーバのトランスポートレイヤがレスポンスを送るのを支援するためである。

サーバトランスポートが受け取ったリクエストの、以下のような部分を考えてみる。

```
INVITE sip:bob@Biloxi.com SIP/2.0
```

Via: SIP/2.0/UDP bobspc.biloxi.com:5060

リクエストは 192.0.2.4 というソース IP アドレスと共に受け取られた。リクエストを上に渡す前に、トランスポートは、received パラメータを追加する。そのため、リクエストの一部は以下のようになるだろう。

INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;received=192.0.2.4

次に、サーバトランスポートはリクエストをサーバトランザクションにマッチしようとする。それは 17.2.3 節に述べられているマッチングルールを使用してそれを行う。マッチするサーバトランザクションが見つかる場合、リクエストは、処理のためにそのトランザクションに渡される。マッチするものが見つからない場合、リクエストはコアに渡される。コアはそのリクエストのための新規サーバトランザクションを構築することを決定するかもしれない。UAS コアが INVITE に対して 2xx レスポンスを送るとき、サーバトランザクションは破棄されるということに注意すること。これは、ACK が到着するときに、マッチするサーバトランザクションが一つもないだろうことを意味し、このルールに基づいて ACK は UAS コアに渡されてそこで処理されることを意味する。

18.2.2 **レスポンスの送信** (Sending Responses)

サーバトランスポートはレスポンスをどこに送るか決定するために、最初の Via ヘッダフィールドの値を使用する。それは以下の手順に従わなくてはならない[MUST]。

- sent-protocol が TCP や SCTP、あるいはそれらよりも上の TLS のような信頼性のあるプロトコルの場合、レスポンスは、そのトランザクションを生成したオリジナルリクエストの発信元への既存のコネクションを使用して(そのコネクションがまだオープンしていれば)送られなくてはならない[MUST]。これは、サーバトランスポートがサーバトランザクションとトランスポートコネクションの間の関係を保持することを要求する。そのコネクションがもはやオープンしていない場合、サーバは、もし存在すれば received パラメータの IP アドレスへのコネクションを、sent-by 値のポートを使用して、またはポートが指定されていなければそのトランスポートのデフォルトポートを使用して、オープンするべきである[SHOULD]。そのコネクションの試みが失敗する場合、コネクションをオープンしてレスポンスを送るための IP アドレスとポートを決定するために、サーバは参考文献[4]のサーバのための手順を使うべきである[SHOULD]。

- そうでなければ、Via ヘッダフィールド値が maddr パラメータを含む場合、sent-by で示されるポート、または存在しなければポート 5060 を使用して、そこにリストされているアドレスにレスポンスがフォワードされなくてはならない[MUST]。アドレスがマルチキャストアドレスの場合、ttl パラメータで示される TTL を使用して、あるいはそのパラメータが存在しないときは TTL を 1 として、レスポンスが送られるべきである[SHOULD]。

- そうでなければ(信頼性のないユニキャストトランスポートでは)、最初の Via が received パラメータを持つ場合、sent-by 値で示されるポート、あるいは明示的に指定されていないときは 5060 を使用して、received パラメータのアドレスにレスポンスが送られなくてはならない[MUST]。これがたとえば ICMP の「port unreachable」レスポンスを引き出すなどして失敗する場合、レスポンスをどこに送るか決定するために、参考文献[4]の 5 節の手順を使うべきである

[SHOULD]。

- そうでなければ、それが receiver に tag 付けされていなかった場合、参考文献[4]の 5 節に述べられている手順を使って、sent-by 値によって示されるアドレスにレスポンスが送られなくてはならない[MUST]。

18.3 フレーム化 (Sending Responses)

メッセージ指向のトランスポート(UDP など)の場合、メッセージが Content-Length ヘッダフィールドを持っているなら、メッセージボディは多くのバイトを含むと想定される。トランスポートパケット中のボディの終わり以降に付加的なバイトがある場合、それらは破棄されなくてはならない[MUST]。トランスポートパケットがメッセージボディが終わる前に終了する場合、これはエラーとみなされる。メッセージがレスポンスである場合、それは破棄されなくてはならない[MUST]。メッセージがリクエストである場合、エレメントは 400(Bad Request)レスポンスを生成するべきである[SHOULD]。メッセージが Content-Length ヘッダフィールドを持たない場合、メッセージボディはトランスポートパケットの終わりで終了すると想定される。

ストリーム指向のトランスポート(TCP など)の場合、Content-Length ヘッダフィールドはボディのサイズを示す。Content-Length ヘッダフィールドは、ストリーム指向のトランスポートとともに使用されなくてはならない[MUST]。

18.4 エラーハンドリング (Error Handling)

エラーハンドリングは、メッセージがリクエストであるかレスポンスであるかに依存しない。

トランスポートユーザが、信頼性のないトランスポート上でメッセージを送ることを要求し、その結果が ICMP エラーの場合、動作は ICMP エラーのタイプに依存する。ホスト、ネットワーク、ポートまたはプロトコルの unreachable エラー、あるいはパラメータに問題があるエラーは、トランスポートレイヤがトランスポートユーザに送信時の失敗を通知する原因になるべきである[SHOULD]。Source quench と TTL exceeded の ICMP エラーは無視するべきである[SHOULD]。

トランスポートユーザが信頼性のあるトランスポート上でリクエストを送ることを要求し、その結果がコネクションの失敗である場合、トランスポートレイヤはトランスポートユーザに送信時の失敗を通知するべきである[SHOULD]。

１９．コモンメッセージコンポーネント (Common Message Components)

SIP メッセージ中の様々な場所に現れる(時にはメッセージの外に)、個別に議論する価値がある、SIP メッセージの特定のコンポーネントがある。

19.1 SIP URI と SIPS URI (SIP and SIPS Uniform Resource Indicators)

SIP URI または SIPS URI はコミュニケーションリソースを識別する。すべての URI と同様に、SIP URI と SIPS URI は Web ページ、E メールメッセージ、あるいは印刷物に掲載できる。それらはリソースとコミュニケーションセッションを開始し保持するために十分な情報を含んでいる。

コミュニケーションリソースの例としては以下のものがある。

- オンラインサービスのユーザ
- multi-line 電話上のアピアランス
- メッセージングシステム上のメールボックス
- ゲートウェイサービスにおける PSTN 番号
- 組織内のグループ(たとえば、［sales(セールス)］や「helpdesk(ヘルプデスク)」

SIPS URI はリソースに安全にコンタクトすることを指定する。これは特に、UAC とその URI を所持するドメイン間で TLS が使用されることを意味する。そこからは、ユーザに到達するために、特定のセキュリティメカニズムがそのドメインのポリシーに依存する安全なコミュニケーションが利用される。そのリソースと安全にコミュニケーションすることを望む場合は、SIP URI によって記述されるいかなるリソースも、スキームを変更するだけで SIPS URI に「アップグレード」できる。

19.1.1 SIP URI コンポーネントと SIPS URI コンポーネント(SIP and SIPS URI Components)

「sip:」および「sips:」スキームは RFC2396(参考文献[5])のガイドラインに従う。それらは mailto URL に似た形式を使用し、SIP request-header フィールドと SIP message-body の指定を可能にする。これは、Web ページ上または E-メールメッセージ中で URI を使って開始されたセッションのサブジェクト、メディアタイプ、または緊急度を指定することを可能にする。SIP URI や SIPS URI の正式な構文は 25 節で提示される。SIP URI の場合の一般的な形式は以下のようである。

```
sip:user:password@host:port;uri-parameters?headers
```

SIPS URI のフォーマットは、スキームが sip にかわって sips になることを除いて、これと同じである。これらのトークンおよびいくつかの拡張されたトークンは以下のような意味を持つ。

user:

宛先となる host における特定のリソースの識別子。このコンテキスト中の用語「host」はドメインをさすことが多い。URI の「userinfo」はこの user フィールド、password フィールドおよびそれに続く@から成る。URI の userinfo 部分はオプションであり、デスティネーションホストがユーザという観念を持たないとき、またはホスト自身が識別されるリソースであるときは、省略してもよい[MAY]。SIP URI または SIPS URI に@記号が存在する場合、user フィールドを空にしてはいけない[MUST NOT]。

宛先となるホストが電話番号を処理することが可能な場合、たとえばインターネットテレフォニーゲートウェイの場合、RFC2806(参考文献[9])で定義されている telephone-subscriber フィールドを、user フィールドを埋めるために使用してもよい[MAY]。19.1.2 節で述べられる、SIP URI および SIPS URI で telephone-subscriber フィールドをエンコードするための特別なエスケープのルールがある。

password:

user に関連付けられたパスワード。SIP URI および SIPS URI の構文はこのフィールドの存在を認めるが、それの使用は推奨されない[NOT RECOMMENDED]。なぜなら、平文で認証情報を渡す(URI など)ことは、それが使用されるほとんどすべてのケースでセキュリティリスクになることが証明されているからである。たとえば、このフィールドで PIN ナンバーを送ることは、PIN を暴露することになる。

password フィールドは user 部分の単なる拡張であることに注意すること。そのフィールドの password 部分に特別な意味を与えることを望まない実装は、「user:password]を単に一つの文字列として扱ってもよい[MAY]。

host:

SIP リソースを提供するホスト。host 部分は、FQDN、あるいは数値の IPv4 または IPv6 アドレスを含む。可能な場合は、FQDN 形式を使用することが推奨される[RECOMMENDED]。

port:

リクエストが送られるポート番号。

URI パラメータ:

その URI から構築されるリクエストに影響するパラメータ。

URI parameters は、hostport コンポーネントの後に追加され、セミコロンで区切られる。

URI parameters は次の形式を取る。

```
parameter-name "=" parameter-value
```

URI には任意の個数の URI parameters を含められるとはいえ、どの parameter-name も 2 回以上現れてはいけない[MUST NOT]。

この拡張可能なメカニズムは、transport、maddr、ttl、user、method、および lr パラメータを含む。

transport パラメータは、参考文献[4]で規定されているように、SIP メッセージを送るために使用されるトランスポートメカニズムを決定する。SIP はどのようなネットワークトランスポートプロトコルでも使用できる。パラメータ名は UDP(RFC768 参考文献[14])、TCP(RFC761 参考文献[15])、および SCTP(RFC2960 参考文献[16])のために定義されている。SIPS URI では、transport パラメータは信頼性のあるトランスポートを示さなくてはならない[MUST]。

maddr パラメータは、host フィールドから得られるいかなるアドレスも無視して、このユーザのためにコンタクトするサーバアドレスを示す。maddr パラメータが存在するとき、URI の port および transport コンポーネントは maddr パラメータ値で示されるアドレスに適用される。参考文献[4]は、リクエストを送るために送信先アドレス、ポート、トランスポートを取得するための、transport、maddr、および hostport の適切な解釈について述べている。

maddr フィールドは、ルーズソースルーティングの単純な形式として使用されている。これは、デスティネーションへルーティング途中でトラバースしなければならないプロキシを URI が指定することを可能にする。maddr パラメータをこのように使用しつづけることには強く水を差されている(それを可能にするメカニズムは反対されている)。実装はその代わりに、必要であれば既存のルートセットを制定して(8.1.1.1 節参照)、このドキュメントで述べられている Route メカニズムを使用するべきである。これは、トラバースされるノードを記述するための完全な URI を与える。

ttl パラメータは、UDP マルチキャストパケットの存続時間の値を決定する。ttl パラメータは、maddr がマルチキャストアドレスで、かつトランスポートプロトコルが UDP の場合にのみ使用されなくてはならない[MUST]。たとえば、ttl が 15 で、239.255.255.1 へのマルチキャストを使用して alice@atlanta.com へのコールを指定するには、次の URI が使用されるだろう。

sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15

有効な telephone-subscriber 文字列のセットは、有効な user 文字列のサブセットである。user の URI parameter は、期せずして電話番号に似ているユーザ名と電話番号を区別するために存在する。user 文字列が telephone-subscriber としてフォーマットされた電話番号を含む場合、user パラメータ値「phone」が存在するべきである[SHOULD]。このパラメータがない場合でも、SIP URI および SIPS URI の受信者は、ユーザ名のための名前空間のローカル制限が許可する場合は、@の前の部分を電話番号として解釈してもよい[MAY]。

URI から構築された SIP リクエストのメソッドは、method パラメータで指定できる。

lr パラメータが存在するときは、このリソースに対して責任を負うエレメントがこのドキュメントで規定されているルーティングメカニズムを実装することを示す。このパラメータは、プロキシが Record-Route ヘッダフィールド値に置く URI で使用される。また、既存のルートセットの URI に現れるかもしれない。

このパラメータは、RFC2543 と bis-05 までの rfc2543bis ドラフトのストリクトルーティングメカニズムを実装するシステムとの下位互換を実現するために使用される。このパラメータを含まない URI に基づいてリクエストを送る用意をしているエレメントは、受信側エレメントがストリクトルーティングを実装し、Request-URI の情報を保持するためにメッセージを再形成すると仮定できる。

uri-parameter のメカニズムは拡張可能なので、SIP エレメントは理解できないどのような uri-parameter もだまって無視しなくてはならない[MUST]。

Headers:

URI から構築されるリクエストに含められるヘッダフィールド。

SIP リクエストの Headers フィールドは、「?」メカニズムで URI 中に指定できる。ヘッダ名と値は、アンパサンドで区切られた hname = hvalue の対でエンコードされる。特別な hname である「body」

は、関連付けられた hvalue が SIP リクエストの message-body であることを示す。

表 1 は、URI が現れるコンテキストに基づく SIP URI および SIPS URI コンポーネントの使用方法をまとめたものである。external 列には、SIP メッセージ外のどこか(たとえば、Web ページや名刺の上)に現れる URI を記述する。「m」とマークされたエントリは必須、「o」とマークされたエントリはオプション、「-」とマークされたエントリは許可されていない。URI を処理するエレメントは、許可されていないコンポーネントがあった場合にはそれを無視するべきである[SHOULD]。2 番めの列は、オプションのエレメントの初期値を示す(それが存在しない場合の)。「--」はそのエレメントがオプションでないかあるいは初期値を持たないことを示す。

Contact ヘッダフィールド中の URI は、そのヘッダフィールドが現れるコンテキストに依存した異なる制限を持つ。一つはダイアログを確立し維持するメッセージ(INVITE とそれに対する 200(OK)レスポンス)に適用する。他は登録とリダイレクトメッセージ(REGISTER、それに対する 200(OK)レスポンス、およびあらゆるメソッドに対する 3xx クラスレスポンス)に適用する。

19.1.2 文字エスケープ要求事項 (Character Escaping Requirements)

					dialog		
				reg./redir.	Contact/		
	default	Req.-URI	To	From	Contact	R-R/Route	external
user	--	o	o	o	o	o	o
password	--	o	o	o	o	o	o
host	--	m	m	m	m	m	m
port	(1)	o	-	-	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	-	-	o
maddr-param	--	o	-	-	o	o	o
ttl-param	1	o	-	-	o	-	o
transp.-param	(2)	o	-	-	o	o	o
lr-param	--	o	-	-	-	o	o
other-param	--	o	o	o	o	o	o
headers	--	-	-	-	o	-	o

(1): デフォルトポート値はトランスポートおよびスキームに依存する。UDP、TCP、SCTP を使用する sip:の初期値は 5060 である。TCP 上で TLS を使用する sip:、および TCP 上の sips:の初期値は 5061 である。

(2): デフォルトトランスポートはスキームに依存する。sip:では UDP である。sips:では TCP である。

表 1:SIP ヘッダフィールド値、Request-URI、references のための URI コンポーネントの使用方法と初期値

SIP は、SIP URI 中でエスケープされなければならないキャラクターセットを定義する際に RFC2396(参考文献[5])の要求とガイドラインに従い、エスケープにそれの「"%" HEX HEX」メカニズムを使用する。RFC2396(参考文献[5])によると、

与えられたどのURIコンポーネント内で実際に予約されているキャラクターセットも、そのコンポーネントによって定義される。一般的に、文字がエスケープされたUS-ASCIIエンコーディングで置き換えられるときに(参考文献[5])、URIの意味合い(semantics)が変更される場合、その文字は予約される。スペースや制御文字、URIの区切文字などの、除外されたUS-ASCII文字(RFC2396 参考文献[5])も、エスケープされなくてはならない[MUST]。URIはエスケープされていないスペースと制御文字を含んではいけない[MUST NOT]。

各コンポーネントにおいて、有効なBNFセットの拡張が、厳密にどの文字がエスケープされずに現れるかを定義する。他のすべての文字はエスケープされなくてはならない[MUST]。

たとえば、「@」はuserコンポーネントのキャラクターセットに含まれていないので、ユーザ「j@s0n」は少なくとも@記号を、「j%40s0n」のように、エンコードされなければならない。

25節のhnameトークンとhvalueトークンの拡張は、ヘッダフィールド名とヘッダフィールド値中のURIで予約されているすべての文字がエスケープされなくてはならない[MUST]ことを示す。

ユーザコンポーネントの telephone-subscriber サブセットは、エスケープ時の特別な考慮を要する。telephone-subscriber に関する RFC2806(参考文献[9])の記述で予約されていないキャラクターセットは、さまざまな構文エレメントに、SIP URI で使用されるときにエスケープする必要がある多くの文字を含む。user ルールのための BNF 拡張に現れない、telephone-subscriber 中に出てくるどのような文字も、エスケープされなくてはならない[MUST]。

文字エスケープはSIP URIあるいはSIPS URIのhostコンポーネントでは許可されていないことに注意すること(それの拡張では「%」文字は有効ではない)。これはドメイン名の国際化の要求が完成した時点で将来的に変更される可能性が高い。現在の実装では、hostコンポーネントで受け取ったエスケープ文字をそれのエスケープされていない形とリテラルに等価なものとして処理することで堅牢性を高めることを試みてはいけない[MUST NOT]。IDN(ドメイン名の国際化)の要求に合致するために必要とされる動作は、大きく異なるかもしれない。

19.1.3 SIP URI と SIPS URI の例 (Example SIP and SIPS URIs)

```
sip:alice@atlanta.com
sip:alice:secretword@atlanta.com;transport=tcp
sips:alice@atlanta.com?subject=project%20x&priority=urgent
sip:+1-212-555-1212:1234@gateway.com;user=phone
sips:1212@gateway.com
sip:alice@192.0.2.4
sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com
sip:alice;day=tuesday@atlanta.com
```

上記の最後のURIの例は、userフィールド値「alice;day=tuesday」を持つ。上で定義されたエスケープのルールは、このフィールドでセミコロンがエスケープされずに現れることを許可する。このプロトコルの目的のために、そのフィールドは不透明(opaque)である。その値の構造は、そのリソースに責任を負うSIPエレメントに対してのみ有用である。

19.1.4 URI の比較 (URI Comparison)

この仕様中のいくつかのオペレーションは、2 つの SIP URI または SIPS URI が等価かどうか決定することを要求する。この仕様では、レジストラサーバは REGISTER リクエストの Contact URI のバインディングを比較する必要がある(10.3 節参照)。SIP URI および SIPS URI は以下のルールにしたがって等価性を比較される。

- SIP URI と SIPS URI は決して等価にならない。
- SIP URI および SIPS URI の userinfo の比較は大文字小文字を区別する。これには、userinfo がパスワードを含むか telephone-subscriber としてフォーマットされている場合も含まれる。他のすべての URI のコンポーネントの比較は、明示的に定義されていなければ、大文字小文字を区別しない。
- SIP URI および SIPS URI を比較するときに、パラメータとヘッダの並び順は重要ではない。
- 「reserved」セット中のもの以外のキャラクター(RFC2396 参考文献[5]参照)は、それらの「"%" HEX HEX」エンコーディングと等価である。
- ホスト名の DNS ルックアップの結果としての IP アドレスは、そのホスト名とマッチしない。
- 2 つの URI が等価であるためには、user、password、host、および port コンポーネントがマッチしなければならない。

user コンポーネントを省略する URI は、それを含む URI にマッチしない。password コンポーネントを省略する URI は、それを含む URI にマッチしない。

初期値を持つコンポーネントを省略する URI は、そのコンポーネントを明示的にそれの初期値とともに含む URI にはマッチしない。たとえば、オプションの port コンポーネントを省略する URI は、明示的に port 5060 を宣言している URI にはマッチしない。transport-parameter、ttl-parameter、user-parameter、および method の各コンポーネントについても同じことがいえる。

sip:user@host が sip:user@host:5060 と等価ではないと定義することは、RFC2543 からの変更である。アドレスを URI から導出するとき、等価な URI からは等価なアドレスが期待される。sip:user@host:5060 という URI は常にポート 5060 に解決される。sip:user@host という URI は、参考文献[4]で定義されている DNS SRV メカニズムを通して他のポートに解決されるかもしれない。

- URI の uri-parameter コンポーネントは、以下のように比較される。
  - 双方の URI に現れるどのような uri-parameter もマッチしなければならない。
  - 一つの URI のみに現れる user、ttl、あるいは method の uri-parameter は、それが初期値を含んでいたとしてもマッチしない。

- maddr パラメータを含む URI は、maddr パラメータを含まない URI にマッチしない。

- 一つの URI のみに現れるその他すべての uri-parameter は、URI を比較するときは無視される。

o URI の header コンポーネントは決して無視されない。存在するいかなる header コンポーネントも、URI がマッチするためには、両方の URI に存在してマッチしなければならない[MUST]。マッチングルールは、20 節の各ヘッダフィールドに対して定義されている。

以下の各組の URI は等価である。

sip:%61lice@atlanta.com;transport=TCP
sip:alice@AtLanTa.CoM;Transport=tcp

sip:carol@chicago.com
sip:carol@chicago.com;newparam=5
sip:carol@chicago.com;security=on

sip:biloxi.com;transport=tcp;method=REGISTER?to=sip:bob%40biloxi.com
sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.com

sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:alice@atlanta.com?priority=urgent&subject=project%20x

以下の各組の URI は等価ではない。

SIP:ALICE@AtLanTa.CoM;Transport=udp (異なる username)
sip:alice@AtLanTa.CoM;Transport=UDP

sip:bob@biloxi.com (異なるポートに解決されることがある)
sip:bob@biloxi.com:5060

sip:bob@biloxi.com (異なるトランスポートに解決されることがある)
sip:bob@biloxi.com;transport=udp

sip:bob@biloxi.com (異なるポートとトランスポートに解決されることがある)
sip:bob@biloxi.com:6000;transport=tcp

sip:carol@chicago.com (異なる header コンポーネント)
sip:carol@chicago.com?Subject=next%20meeting

sip:bob@phone21.boxesbybob.com (phone21.boxesbybob.com が解決され

sip:bob@192.0.2.4　　る結果がたとえそうだったとしてもマッチしない)

等価性は推移的ではないということに注意すること。

o sip:carol@chicago.com と sip:carol@chicago.com;security=on は等価である

o sip:carol@chicago.com と sip:carol@chicago.com;security=off は等価である

o sip:carol@chicago.com;security=on と sip:carol@chicago.com;security=off は等価ではない

19.1.5 URI からリクエストを作る (Forming Request from a URI)

実装は URI から直接リクエストを作るときは気をつける必要がある。名刺や Web ページ、またはたとえ登録された Contact といったプロトコル内部が元であっても、URI は不適切なヘッダフィールドやボディ部分を含むかもしれない。

実装は、提供されたどのような transport、maddr、ttl、または user パラメータも、形成されたリクエストの Request-URI に含めなくてはならない[MUST]。URI が method パラメータを含む場合は、リクエストのメソッドとしてそれの値が使用されなくてはならない[MUST]。method パラメータは Request-URI に置いてはいけない[MUST NOT]。未知の URI パラメータはメッセージの Request-URI に置かなくてはならない[MUST]。

実装は、URI 中のすべてのヘッダまたはボディ部分の存在を、それらをメッセージに含める要望として処理し、リクエストをコンポーネントごとに引き受ける(honor)ことを選択するべきである[SHOULD]。

実装は次の明らかに危険なヘッダフィールドを引き受ける(honor)べきではない[SHOULD NOT]。それは、From、Call-ID、CSeq、Via、および Record-Route である。

実装は、悪意ある攻撃に無断で使用されることがないように、リクエストされたどのような Route ヘッダフィールド値も引き受ける(honor)べきではない[SHOULD NOT]。

実装は、それのロケーションや能力を不正に知らせることになるかもしれないヘッダフィールドを含めるためのリクエストを引き受ける(honor)べきではない[SHOULD NOT]。これには、Accept、Accept-Encoding、Accept-Language、Allow、Contact(ダイアログの用途で)、Organization、Supported、および User-Agent が含まれる。

実装はリクエストされたすべての記述的ヘッダフィールド(Content-Disposition、Content-Encoding、Content-Language、Content-Length、Content-Type、Date、 Mime-Version、および Timestamp)の正確さを検証するべきである[SHOULD]。

与えられた URI からメッセージを構築することによって形成されたリクエストが有効な SIP リクエストでない場合、その URI は無効である。実装はこのリクエストの送信を続行してはいけない[MUST NOT]。その代わりに、それが起こったコンテキスト中で無効な URI に対して取られるべき措置をとるべきである。

構築されたリクエストはいろいろな形で無効になることがある。それには、ヘッダフィールドの構文エ

ラー、URI パラメータの無効な組み合わせ、あるいはメッセージボディの正しくない記述が含まれる(これだけではないが)。

与えられた URI から形成されたリクエストを送ることは、その実装で利用できない能力を要求するかもしれない。たとえば、URI は実装されていないトランスポートや拡張の使用を示するかもしれない。実装は、それの能力に合うようにこれらのリクエストを修正するのではなく、送ることを拒否するべきである[SHOULD]。実装は、それがサポートしない拡張を要求するリクエストを送ってはいけない[MUST NOT]。

たとえば、そのようなリクエストは、未知のあるいは明らかにサポートされていない値を持つ Require ヘッダパラメータまたは URI の method パラメータの存在を通して形成されることがある。

19.1.6 SIP URI と tel URL を関係付ける (Relating SIP URIs and tel URLs)

tel URL(RFC2806 参考文献[9])が SIP URI または SIPS URI に変換されるとき、tel URL の telephone-subscriber 部分全体(すべてのパラメータを含む)は SIP URI または SIPS URI の userinfo 部分に置かれる。

したがって、tel:+358-555-1234567;postd=pp22 は次のようになる

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

または

sips:+358-555-1234567;postd=pp22@foo.com;user=phone

次のようにはならない

sip:+358-555-1234567@foo.com;postd=pp22;user=phone

または

sips:+358-555-1234567@foo.com;postd=pp22;user=phone

通常、SIP URI または SIPS URI にこのように変換された等価な tel URL は、等価な SIP URI または SIPS URI を生み出さないかもしれない。SIP URI または SIPS URI の userinfo は大文字小文字を区別する文字列として比較される。tel URL の大文字小文字を区別しない部分の不一致と tel URL パラメータの並べ替えは tel URL の等価性に影響を与えない。しかし、それらから形成される SIP URI の等価性には影響を与える。

たとえば、

tel:+358-555-1234567;postd=pp22
tel:+358-555-1234567;POSTD=PP22

は等価であるが、

sip:+358-555-1234567;postd=pp22@foo.com;user=phone
sip:+358-555-1234567;POSTD=PP22@foo.com;user=phone

は等価ではない。

同様に、

tel:+358-555-1234567;postd=pp22;isub=1411
tel:+358-555-1234567;isub=1411;postd=pp22

は等価であるが

sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone
sip:+358-555-1234567;isub=1411;postd=pp22@foo.com;user=phone

は等価でない。

この問題を軽減するために、SIP または SIPS URI の userinfo 部分に置く telephone-subscriber を構築するエレメントは、telephone-subscriber の大文字小文字を区別しないすべての部分を小文字に畳み込み、さらに、isdn-subaddress と post-dial を例外として(これは最初に発生し、また、その順通りに発生するので)、telephone-subscriber をレキシカル(lexical)に並べ替えるべきである[SHOULD]。(将来拡張されるパラメータを除く tel URL のすべてのコンポーネントは、大文字小文字を区別しないで比較されるように定義されている。)

この提案にしたがって、以下の 2 つは

tel:+358-555-1234567;postd=pp22
tel:+358-555-1234567;POSTD=PP22

次のようになり

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

以下の 2 つは

tel:+358-555-1234567;tsp=a.b;phone-context=5
tel:+358-555-1234567;phone-context=5;tsp=a.b

次のようになる

sip:+358-555-1234567;phone-context=5;tsp=a.b@foo.com;user=phone

19.2 **オプションタグ** (Option Tags)

オプションタグはSIPの新しいオプション(拡張)を指定するために使用される一意の識別子である。これらのタグはRequireヘッダフィールド(20.32節)、Proxy-Requireヘッダフィールド(20.29節)、Supportedヘッダフィールド(20.37節)、およびUnsupportedヘッダフィールド(20.40節)で使用される。これらのオプションは、それらのヘッダフィールド中で「option-tag = token」形式のパラメータとして現れる(トークンの定義については25節参照のこと)ことに注意すること。

オプションタグはstandards track RFCで定義される。これは過去の慣例からの変更であり、複数ベンダーの相互運用性を継続することを確実にするために制定された(20.32節と20.37節の議論を参照)。簡単に参照できることを保証するために、オプションタグのIANAレジストリが使用される。

19.3 **タグ**

「tag」パラメータはSIPメッセージのToフィールドとFromフィールドで使用される。それは、ダイアログの各参加者からの、2つのtagとCall-IDの組み合わせであるダイアログを識別するための一般的なメカニズムとしての役目を果たす。UAがダイアログ外のリクエストを送るとき、それは、ダイアログIDの「半分」を提供するFrom tagのみを含む。ダイアログはレスポンス(一つまたは複数)で完成する。各レスポンスはToヘッダフィールドに残りの半分を与える。SIPリクエストのフォークとは、一つのリクエストから複数のダイアログが確立できることを意味する。このことも、両方の側から提供されるダイアログ識別子の必要性を説明する。受信側の協力なしには、発信元は一つのリクエストで確立される複数のダイアログを明確化することができない。

tagがリクエストやレスポンスに挿入されるためにUAによって生成されるとき、それはグローバルに一意で、32ビットのランダム性を持って暗号学的にランダムでなくてはならない[MUST]。この選択の要求の特性は、UAがINVITEのFromヘッダに、その同じINVITEに対するレスポンスのToヘッダに置くだろうものとは異なるtagを置くことである。これはUAがそれ自身をセッションに招待するために必要になる。これはPSTNゲートウェイで呼を「ヘアピンする」ための一般的なケースである。同様に、異なる呼に対する2つのINVITEは別のFrom tagを持ち、異なる呼に対する2つのレスポンスは別のToタグを持つだろう。

グローバルな一意性に対する要求とは別に、tagを生成するためのアルゴリズムは実装に固有である。tagはフォールトトレラントシステムにおいて役に立つ。そこでは、サーバの故障後に代わりのサーバでダイアログが回復される。故障したサーバ上のダイアログの一部としてバックアップがリクエストを認識でき、したがってそれがダイアログとそれに関連する他のすべての状態の回復を試みるべきであると決定できるようなtagを、UASは選択できる。

２０．ヘッダフィールド (Header Fields)

ヘッダフィールドの一般的な構文は7.3節に含まれている。本節では、ヘッダフィールドの完全なセットを、構文、意味、および使用方法の注意事項と共にリストする。本節全体を通して、現在のHTTP/1.1仕様(RFC2616 参考文献[8])のX.Y節を参照するために、[HX.Y]という表記を使用する。各ヘッダフィールドの例も与えられている。

メソッドとプロキシ処理に関連するヘッダフィールドについての情報は、表2と表3にまとめられている。

「where」列は、そのヘッダフィールドが使用されるリクエストとレスポンスのタイプを記述する。この

列の値は以下のようである。

R: そのヘッダフィールドはリクエスト中にのみ現れる。

r: そのヘッダフィールドはレスポンス中にのみ現れる。

2xx、4xx、など: 数値や数字の範囲はレスポンスコードを示す。それと共にそのヘッダフィールドを使用できる。

c: そのヘッダフィールドはリクエストからレスポンスにコピーされる。

「where」列の空のエントリは、そのヘッダフィールドがすべてのリクエストおよびレスポンスに存在できることを示す。

「proxy」列は、プロキシがヘッダで実行することができるオペレーションを記述する。

a: そのヘッダフィールドが存在しない場合、プロキシはそれを追加または結合できる。

m: プロキシは既存のヘッダフィールド値を修正できる。

d: プロキシはヘッダフィールド値を削除できる。

r: プロキシはそのヘッダフィールドを読めなければならず、それゆえこのヘッダフィールドは暗号化できない。

その次の 6 列は、メソッド中のヘッダフィールドの存在に関係する。

c: Conditional; そのヘッダフィールドに対する要求はメッセージのコンテキストに依存する。

m: そのヘッダフィールドは必須である。

m*: そのヘッダフィールドは送られるべき[SHOULD]だが、クライアント/サーバはそのヘッダフィールドがないメッセージを受信することに備える必要がある。

o: そのヘッダフィールドはオプションである。

t: そのヘッダフィールドは送られるべき[SHOULD]だが、クライアント/サーバはそのヘッダフィールドがないメッセージを受信することに備える必要がある。ストリームベースのプロトコル(たとえば TCP)がトランスポートとして使用される場合、そのヘッダフィールドが送られなくてはならない[MUST]。

*: そのヘッダフィールドは、メッセージボディが空でない場合に要求される。詳細については 20.14、20.15、7.4 節参照のこと。

-:　そのヘッダフィールドは適用不可能である。

「オプション」は、あるエレメントがそのヘッダフィールドをリクエストまたはレスポンスに含めてもよく[MAY]、UA はリクエストまたはレスポンスにそのヘッダが存在する場合にそれを無視してもよい[MAY]ことを意味する(このルールの例外は、20.32 で議論される Require ヘッダフィールドである)。「必須」のヘッダフィールドは、リクエスト中に存在しなくてはならず[MUST]、そのリクエストを受け取る UAS によって理解されなくてはならない[MUST]。必須のレスポンスヘッダフィールドは、レスポンス中に存在しなくてはならず[MUST]、そのレスポンスを処理する UAC によって理解されなくてはならない[MUST]。「適用不可能」は、そのヘッダフィールドがリクエスト中に存在してはいけない[MUST NOT]ことを意味する。誤ってリクエスト中に置かれた場合、それはそのリクエストを受け取る UAS によって無視されなくてはならない[MUST]。同様に、レスポンスに対して「適用不可能」とラベル付けされたヘッダフィールドは、UAS がそのヘッダフィールドをレスポンス中に置いてはいけなく[MUST NOT]、UAC はレスポンス中のそのヘッダを無視しなくてはならない[MUST]ことを意味する。

UA は理解できない拡張ヘッダパラメータを無視するべきである[SHOULD]。

いくつかの一般的なヘッダフィールド名のコンパクトな形式も、メッセージ全体のサイズが問題となるときに使用するために定義されている。

Contact ヘッダフィールド、From ヘッダフィールド、および To ヘッダフィールドは URI を含む。その URI がカンマ、疑問符、あるいはセミコロンを含む場合、その URI はカギ括弧(< と >)で囲まれなくてはならない[MUST]。いかなる URI パラメータもこれらの括弧内に含められる。URI がカギ括弧で囲まれていない場合、セミコロンで区切られたどのようなパラメータも URI パラメータではなくヘッダパラメータである。

20.1　Accept

Accept ヘッダフィールドは[H14.1]で定義されている構文に従う。Accept ヘッダフィールドが存在しない場合にサーバが application/sdp という初期値を仮定するべきである[SHOULD]ということを除けば、意味合い(semantics)も同じである。

空の Accept ヘッダフィールドは、どの形式も受け入れられないことを意味する。

例:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R	-	o	-	o	m*	o
Accept	2xx	-	-	-	o	m*	o
Accept	415	-	c	-	c	c	c
Accept-Encoding	R	-	o	-	o	o	o
Accept-Encoding	2xx	-	-	-	o	m*	o
Accept-Encoding	415	-	c	-	c	c	c
Accept-Language	R	-	o	-	o	o	o

Accept-Language	2xx		-	-	-	o	m*	o
Accept-Language	415		-	c	-	c	c	c
Alert-Info	R	ar	-	-	-	o	-	-
Alert-Info	180	ar	-	-	-	o	-	-
Allow	R		-	o	-	o	o	o
Allow	2xx		-	o	-	m*	m*	o
Allow	r		-	o	-	o	o	o
Allow	405		-	m	-	m	m	m
Authentication-Info	2xx		-	o	-	o	o	o
Authorization	R		o	o	o	o	o	o
Call-ID	c	r	m	m	m	m	m	m
Call-Info		ar	-	-	-	o	o	o
Contact	R		o	-	-	m	o	o
Contact	1xx		-	-	-	o	-	-
Contact	2xx		-	-	-	m	o	o
Contact	3xx	d	-	o	-	o	o	o
Contact	485		-	o	-	o	o	o
Content-Disposition			o	o	-	o	o	o
Content-Encoding			o	o	-	o	o	o
Content-Language			o	o	-	o	o	o
Content-Length		ar	t	t	t	t	t	t
Content-Type			*	*	-	*	*	*
CSeq	c	r	m	m	m	m	m	m
Date		a	o	o	o	o	o	o
Error-Info	300-699	a	-	o	o	o	o	o
Expires			-	-	-	o	-	o
From	c	r	m	m	m	m	m	m
In-Reply-To	R		-	-	-	o	-	-
Max-Forwards	R	amr	m	m	m	m	m	m
Min-Expires	423		-	-	-	-	-	m
MIME-Version			o	o	-	o	o	o
Organization		ar	-	-	-	o	o	o

表 2: ヘッダフィールドの概要(A～O)

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	ar	-	-	-	o	-	-
Proxy-Authenticate	407	ar	-	m	-	m	m	m
Proxy-Authenticate	401	ar	-	o	o	o	o	o
Proxy-Authorization	R	dr	o	o	-	o	o	o

Proxy-Require	R	ar	-	o	-	o	o	o
Record-Route	R	ar	o	o	o	o	o	-
Record-Route	2xx,18x	mr	-	o	o	o	o	-
Reply-To			-	-	-	o	-	-
Require		ar	-	c	-	c	c	c
Retry-After	404,413,480,486		-	o	o	o	o	o
	500,503		-	o	o	o	o	o
	600,603		-	o	o	o	o	o
Route	R	adr	c	c	c	c	c	c
Server	r		-	o	o	o	o	o
Subject	R		-	-	-	o	-	-
Supported	R		-	o	o	m*	o	o
Supported	2xx		-	o	o	m*	m*	o
Timestamp			o	o	o	o	o	o
To	c(1)	r	m	m	m	m	m	m
Unsupported	420		-	m	-	m	m	m
User-Agent			o	o	o	o	o	o
Via	R	amr	m	m	m	m	m	m
Via	rc	dr	m	m	m	m	m	m
Warning	r		-	o	o	o	o	o
WWW-Authenticate	401	ar	-	m	-	m	m	m
WWW-Authenticate	407	ar	-	o	-	o	o	o

表 3: ヘッダフィールドの概要(P〜Z); (1): tag が追加されてコピーされることもある

```
Accept:application/sdp;level=1,application/x-private,text/html
```

20.2 Accept-Encoding

Accept-Encoding ヘッダフィールドは、Accept に似ているが、レスポンスで受け入れ可能な content-codings[H3.5]を制限する。[H14.3]を参照のこと。SIP における意味合い(semantics)は[H14.3]で定義されているものと同じである。

空の Accept-Encoding ヘッダフィールドは認められている。それは Accept-Encoding: identity と等価である。つまり、identity エンコーディング(エンコーディングなしの意味)だけが許可されているということである。

Accept-Encoding ヘッダフィールドが存在しない場合、初期値である identity をサーバは仮定するべきである[SHOULD]。

これは HTTP の定義とは若干異なる。HTTP の定義では、存在しない場合はどのようなエンコーディングでも使用できるが、identity エンコーディングが好ましい。

例:

```
Accept-Encoding: gzip
```

20.3 Accept-Language

Accept-Language ヘッダフィールドは、レスポンス中でメッセージボディとして運ばれるリーズンフレーズ、セッション記述、あるいはステータスレスポンスに好ましい言語を示すために、リクエストで使用される。Accept-Language ヘッダフィールドが存在しない場合、そのクライアントに対してはすべての言語が認められるとサーバは仮定するべきである[SHOULD]。

Accept-Language ヘッダフィールドは[H14.4]で定義されている構文に従う。言語を q パラメータに基づいて順番付けするルールは SIP にも適用される。

例:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

20.4 Alert-Info

INVITE リクエスト中に存在するとき、Alert-Info ヘッダフィールドは UAS に対して代替の呼び出し音(ring tone)を指定する。180(Ringing)レスポンス中に存在するとき、Alert-Info ヘッダフィールドは UAC に対して代替の ringback を指定する。典型的な使用方法は、プロキシが特有の呼び出し機能を提供するためにこのヘッダを挿入することである。

Alert-Info ヘッダフィールドはセキュリティリスクを持ち込むことがある。これらのリスクとそれをハンドリングする方法は、Call-Info ヘッダフィールドについて議論している 20.9 節で(リスクが同一のため)議論する。

さらに、ユーザはこの機能を選択的に無効にすることができるべきである[SHOULD]。

これは、信頼できないエレメントによってこのヘッダが使用されることで生じるかもしれない破壊を防ぐ役に立つ。

例:

```
Alert-Info: <http://www.example.com/sounds/moo.wav>
```

20.5 Allow

Allow ヘッダフィールドは、そのメッセージを生成する UA がサポートするメソッドの組をリストする

UA が理解する ACK と CANCEL を含むすべてのメソッドは、Allow ヘッダが存在するときはその中のメソッドのリストに含まれなくてはならない[MUST]。Allow ヘッダフィールドがないことを、メッセージを送る UA がどのメソッドもサポートしないという意味に解釈してはいけない[MUST NOT]。そうではなくむしろ、それは、UA がどのメソッドをサポートするかについてのいかなる情報も提供していないことを示唆する。

OPTIONS 以外のメソッドに対するレスポンス中で Allow ヘッダフィールドを供給することは、必要とされるメッセージの数を減らす。

例:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

20.6 Authentication-Info

Authentication-Info ヘッダフィールドは相互認証に HTTP ダイジェスト認証を提供する。UAS は、Authorization ヘッダフィールドに基づきダイジェスト認証を使用して正しく認証されたリクエストに対する 2xx レスポンス中に、このヘッダフィールドを含めてもよい[MAY]。

構文と意味合い(semantics)は RFC2617(参考文献[17])で規定されているものに従う。

例:

```
Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c"
```

20.7 Authorization

Authorization ヘッダフィールドは、UA の認証の信用証明書を含む。22.2 節で Authorization ヘッダフィールドの使用方法の概略を示し、22.4 節で HTTP 認証で使用されるときの構文と意味合い(semantics)を述べる。

Proxy-Authorization に加えてこのヘッダフィールドは、複数ヘッダフィールド値に関する一般ルールを破る。カンマ区切りのリストではないが、このヘッダフィールド名は複数回存在でき、7.3 節で述べられてる通常のルールを使用して一つのヘッダ行に結合してはいけない[MUST NOT]。

下記の例では、Digest パラメータは引用符で囲まれていない。

```
Authorization:
Digest username="Alice", realm="atlanta.com",
nonce="84a4cc6f3082121f32b42a2187831a9e",
response="7587245234b3434cc3412213e5f113a5432"
```

20.8 Call-ID

Call-ID ヘッダフィールドは、特定の招待または特定のクライアントのすべての登録を一意に識別する。一つのマルチメディアカンファレンスは異なる Call-ID でいくつかの呼を引き起こすことができる(たとえば、ユーザが一個人を(長時間継続している)同一のカンファレンスに複数回招待する場合)。Call-ID は大文字小文字を区別し、単純にバイトごとに比較される。

Call-ID ヘッダフィールドのコンパクトフォームは i である。

例:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com
i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4
```

20.9 Call-Info

Call-Info ヘッダフィールドは、それがリクエスト中あるいはレスポンス中のどちらで見つけられるかに応じて、発信者または着信者についての付加的な情報を提供する。URI の目的は「purpose」パラメータで述べられる。「icon」パラメータは、発信者または着信者のアイコン表示に適切な画像を示す。「info」パラメータは一般的に、たとえば Web ページを介して、発信者あるいは着信者を説明する。「card」パラメータは、たとえば、vCard(参考文献[36])または LDIF(参考文献[37])フォーマットで名刺を提供する。追加のトークンを IANA と 27 節の手順を使用して登録できる。

Call-Info ヘッダフィールドの利用はセキュリティリスクを引き起こすことがある。着信者が悪意のある発信者が提供した URI を取り出す場合、着信者は、不適切または不快なコンテンツ、危険または不法なコンテンツ、などを表示するリスクにさらされる。したがって、UA は、そのヘッダフィールドを発信したエレメントの信頼性を検証でき、そのエレメントを信頼できる場合にのみ、Call-Info ヘッダフィールドの情報を描画することが推奨される[RECOMMENDED]。プロキシもこのヘッダフィールドをリクエストに挿入できるので、これは相手 UA でなくてもよい。

例:

```
Call-Info:  <http://wwww.example.com/alice/photo.jpg> ;purpose=icon,
<http://www.example.com/alice/> ;purpose=info
```

20.10 Contact

Contact ヘッダフィールドは、URI を提供する。その URI の意味は、それが含まれるリクエストまたはレスポンスのタイプに依存する。

Contact ヘッダフィールド値は、ディスプレイネーム、URI パラメータを持つ URI、およびヘッダパラメータを含むことができる。

このドキュメントでは Contact のパラメータ「q」と「expires」を定義する。これらのパラメータは Contact が REGISTER リクエスト、REGISTER レスポンス、または 3xx レスポンス中に存在するときにのみ使用される。追加のパラメータが他の仕様で定義されるかもしれない。

ヘッダフィールド値がディスプレイネームを含むとき、URI(すべての URI パラメータを含む)は「<」と「>」で囲まれる。「<」と「>」が存在しない場合、URI の後に続く全てのパラメータは URI パラメータではなく、ヘッダパラメータである。ディスプレイネームはトークンあるいは(より大きな文字セットが望まれる場合は)引用符で囲まれた文字列にできる。

「display-name」が空であっても、「addr-spec」がカンマ、セミコロン、または疑問符を含む場合は「name-addr」形式が使用されなくてはならない[MUST]。display-name と「<」の間に LWS があってもなくて

もよい。

ディスプレイネーム、URI と URI パラメータ、およびヘッダパラメータをパースするためのこれらのルールは、To と From のヘッダフィールドにも適用される。

Contact ヘッダフィールドは、HTTP の Location ヘッダフィールドに似た機能を果たす。そうではあるが、HTTP の Location ヘッダフィールドは引用符で囲まれていない一つのアドレスを認めるだけである。URI は予約文字としてカンマとセミコロンを含むので、それらはそれぞれヘッダの区切文字およびパラメータの区切文字と間違えられることがある。

Contact ヘッダフィールドのコンパクトフォームは(moved の) m である。

例:

```
Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
;q=0.7; expires=3600,
"Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
m:  <sips:bob@192.0.2.4>;expires=60
```

20.11 Content-Disposition

Content-Disposition ヘッダフィールドは、メッセージボディまたは(マルチパートメッセージでは)メッセージボディ部分が UAC や UAS によってどのように解釈されるかを説明する。この SIP ヘッダフィールドは MIME の Content-Type (RFC2183 参考文献[18])を拡張する。

Content-Disposition ヘッダのいくつかの新しい「disposition-types」が SIP で定義された。「session」という値は、ボディ部分が呼または early(呼以前の)メディアのいずれかに対するセッションを記述することを示す。「render」という値は、ボディ部分が表示(display)されるか、そうでなければ描画/再生/提示(render)されるべきであることを示す。(SIP メッセージの MIME ボディはユーザに対して表示されないことが多いので)MIME ボディがメッセージ全体のレンダリングの一部として表示されるという暗示的意味を避けるために、「inline」ではなく「render」という値が使用されることに注意すること。下位互換性のために、Content-Disposition ヘッダがない場合、サーバは、Content-Type が application/sdp のボディは disposition が「session」であると仮定する一方、その他の content タイプは「render」であると仮定するべきである[SHOULD]。

「icon」という disposition タイプは、ボディ部分が、発信者または着信者のアイコン表示に適切な画像を含むことを示す。それは、メッセージが受け取られたとき、あるいはダイアログが行われている間ずっと、ユーザエージェントが通知目的で描画してもよい。「alert」という値は、ボディ部分がオーディオクリップのような情報を含むことを示す。それは、ユーザにリクエスト(通常、ダイアログを開始するリクエスト)の受信に対する注意を喚起する試みとしてユーザエージェントが再生するべきである。この注意喚起のボディは、たとえば電話の呼び出し音として 180(Ringing)暫定レスポンス(provisional response)が送られた後に再生することができる。

ユーザにコンテンツを描画/再生/提示(render)する disposition タイプを持ついかなる MIME ボディも、

メッセージが適切に認証されたときにのみ処理されるべきである。

ハンドリングパラメータ(handling-param)は、content タイプまたは disposition タイプが理解できないメッセージボディを受け取った場合に、UAS がどのように動作するべきかを記述する。このパラメータは「optional」と「required」という値を定義している。ハンドリングパラメータがない場合は、「required」と仮定されるべきである[SHOULD]。ハンドリングパラメータについては RFC3204(参考文献[19])に述べられている。

このヘッダフィールドがない場合、MIME タイプが content disposition の初期値を決定する。規定されない場合には「render」と仮定される。

例:

```
Content-Disposition: session
```

20.12 Content-Encoding

Content-Encoding ヘッダフィールドは media-type のモディファイアとして使用される。これが存在するとき、その値は entity-body にどんな付加的なコンテンツコーディングが適用されているかを示し、それゆえ Content-Type ヘッダフィールドによって参照される media-type を取得するためにどんなデコーディングメカニズムが適用されなくてはならない[MUST]かを示す。Content-Encoding は主に、ボディの元となっているメディアタイプのアイデンティティを失わずにそれを圧縮することを可能にするために使用される。

entity-body に複数のエンコーディングが適用されている場合、適用された順番でコンテンツコーディングがリストされなくてはならない[MUST]。

すべての content-coding 値は大文字小文字を区別しない。IANA は content-coding 値トークンのレジストリとしての役を務める。content-coding の構文の定義については[H3.5]を参照のこと。

クライアントはコンテントエンコーディングをリクエスト中のボディに適用してもよい[MAY]。サーバはレスポンス中のボディにコンテントエンコーディングを適用してもよい[MAY]。サーバはリクエストの Accept-Encoding ヘッダフィールドにリストされているエンコーディングのみを使用しなくてはならない[MUST]。

Content-Encoding ヘッダフィールドのコンパクトフォームは e である。

例

```
Content-Encoding: gzip
e: tar
```

20.13 Content-Language

[H14.12]参照。

例:

```
Content-Language: fr
```

20.14 Content-Length

Content-Length ヘッダフィールドは、受信者に送られた message-body のサイズを、オクテットの個数(10進数)で示す。アプリケーションは、エンティティのメディアタイプに関係なく、送られる message-body のサイズを示すためにこのフィールドを使用するべきである[SHOULD]。ストリームベースのプロトコル(TCP など)がトランスポートとして使用される場合は、このヘッダフィールドが使用されなくてはならない[MUST]。

message-body のサイズにはヘッダフィールドとボディを分ける CRLF を含まない。ゼロ以上のいかなる Content-Length も有効な値である。メッセージ中にボディが存在しない場合、Content-Length ヘッダフィールドはゼロに設定されなくてはならない[MUST]。

Content-Length を省略する能力は、レスポンスを動的に生成する CGI ライクのスクリプトの作成を容易にする。

このヘッダフィールドのコンパクトフォームは l である。

例:

```
Content-Length: 349
l: 173
```

20.15 Content-Type

Content-Type ヘッダフィールドは受信者に送られた message-body のメディアタイプを示す。「media-type」エレメントは[H3.7]で定義されている。Content-Type ヘッダフィールドはボディが空でない場合は必ず存在しなくてはならない[MUST]。ボディが空で Content-Type ヘッダフィールドが存在する場合、その特定のタイプのボディの長さがゼロであることを示す(たとえば、空の音声ファイル)。

このヘッダのコンパクトフォームは c である。

例:

```
Content-Type: application/sdp
c: text/html; charset=ISO-8859-4
```

20.16 CSeq

リクエスト中の CSeq ヘッダフィールドは、一つの 10 進数のシーケンス番号とリクエストメソッドを含む。シーケンス番号は 32 ビットの符号なし整数で表現可能でなくてはならない[MUST]。CSeq のメソッド部分は大文字小文字を区別する。CSeq ヘッダは、ダイアログ内のトランザクションを順番付けるため、トランザクションを一意に識別するための手段を提供するため、そして新規リクエストとリクエストの再送を区別するため、の役に立つ。2 つの CSeq ヘッダフィールドは、シーケンス番号とメソッドが同一であれば、等価であ

るとみなされる。

例:

```
CSeq: 4711 INVITE
```

20.17 Date

Date ヘッダフィールドは日付と時間を含む。HTTP/1.1 とは違い、SIP は、日付に最新の RFC1123(参考文献[20])のフォーマットのみをサポートする。RFC1123 ではあらゆるタイムゾーンを許可するが、[H3.3]にあるように、SIP は SIP-date のタイムゾーンを GMT に制限する。RFC1123 の date は大文字小文字を区別する。

Date ヘッダフィールドは、リクエストまたはレスポンスが最初に送られた時間を反映する。

Date ヘッダフィールドは、時間の概念を取得するためのバッテリーバックアップされたクロックを持たない単純なエンドシステムによって使用されることがある。しかしながら、それは GMT 形式なので、クライアントが GMT からのオフセットを知っている必要がある。

例:

```
Date: Sat, 13 Nov 2010 23:29:00 GMT
```

20.18 Error-Info

Error-Info ヘッダフィールドは、エラーステータスレスポンスについての付加的な情報へのポインタを提供する。

SIP の UAC は、PC ソフトクライアントにおけるポップアップウィンドウや音や、一般の電話やゲートウェイを介して接続されたエンドポイントにおける音声限定など様々なユーザインターフェース能力を持つ。詳細なリーズンフレーズを含むエラーステータスコードを送ることと録音されたオーディオを再生することのいずれかを選択してエラーを生成することをサーバに強いるのではなく、Error-Info ヘッダフィールドはその両方を送ることを可能にする。UAC は次に、どちらのエラーインジケーターを発信者に表示/再生するかという選択肢を持つ。

UAC は、Error-Info ヘッダフィールドの SIP URI または SIPS URI をリダイレクトの Contact であるかのように処理し、新規 INVITE を生成してもよい[MAY]。その結果として、録音されたアナウンスのセッションが確立されることになる。非 SIP URI をユーザに対して表示/再生してもよい[MAY]。

例:

```
SIP/2.0 404 The number you have dialed is not in service
Error-Info: <sip:not-in-service-recording@atlanta.com>
```

20.19 Expires

Expires ヘッダフィールドは、メッセージ(またはコンテンツ)がそれ以降期限切れになる相対時間を与え

る。

これの正確な意味は、メソッドに依存する。

INVITE の期限切れ時間は、その招待で生じるセッションの実際の継続時間に影響しない。しかしながら、セッション記述プロトコルがセッションの継続時間のタイムリミットを表現する能力を提供するかもしれない。

このフィールドの値は、リクエストの受信時から計測された 0 と 2^{32}-1 の間の、秒を表す整数(10 進数)である。

例:

```
Expires: 5
```

20.20 From

From ヘッダフィールドはリクエストの起動者を示す。これはダイアログの起動者とは異なるかもしれない。着信者が発信者に送るリクエストは、From ヘッダフィールドに着信者のアドレスを使用する。

オプションの「display-name」は、ヒューマンユーザインターフェースによって表示されることを意図している。システムは、クライアントのアイデンティティが隠されたままになっている場合、ディスプレイネーム「Anonymous」を使用するべきである[SHOULD]。「display-name」が空であっても、「addr-spec」がカンマ、セミコロン、または疑問符を含む場合は「name-addr」形式が使用されなくてはならない[MUST]。構文の問題は 7.3.1 節で議論されている。

2 つの From ヘッダフィールドは、その URI がマッチし、かつ、パラメータがマッチする場合は、等価である。一方のヘッダフィールドにあり、他方にない拡張パラメータは、比較では無視される。これは、ディスプレイネーム、および、カギ括弧(＜と＞)の有無は、マッチングに影響を与えないということを意味する。

ディスプレイネーム、URI と URI パラメータ、およびヘッダフィールドパラメータをパースするためのルールについては 20.10 節参照のこと。

From ヘッダフィールドのコンパクトフォームは f である。

例:

```
From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s
From: sip:+12125551212@server.phone2net.com;tag=887s
f: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

20.21 In-Reply-To

In-Reply-To ヘッダフィールドは、この呼が参照する、または返送する Call-ID を列挙する。これらの Call-ID は、クライアントがキャッシュして、返送する電話の In-Reply-To ヘッダフィールドに含められる

かもしれない。

これは ACD (automatic call distribution)システムが返信を最初の呼の発信元にルーティングすることを可能にする。これはまた、着信者が以前彼らが発信した呼への返信のみを受け入れられるように呼をフィルターすることを可能にする。このフィールドはリクエスト認証のための代替ではない。

例:

```
In-Reply-To:70710@saturn.bell-tel.com,17320@saturn.bell-tel.com
```

20.22 Max-Forwards

Max-Forwards ヘッダフィールドは、リクエストを次のダウンストリームサーバにフォワードできるプロキシまたはゲートウェイの数を制限するために、すべての SIP メソッドと共に使用しなければならない。これは、リクエストチェーン内で失敗またはループしているように見えるリクエストチェーンを、クライアントがトレースすることを試みるときにも有用である。

Max-Forwards 値は、このリクエストメッセージがフォワードされることを認められている残り回数を示す 0～255 の整数である。このカウントは、そのリクエストをフォワードする各サーバで減少させられる。推奨される初期値は 70 である。

このヘッダフィールドは、他の方法でループの検知を保証できないエレメントによって挿入されるべきである。たとえば、B2BUA は Max-Forwards ヘッダフィールドを挿入するべきである。

例:

```
Max-Forwards: 6
```

20.23 Min-Expires

Min-Expires ヘッダフィールドは、そのサーバで管理されるソフトステート(soft-state)のエレメントにおいてサポートされる最小リフレッシュ間隔を伝える。これは、レジストラサーバによって保存される Contact ヘッダフィールドを含む。Min-Expires ヘッダフィールドは、0 から 2^{32}-1 までの秒数を表す 10 進整数を含む。423(Interval Too Brief)レスポンス中でのこのヘッダフィールドの用法は、10.2.8、10.3、および 21.4.17 節で述べられている。

例:

```
Min-Expires: 60
```

20.24 MIME-Version

[H19.4.1]参照。

例:

MIME-Version: 1.0

20.25 Organization

Organization ヘッダフィールドはリクエストまたはレスポンスを発行するエンティティが所属する組織の名称を伝える。

このフィールドは呼をフィルターするためにクライアントソフトウェアが使用してもよい[MAY]。

例:

Organization: Boxes by Bob

20.26 Priority

Priority ヘッダフィールドは、クライアントが認識するリクエストの緊急度を示す。Priority ヘッダフィールドは、SIP リクエストが受信側の人またはエージェントに対して持つべき優先順位を記述する。たとえばそれは、呼のルーティングや受け入れの決定において考慮に入れられるかもしれない。これらの決定で、Priority ヘッダフィールドを含まないメッセージは、あたかもそれが「normal」という Priority を指定したかのように扱われるべきである[SHOULD]。Priority ヘッダフィールドは、ルータのパケットフォワーディングの優先順位や PSTN ゲートウェイのサーキットへのアクセスといったようなコミュニケーションリソースの用法に影響を与えない。Priority ヘッダフィールドは「non-urgent」、「normal」、「urgent」、および「emergency」を持つことができるが、追加の値がどこかで定義されることがあり得る。「emergency」という値は、生命、身体、または財産が差し迫った危機にあるときにのみ使用されることが推奨される[RECOMMENDED]。他の点では、このヘッダフィールドに対しては意味合い(semantics)は定義されていない。

これらは、RFC2076(参考文献[38])の値に「emergency」を追加したものである。

例:

Subject: A tornado is heading our way!
Priority: emergency

または

Subject: Weekend plans
Priority: non-urgent

20.27 Proxy-Authenticate

Proxy-Authenticate ヘッダフィールド値は、認証チャレンジを含む。

このヘッダフィールドの使用方法は[H14.33]で定義されている。使用方法についての更なる詳細については 22.3 節参照のこと。

例:

```
Proxy-Authenticate: Digest realm="atlanta.com",
domain="sip:ss1.carrier.com", qop="auth",
nonce="f84f1cec41e6cbe5aea9c8e88d359",
opaque="", stale=FALSE, algorithm=MD5
```

20.28 Proxy-Authorization

Proxy-Authorization ヘッダフィールドは、認証を求めるプロキシにクライアントがそれ自身(またはそれのユーザ)の識別をすることを可能にする。Proxy-Authorization フィールド値は、プロキシのためのユーザエージェントの認証情報および(または)要求されているリソースの領域(realm)を含む信用証明書から成る。

このヘッダフィールドの使用方法の定義については 22.3 節を参照のこと。

Authorization に加えてこのヘッダフィールドは、複数ヘッダフィールド名に関する一般ルールを破る。カンマ区切りのリストではないが、このヘッダフィールド名は複数回存在でき、7.3.1 節で述べられてる通常のルールを使用して一つのヘッダに結合してはいけない[MUST NOT]。

例:

```
Proxy-Authorization:Digestusername="Alice", realm="atlanta.com",
nonce="c60f3082ee1212b402a21831ae",
response="245f23415f11432b3434341c022"
```

20.29 Proxy-Require

Proxy-Require ヘッダフィールドは、プロキシによってサポートされなければならない、プロキシが理解する(proxy-sensitive)機能を示すために使用される。このメッセージの仕組みと使用例の詳細については 20.32 節参照のこと。

例:

```
Proxy-Require: foo
```

20.30 Record-Route

Record-Route ヘッダフィールドは、ダイアログ中の以降のリクエストをプロキシを通してルーティングさせるために、そのプロキシによってリクエストに挿入される。

Route ヘッダフィールドと共にそれを使用することについての例は 16.12.1 節で述べられている。

例:

```
Record-Route:<sip:server10.biloxi.com;lr>, <sip:bigbox3.site3.atlanta.com;lr>
```

20.31 Reply-To

Reply-To ヘッダフィールドは、From ヘッダフィールドと異なるかもしれない論理的な返送 URI を含む。たとえば、その URI は、受け取れなかった電話あるいは確立されなかったセッションに返信するために使用してもよい[MAY]。ユーザが匿名(anonymous)のままでいることを望む場合、Reply-To ヘッダフィールドはリクエストから省略されるかまたはどのような個人情報も漏らさないような方法で存在させられるべきである[SHOULD]。

「display-name」が空であっても、「addr-spec」がカンマ、セミコロン、または疑問符を含む場合は「name-addr」形式が使用されなくてはならない[MUST]。構文の問題は 7.3.1 節で議論されている。

例:

```
Reply-To: Bob <sip:bob@biloxi.com>
```

20.32 Require

Require ヘッダフィールドは、リクエストを処理するために UAS がサポートすることを UAC が期待するオプションについて、UAC が UAS に伝えるために使用される。オプションのヘッダではあるが、Require が存在する場合はそれを無視してはいけない[MUST NOT]。

Require ヘッダフィールドは 19.2 節で述べられているオプションタグのリストを含む。各オプションタグは、リクエストを処理するために理解されなくてはならない[MUST]SIP の拡張を定義する。これは、ある特定の拡張ヘッダフィールドのセットが理解される必要があることを示すためによく利用される。この仕様に準拠する UAC は、standards-track RFC に一致するオプションタグのみを含まなくてはならない[MUST]。

例:

```
Require: 100rel
```

20.33 Retry-After

Retry-After ヘッダフィールドは、リクエストを行うクライアントがどれくらいの間サービスを利用できないと予測されるかを示すために 500(Server Internal Error) または 503(Service Unavailable)レスポンスで、また、着信側パーティが再び有効になると予測されるのはいつかを示すために 404(Not Found)、413(Request Entity Too Large)、480(Temporarily Unavailable)、486(Busy Here)、600(Busy)、あるいは 603(Decline)レスポンスで使用できる。このフィールドの値は、レスポンスがあった時間後の秒をあらわす正の整数(10 進数)である。

折り返し電話の時間についての付加情報を示すオプションのコメントを使用できる。オプションの「duration」パラメータは、着信側パーティが有効になりはじめる時間から開始してどれだけの間到達可能であるかを示す。duration パラメータが与えられない場合、サービスは無期限に利用可能であると仮定される。

例:

```
Retry-After: 18000;duration=3600
```

Retry-After: 120 (I'm in a meeting)

20.34 Route

Route ヘッダフィールドは、リストされたプロキシのセットを経由してリクエストをルーティングさせるために使用される。Route ヘッダフィールドの使用例は 16.12.1 節で述べられている。

例:

Route:<sip:bigbox3.site3.atlanta.com;lr>, <sip:server10.biloxi.com;lr>

20.35 Server

Server ヘッダフィールドは、リクエストをハンドリングするために UAS が使用するソフトウェアについての情報を含む。

サーバの具体的なソフトウェアバージョンを明かすことは、セキュリティホールを含むことが知られているソフトウェアへの攻撃に対してサーバがより脆弱になることを認めるかもしれない。実装者は Server ヘッダフィールドを、設定可能なオプションにすべきである[SOULD]。

例:

Server: HomeServer v2

20.36 Subject

Subject ヘッダフィールドは、呼の概要を提供するか、または呼の性質を示す。そうすることでセッション記述をパースすることなしに呼のフィルタリングを可能にする。セッション記述は招待と同じサブジェクト表示を使用する必要はない。

このヘッダフィールドのコンパクトフォームは s である。

例:

Subject: Need more boxes
s: Tech Support

20.37 Supported

Supported ヘッダフィールドは、UAC または UAS がサポートするすべての拡張を列挙する。

Supported ヘッダフィールドは、UAC または UAS で理解される、19.2 節で述べられているオプションタグのリストを含む。この仕様に準拠する UA は standards-track RFC に一致するオプションタグのみを含まなくてはならない[MUST]。空の場合は、拡張がひとつもサポートされないことを意味する。

Supported ヘッダフィールドのコンパクトフォームは k である。

例:

```
Supported: 100rel
```

20.38 Timestamp

Timestamp ヘッダフィールドは、UAC が UAS にそのリクエストをいつ送ったのかについて述べる。

このヘッダフィールドを含むリクエストに対するレスポンスをどのように生成するかについての詳細は、8.2.6 節参照のこと。このヘッダを利用する規範となる動作はここでは定義されないが、それは拡張あるいは SIP アプリケーションが RTT 予測値を取得することを可能にする。

例:

```
Timestamp: 54
```

20.39 To

To ヘッダフィールドはリクエストの論理的な受信者を指定する。

オプションの「display-name」は、ヒューマンユーザインターフェースによって表示されることを意図している。「tag」パラメータは、ダイアログを特定するための一般的なメカニズムとしての役に立つ。

「tag」パラメータの詳細については 19.3 節参照のこと。

To ヘッダフィールドが等価かどうかの比較は、From ヘッダフィールドの比較と同じである。ディスプレイネーム、URI と URI パラメータ、およびヘッダフィールドラメータをパースするためのルールについては 20.10 節参照のこと。

このヘッダのコンパクトフォームは t である。

以下は有効な To ヘッダフィールドの例である。

```
To: The Operator <sip:operator@cs.columbia.edu>;tag=287447
t: sip:+12125551212@server.phone2net.com
```

20.40 Unsupported

Unsupported ヘッダフィールドは、UAS がサポートしない機能をリストする。モチベーションについては 20.32 節を参照のこと。

例:

```
Unsupported: foo
```

20.41 User-Agent

User-Agent ヘッダフィールドは、リクエストを開始する UAC についての情報を含む。このヘッダフィールドの意味合い(semantics)は[H14.43]で定義されている。

ユーザエージェントの具体的なソフトウェアバージョンを明かすことは、セキュリティホールを含むことが知られているソフトウェアへの攻撃に対してユーザエージェントがより脆弱になることを認めるかもしれない。実装者は User-Agent ヘッダフィールドを、設定可能なオプションにするべきである[SHOULD]。

例:

```
User-Agent: Softphone Beta1.5
```

20.42 Via

Via ヘッダフィールドは、これまでにリクエストがたどったパスと、レスポンスをルートするときにたどるべきパスを示す。Via ヘッダフィールド値の branchID パラメータは、トランザクション識別子としての役に立ち、ループ検知のためにプロキシに利用される。

Via ヘッダフィールド値は、メッセージを送るために使用されるトランスポートプロトコル、クライアントのホスト名またはネットワークアドレス、およびおそらくはそれがレスポンスを受け取ることを望むポート番号を含む。Via ヘッダフィールド値は、「maddr」、「ttl」、「received」、および「branch」などのパラメータ(意味と使用方法は他の節で述べられている)を含むこともできる。この仕様に準拠する実装では、branch パラメータの値は 8.1.1.7 節で述べられている「z9hG4bK」というマジッククッキーで始まらなくてはならない[MUST]。

ここで定義されているトランスポートプロトコルは、「UDP」、「TCP」、「TLS」、および「SCTP」である。「TLS」は TCP 上の TLS(TLS over TCP)を意味する。リクエストが SIPS URI に対して送られるとき、プロトコルは SIP を示したままで、トランスポートプロトコルは TLS になる。

```
Via:SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7
Via:SIP/2.0/UDP 192.0.2.1:5060 ;received=192.0.2.207;branch=z9hG4bK77asjd
```

このヘッダのコンパクトフォームは v である。

この例では、メッセージは 2 つのアドレス、192.0.2.1 と 192.0.2.207、を持つマルチホームホストから開始された。送信者はどのネットワークインターフェースが使用されるかということについて間違った推測をした。Erlang.bell-telephone.com はミスマッチに気付き、前のホップの Via ヘッダフィールド値に(そのパケットが実際にやって来たアドレスを含む)パラメータを追加した。

ホストまたはネットワークアドレスとポート番号は、SIP URI の構文にしたがうために必要ではない。特に、以下に示すように「:」または「/」の両側に LWS をおくことが許可されている。

```
Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16
;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1
```

この仕様では、すべてのリクエストに branch パラメータが存在することを必須としているが、ヘッダフィールドの BNF はそれがオプションであることを示している。このことは、branch パラメータを挿入することを必須としていなかった RFC2543 エレメントとの相互オペレーションを可能にする。

2 つの Via ヘッダフィールドは、その sent-protocol と sent-by-field が同一で、かつ、どちらも同じパラメータのセットを持ち、かつ、すべてのパラメータ値が同一の場合、等価である。

20.43 Warning

Warning ヘッダフィールドはレスポンスのステータスについての付加情報を伝えるために使用される。Warning ヘッダフィールド値はレスポンスで送られ、3 桁の警告コード(warn-code)、ホスト名、および警告文(warn-text)を含む。

「warn-text」は、レスポンスを受け取るユーザ(人間)が理解できる可能性が最も高い自然言語であるべきである。この決定は、ユーザのロケーション、リクエストの Accept-Language フィールド、あるいはレスポンスの Content-Language フィールドのような利用可能な情報に基づいて下すことができる。デフォルトの言語は i-default(参考文献[21])である。

現在定義されている「warn-code」が以下にリストされている。それぞれには推奨される warn-text とその意味説明が英語で付いている。これらの警告は、セッション記述によって引き起こされた失敗について述べている。警告コードの最初の桁は、SIP 固有の警告であることを示す 3 で始まる。Warning の 300 から 329 はセッション記述のキーワードの問題を示すために予約されている。330 から 339 はセッション記述で要求された基本的なネットワークサービスに関係する警告。370 から 379 はセッション記述で要求された定量的な QoS パラメータに関係する警告。そして、390 から 399 は上記のカテゴリに当てはまらないその他の警告である。

300 Incompatible network protocol (互換性がないネットワークプロトコル):
セッション記述に含まれる一つ以上のネットワークプロトコルが利用できない。

301 Incompatible network address formats (互換性がないネットワークアドレスフォーマット):
セッション記述に含まれる一つ以上のネットワークアドレスフォーマットが利用できない。

302 Incompatible transport protocol (互換性がないトランスポートプロトコル):
セッション記述で述べられている一つ以上のトランスポートプロトコルが利用できない。

303 Incompatible bandwidth units (互換性がない帯域幅の単位):
セッション記述に含まれる一つ以上の帯域幅測定単位が理解できなかった。

304 Media type not available (利用できないメディアタイプ):
セッション記述に含まれている一つ以上のメディアタイプが利用できない。

305 Incompatible media format (互換性がないメディアフォーマット):
セッション記述に含まれている一つ以上のメディアフォーマットが利用できない。

306 Attribute not understood (理解できない属性):
セッション記述中の一つ以上のメディア属性がサポートされていない。

307 Session description parameter not understood (理解できないセッション記述パラメータ):
上にリストされた以外のパラメータを理解できなかった。

330 Multicast not available (マルチキャストは利用できない): ユーザがいる場所のサイトはマルチキャストをサポートしていない。

331 Unicast not available (ユニキャストは利用できない):
ユーザがいる場所のサイトはユニキャストによるコミュニケーションをサポートしていない(通常、ファイアウォールが存在するため)。

370 Insufficient bandwidth (帯域不足):
セッション記述で指定された帯域またはメディアで定義された帯域が、利用できることがわかっているものを超えている。

399 Miscellaneous warning (その他の警告):
ユーザ(人間)に提示される、またはログに残される任意の情報を警告文に含めることができる。この警告を受け取るシステムは、どのような自動動作もとってはいけない[MUST NOT]。

1xx および 2xx は HTTP/1.1 に採用されている。

27.2 節で定義されているように、追加の warn-code を、IANA を通して定義できる。

例:

```
Warning: 307 isi.edu "Session parameter 'foo' not understood"
Warning: 301 isi.edu "Incompatible network address type 'E.164'"
```

20.44 WWW-Authenticate

WWW-Authenticate ヘッダフィールド値は、認証チャレンジを含む。使用方法の更なる詳細については 22.2 節を参照のこと。

例:

```
WWW-Authenticate: Digest realm="atlanta.com",
domain="sip:boxesbybob.com", qop="auth",
nonce="f84f1cec41e6cbe5aea9c8e88d359",
opaque="", stale=FALSE, algorithm=MD5
```

２１．レスポンスコード

レスポンスコードはHTTP/1.1レスポンスコードに合致し、拡張されている。すべてのHTTP/1.1レスポンスコードが割り当てられているわけではなく、ここでは割り当てられているものだけについて述べる。他のHTTP/1.1レスポンスコードは使用されるべきではない[SHOULD NOT]。また、SIPは新しいクラス6xxを定義する。

21.1　暫定レスポンス(provisional response)1xx

暫定レスポンス(provisional response)は(通知レスポンスとしても知られている)、コンタクトしたサーバが更なるアクションを実行しており、まだ確定レスポンス(definitive response)を持たないことを示す。サーバは、最終レスポンス(Final response)を取得するまでに200ミリ秒以上を要することが予想される場合に1xxレスポンスを送る。1xxレスポンスは信頼性を持って送信されないことに注意すること。それはクライアントがACKを送る原因にならない。暫定(1xx)レスポンスはメッセージボディ(セッション記述を含む)を含めてもよい[MAY]。

21.1.1　100 Trying (試行中)

このレスポンスは、リクエストがネクストホップサーバに受け取られており、この呼に成り代わって、特に指定されていないアクションが取られていることを示す(例: データベースが検索されている)。このレスポンスは他のすべての暫定レスポンス(provisional response)と同様に、UACによるINVITEの再送を停止させる。100(Trying)レスポンスは、ステートフルプロキシによってアップストリームに決してフォワードされないという点において他の暫定レスポンス(provisional response)と異なる。

21.1.2　180 Ringing (呼び出し中)

INVITEを受け取るUAがユーザに注意を促そうと試みている。このレスポンスはローカルのringbackを開始するために使用してもよい[MAY]。

21.1.3　181 Call Is Being Forwarded (呼がフォワードされている)

サーバは、別のデスティネーションのセットに呼がフォワードされていることを示すために、このステータスコードを使用してもよい[MAY]。

21.1.4　182 Queued (キューイングされた)

着信側パーティは一時的に電話に出られないが、サーバはその呼を拒否するのではなくキューに入れることにした。着信者が電話に出られるようになったとき、それは適切な最終ステータスレスポンスを返送する。リーズンフレーズは呼のステータスについての更なる詳細を与えてもよい[MAY]。たとえば、"5 calls queued; expected waiting time is 15 minutes" (「5つの呼がキューに入れられている。予想待ち時間は15分」)。サーバは、キューに入れられた呼のステータスについて、発信者をアップデートするためにいくつかの182 (Queued)レスポンスを発行してもよい[MAY]。

21.1.5　183 Session Progress (セッションの進捗状況)

183(Session Progress)レスポンスは、特に区別のない呼の進捗状況についての情報を伝えるために使用される。リーズンフレーズ、ヘッダフィールド、あるいはメッセージボディを、呼の進捗状況についての更なる詳細を伝えるために使用してもよい[MAY]。

21.2 **成功レスポンス**(success response)**レスポンス** 2xx

リクエストが成功した。

21.2.1 200 OK

リクエストは成功した。レスポンスで返送された情報は、リクエストで使用されたメソッドに依存する。

21.3 **リダイレクト レスポンス** 3xx

3xx レスポンスは、ユーザの新たなロケーションについての、あるいは呼を満足させることができるかもしれない代替サービスについての情報を与える。

21.3.1 300 Multiple Choices (**複数の選択肢がある**)

リクエスト中のアドレスが解決されて、各々がそれ自身の特定のロケーションを持ついくつかの選択肢が得られた。そしてユーザ(または UA)は好みのコミュニケーションエンドポイントを選び、それのリクエストをそのロケーションにリダイレクトすることができる。

このレスポンスは、Accept リクエストヘッダフィールドで許可されている場合、ユーザまたは UA が最も好ましいものを選ぶことができるように、リソースの特性と場所のリストが収められたメッセージボディを含めてもよい[MAY]。しかしながら、このメッセージボディのための MIME タイプはひとつも定義されていない。

選択肢は Contact フィールド(20.10 節)としてもリストされるべきである[SHOULD]。HTTP とは違い、SIP のレスポンスは、いくつかの Contact フィールドまたは Contact フィールド中にアドレスのリストを含んでもよい[MAY]。UA は Contact ヘッダフィールド値を自動リダイレクションに使用してもよい[MAY]。あるいは選択の確認をユーザに依頼してもよい[MAY]。しかしながら、この仕様ではそのような自動選択のためのいかなる標準も定義しない。

> 着信者に複数の異なるロケーションで到達可能で、かつ、サーバがリクエストをプロキシすることができないまたは好まない場合に、このステータスレスポンスがふさわしい。

21.3.2 301 Moved Permanently (**恒久的に移動した**)

Request-URI のアドレスでユーザをもはや見つけることができないので、リクエストを行っているクライアントは Contact ヘッダフィールド(20.10 節)で与えられた新しいアドレスで再試行するべきである[SHOULD]。リクエストをする側は、すべてのローカルディレクトリ、アドレス帳、およびユーザのロケーションのキャッシュをこの新しい値でアップデートし、以降のリクエストをリストされたアドレスにリダイレクトするべきである[SHOULD]。

21.3.3 302 Moved Temporarily (**一時的に移動した**)

リクエストを行っているクライアントは Contact ヘッダフィールド(20.10 節)で与えられた新しいアドレスでリクエストを再試行するべきである[SHOULD]。新しいリクエストの Request-URI は、レスポンスの Contact ヘッダフィールドの値を使用する。

Contact URI の有効期間は Expires(20.19 節)ヘッダフィールドまたは Contact ヘッダフィールドの expires パラメータを介して示すことができる。プロキシと UA は両方とも有効期限時間の間この URI をキャ

ッシュしてもよい[MAY]。明示的な有効期限時間がない場合、そのアドレスは再帰のために一度だけ有効であり、以降のトランザクションのためにキャッシュしてはいけない[MUST NOT]。

Contact ヘッダフィールドからキャッシュされた URI が失敗する場合、リダイレクトしたリクエストの Request-URI をもう一度だけ試してもよい[MAY]。

一時的な URI は有効期限時間よりも早く無効になってしまったのかもしれず、新しい一時アドレスが有効かもしれない。

21.3.4 305 Use Proxy (**プロキシを使用せよ**)

リクエストしたリソースは、Contact フィールドで与えられたプロキシを通してアクセスされなくてはならない[MUST]。Contact フィールドはプロキシの URI を与える。受信者はプロキシを経由してこの一つのリクエストを繰り返すことを期待されている。305(Use Proxy)レスポンスは UAS によってのみ生成されなくてはならない[MUST]。

21.3.5 380 Alternative Service (**代替サービス**)

呼は成功せずとも代替サービスは可能である。

代替サービスはレスポンスのメッセージボディで述べられている。そのようなボディのフォーマットはここで定義されず、将来の標準化の対象になるかもしれない。

21.4 **リクエスト失敗 レスポンス** 4xx

4xx レスポンスは、特定のサーバからの確定的な失敗レスポンスである。クライアントは同じリクエストを、それを修正しないで、再試行するべきではない[SHOULD NOT](例: 適切な authorization を追加する)。しかしながら、同じリクエストを別のサーバに送れば成功するかもしれない。

21.4.1 400 Bad Request (**不正なリクエスト**)

リクエストが不正な形式の構文のため理解できなかった。リーズンフレーズは更に詳しく構文の問題を識別するべきである[SHOULD]。
例: "Missing Call-ID header field" (Call-ID ヘッダフィールドが見つからない)

21.4.2 401 Unauthorized (**認可されていない**)

リクエストはユーザ認証を要求する。このレスポンスは UAS とレジストラサーバが発行する。一方、407(Proxy Authentication Required)はプロキシサーバが使用する。

21.4.3 402 Payment Required (**料金支払いが必要**)

将来の使用のために予約されている。

21.4.4 403 Forbidden (**禁止**)

サーバはリクエストを理解したが、それを実行することを拒否している。認証は役に立たないので、リクエストは繰り返されるべきではない[SHOULD NOT]。

21.4.5　404 Not Found (**見つからない**)

Request-URI で指定されたドメインにユーザが存在しないという確定的な情報をサーバが持っている。このステータスは、リクエストの受信者がハンドリングするどのドメインにも Request-URI のドメインがマッチしない場合にも返送される。

21.4.6　405 Method Not Allowed (**メソッドが許可されていない**)

Request-Line で指定されたメソッドは理解されたが、Request-URI で識別されるアドレスに対して許可されていない。

レスポンスは、示されたアドレスに対して有効なメソッドのリストを含む Allow ヘッダフィールドを含まなくてはならない[MUST]。

21.4.7　406 Not Acceptable (**受け入れできない**)

リクエストによって識別されたリソースは、リクエストで送られた Accept ヘッダフィールドによれば、受け入れ不能なコンテンツ特性を持つレスポンスエンティティを生成することしかできない。

21.4.8　407 Proxy Authentication Required (**プロキシ認証が必要**)

このコードは 401(Unauthorized)と似ているが、クライアントがそれ自身をまずはじめにプロキシで認証しなくてはならない[MUST]ことを示す。SIP のアクセス認証については 26 節と 22.3 節で説明されている。

このステータスコードは、着信者ではなくコミュニケーションチャンネル(たとえば、テレフォニーゲートウェイ)へのアクセスが認証を要求するときにアプリケーションに対して使用できる。

21.4.9　408 Request Timeout (**リクエストがタイムアウトした**)

サーバは、たとえば時間内にユーザのロケーションを確定できなかった場合など、適切な時間内にレスポンスを生成できなかった。クライアントは、後ほどいつでも、修正なしでそのリクエストを繰り返してもよい[MAY]。

21.4.10　410 Gone (**リソースが既に存在しない**)

リクエストされたリソースがそのサーバでもはや利用不可能で、フォワード先のアドレスもわからない。この状態は恒久的なものであると考えられる。その状態が恒久的なものかどうかサーバが知らない、あるいは確定するための機能を持たない場合、代わりにステータスコード 404(Not Found)が使用されるべきである[SHOULD]。

21.4.11　413 Request Entity Too Large (**リクエストのエンティティが大きすぎる**)

リクエストの entity-body が、サーバが処理をいとわない、または処理できるサイズよりも大きいので、サーバはリクエストの処理を拒否している。サーバは、クライアントがリクエストを継続するのを防ぐために、コネクションをクローズしてもよい[MAY]。

その状態が一時的である場合、サーバは、それが一時的でありどのくらいの時間経過後にクライアントが再び試行してもい[MAY]かを示すために、Retry-After ヘッダフィールドを含めるべきである[SHOULD]。

21.4.12　414 Request-URI Too Long (Request-URI **が長すぎる**)

サーバが解釈をいとわずに行う長さよりも Request-URI が長いので、サーバはリクエストを処理することを拒否している。

21.4.13　415 Unsupported Media Type (**サポートされていないメディアタイプ**)

リクエストのメッセージボディが、リクエストされたメソッドに対してそのサーバでサポートしていないフォーマットなので、サーバはリクエストを処理することを拒否している。サーバは、コンテンツの個々の問題に応じて Accept、Accept-Encoding、または Accept-Language ヘッダフィールドを使用して、受け入れ可能なフォーマットのリストを返送しなくてはならない[MUST]。このレスポンスの UAC での処理については 8.1.3.5 節で述べられている。

21.4.14　416 Unsupported URI Scheme (**サポートされていない** URI **スキーム**)

Request-URI の URI のスキームがサーバの知らないものなので、サーバはリクエストを処理できない。このレスポンスのクライアントでの処理については 8.1.3.5 節で述べられている。

21.4.15　420 Bad Extension (**不正な拡張**)

Proxy-Require(20.29 節)または Require(20.32 節)ヘッダフィールドで指定されたプロトコル拡張を、サーバは理解しなかった。サーバはレスポンスの Unsupported ヘッダフィールドに、サポートしていない拡張のリストを含めなくてはならない[MUST]。このレスポンスの UAC での処理については 8.1.3.5 節で述べられている。

21.4.16　421 Extension Required (**拡張が必要**)

UAS はリクエストを処理するために特定の拡張を必要とするが、この拡張はリクエストの Supported ヘッダフィールドにリストされていない。このステータスコードを持つレスポンスは、必要とされる拡張をリストする Require ヘッダフィールドを含めなくてはならない[MUST]。

UAS は、クライアントに対していかなる有用なサービスも本当に提供できないのでなければ、このレスポンスを使用するべきではない[SHOULD NOT]。そのかわり、望ましい拡張が Supported ヘッダフィールドにリストされていない場合は、サーバはベースライン SIP 能力とクライアントがサポートしている拡張を使用してリクエストを処理するべきである[SHOULD]。

21.4.17　423 Interval Too Brief (**間隔が短すぎる**)

リクエストによってリフレッシュされたリソースの有効期限時間が短すぎるため、サーバはリクエストを拒否している。このレスポンスは、Contact ヘッダフィールドの有効期限時間が短すぎた登録を拒否するためにレジストラサーバが使用できる。このレスポンスとそれに関連する Min-Expires ヘッダフィールドの使用については 10.2.8、10.3、および 20.23 節で述べられている。

21.4.18　480 Temporarily Unavailable (**一時的に利用不可**)

着信者のエンドシステムにうまくコンタクトしたが、着信者は現在電話を受けられない(たとえば、ログインしていない、着信者とのコミュニケーションを妨げる方法でログインしている、または「取り込み中」機能を作動させている)。このレスポンスは、Retry-After ヘッダフィールドで電話をかけるのに都合のいい時間を示してもよい[MAY]。そのユーザは別の場所でも電話を受けられるということがありうる(このサーバの知らないうちに)。リーズンフレーズは、なぜ着信者が電話に出られないのかについてのより明確な理由

を示すべきである[SHOULD]。この値はUAによって設定可能であるべきである[SHOULD]。ステータス486(Busy Here)を、呼の失敗についての特定の理由をより明確に示すために使用してもよい[MAY]。

このステータスは、Request-URIで識別されるユーザを認識するがそのユーザに対する有効なフォワードロケーションを現在持っていないリダイレクトサーバまたはプロキシサーバによっても返送される。

21.4.19 481 Call/Transaction Does Not Exist(**呼またはトランザクションが存在しない**)

このステータスは、既存のどのダイアログやトランザクションにもマッチしないリクエストをUASが受け取ったことを示す。

21.4.20 482 Loop Detected (**ループが検知された**)

サーバがループを検知した(16.3節の項目4)。

21.4.21 483 Too Many Hops (**ホップが多すぎる**)

サーバが値ゼロのMax-Forwardsヘッダフィールド(20.22節)を含むリクエストを受け取った。

21.4.22 484 Address Incomplete (**アドレスが不完全**)

サーバが不完全なRequest-URIを持つリクエストを受け取った。付加情報がリーズンフレーズで提供されるべきである[SHOULD]。

このステータスコードは重複ダイアリング(overlap dialing)を可能にする。重複ダイアリングでは、クライアントはダイアル文字列の長さを知らない。クライアントはユーザに更なる入力を促しながら、484(Address Incomplete)ステータスレスポンスを受け取らなくなるまで徐々に長くなる文字列を送る。

21.4.23 485 Ambiguous (**不明瞭**)

Request-URIが不明瞭だった。レスポンスはContactヘッダフィールドに、可能性がある不明瞭ではないアドレスのリストを含めてもよい[MAY]。選択肢を明らかにすることでユーザまたは組織のプライバシーを犯すことがある。不明瞭なRequest-URIに対して、404(Not Found)でレスポンスするかまたは可能性のある選択肢のリストを隠すようにサーバを設定することが可能でなくてはならない[MUST]。

以下はRequest-URI(sip:lee@example.com)を持つリクエストに対するレスポンスの例である。

```
SIP/2.0 485 Ambiguous
Contact: Carol Lee <sip:carol.lee@example.com>
Contact: Ping Lee <sip:p.lee@example.com>
Contact: Lee M. Foote <sips:lee.foote@example.com>
```

いくつかのEメールシステムやボイスメールシステムはこの機能を提供する。意味合い(semantics)が異なるので3xxと区別してステータスコードは使用される。300では、提供された選択肢で同じ人やサービスに到達すると推測される。3xxレスポンスに対しては自動選択やシーケンシャルサーチが意味をなすのに対して、485(Ambiguous)レスポンスに対してはユーザの介入が要求される。

21.4.24　486 Busy Here (**ここは現在ビジー**)

着信者のエンドシステムにうまくコンタクトしたが、着信者は現在このエンドシステムで別の呼を受け取りたいと思っていないか受け取れない。このレスポンスは、Retry-After ヘッダフィールドで電話をかけるのに都合のいい時間を示してもよい[MAY]。そのユーザは、たとえばボイスメールサービスを介して、別の場所でも電話を受けられるということがありうる。この呼を他のどのエンドシステムも受け入れることができないことをクライアントが知っている場合、ステータス 600(Busy Everywhere)が使用されるべきである[SHOULD]。

21.4.25　487 Request Terminated (**リクエストが終了させられた**)

リクエストは BYE または CANCEL リクエストで終了させられた。このレスポンスは CANCEL リクエスト自身に対しては決して返送されない。

21.4.26　488 Not Acceptable Here (**ここでは受け入れ不能**)

このレスポンスは 606(Not Acceptable)と同じ意味を持つが、Request-URI でアドレス指定された特定のリソースに対してのみ適用されるので、リクエストは他の場所で成功するかもしれない。

OPTIONS リクエストに対する 200(OK)レスポンスのメッセージボディと同じように、INVITE の Accept ヘッダフィールド(または存在しなければ application/sdp)にしたがってフォーマットされた、メディア能力の記述を含むメッセージボディがレスポンス中に存在してもよい[MAY]。

21.4.27　491 Request Pending (**リクエストペンディング**)

リクエストは、同じダイアログ内に保留中のリクエストを持つ UAS に受け取られた。14.2 節で、このようなグレア(glare)な状況をどのように解決するかについて述べる。

21.4.28　493 Undecipherable (**解読不能**)

リクエストを UAS が受け取った。そのリクエストにはその受信者が適切な暗号解読鍵を所持しないかあるいは提供しない、暗号化された MIME ボディを含んでいた。このレスポンスは、この UA に送られる MIME ボディを暗号化するために使用されるべき適切な公開鍵を含むひとつのボディを持ってもよい[MAY]。このレスポンスコードの用法の詳細は 23.2 節で述べられている。

21.5　**サーバでの失敗 レスポンス** 5xx

5xx レスポンスは、サーバ自身がエラーを起こしたときに与えられる失敗レスポンスである。

21.5.1　500 Server Internal Error (**サーバ内部エラー**)

サーバがリクエストを遂行することを妨げる予期しない状態に遭遇した。クライアントは特定のエラー状態を表示してもよい[MAY]、数秒後にリクエストを再試行してもよい[MAY]。

その状態が一時的なものである場合、サーバは Retry-After ヘッダフィールドを使用して、クライアントがいつリクエストを再試行できるか示してもよい[MAY]。

21.5.2　501 Not Implemented (**実装されていない**)

サーバはリクエストを遂行するために必要とされる機能をサポートしていない。これは、UAS がリクエストメソッドを認識せず、いかなるユーザに対してもそれをサポートできないときに適切なレスポンスである。

(プロキシはメソッドに依らずすべてのリクエストをフォワードする)

サーバがリクエストメソッドは認識するがそのメソッドが許可されていないかサポートされていないときには、405(Method Not Allowed)が送られることに注意すること。

21.5.3 502 Bad Gateway (**不正なゲートウェイ**)

サーバが、ゲートウェイまたはプロキシとして動作している間に、リクエストの遂行を試みてアクセスしたダウンストリームサーバから無効なレスポンスを受け取った。

21.5.4 503 Service Unavailable (**サービスを利用できない**)

サーバは、一時的な過負荷またはメンテナンスのため、一時的にリクエストを処理できない。サーバは、Retry-After ヘッダフィールドでクライアントがいつリクエストを再試行するべきか示してもよい[MAY]。Retry-After が与えられない場合、クライアントは 500(Server Internal Error)レスポンスを受け取ったかのように動作しなくてはならない[MUST]。

503(Service Unavailable)を受け取るクライアント(プロキシまたは UAC)は、代替のサーバにリクエストのフォワードを試みるべきである[SHOULD]。それは、(もし存在すれば)Retry-After ヘッダフィールドで指定された期間、そのサーバに他のどのようなリクエストもフォワードするべきではない[SHOULD NOT]。

サーバは 503(Service Unavailable)でレスポンスする代わりに、コネクションを拒否するかリクエストを取りやめてもよい[MAY]。

21.5.5 504 Server Time-out (**サーバタイムアウト**)

サーバは、リクエストの処理を試みてアクセスした外部サーバからタイムリーなレスポンスを受け取らなかった。Expires ヘッダフィールドで指定した期間内にアップストリームサーバからレスポンスがない場合、408(Request Timeout)が代わりに使用されるべきである。

21.5.6 505 Version Not Supported (**サポートされていないバージョン**)

サーバはリクエストで使用された SIP プロトコルのバージョンをサポートしていないかサポートを拒否している。サーバは、クライアントとして同じメジャーバージョンを使用するリクエストを、このエラーメッセージ以外では完了できないか完了するつもりがないことを示している。

21.5.7 513 Message Too Large (**メッセージが大きすぎる**)

メッセージ長がサーバの処理能力を超えたので、サーバはリクエストを処理できなかった。

21.6 **グローバル失敗 レスポンス** 6xx

6xx レスポンスは、サーバが、Request-URI で示された特定のインスタンスだけではなく特定のユーザについての確定的な情報を持つことを示す。

21.6.1 600 Busy Everywhere (**どの場所もビジー**)

着信者のエンドシステムにうまくコンタクトしたが、着信者はビジーで今は呼を受けることを望んでいない。このレスポンスは、Retry-After ヘッダフィールドで電話をかけるのに都合のいい時間を示してもよい[MAY]。着信者が呼を断る理由を明かすことを望まない場合、着信者は代わりにステータスコード

603(Decline)を使用する。このステータスレスポンスは、他のどのエンドポイント(たとえばボイスメールシステム)もそのリクエストに答えないことをクライアントが知っている場合にのみ返送される。そうでなければ、486(Busy Here)が返送されるべきである。

21.6.2　603 Decline (辞退)

着信者のマシンにうまくコンタクトしたが、ユーザが明らかに参加することを望まないか参加できない。このレスポンスはRetry-Afterヘッダフィールドで電話をかけるのに都合のいい時間を示してもよい[MAY]。このステータスレスポンスは、他のどのエンドポイントもそのリクエストに答えないことをクライアントが知っている場合にのみ返送される。

21.6.3　604 Does Not Exist Anywhere (どこにも存在しない)

Request-URI で示されたユーザがどこにも存在しないという信頼できる情報をサーバが持っている。

21.6.4　606 Not Acceptable (受け入れ不能)

ユーザのエージェントにうまくコンタクトしたが、たとえばリクエストしたメディア、帯域、またはアドレッシングスタイルのようなセッション記述の、何らかの側面が受け入れ不可能だった。

606(Not Acceptable)レスポンスは、ユーザはコミュニケーションを望んでいるが記述されたセッションを十分にサポートできないことを意味する。606(Not Acceptable)レスポンスは、記述されたセッションをなぜサポートできないのかを述べる理由のリストを Warning ヘッダフィールドに含んでもよい[MAY]。Warning の理由コードは 20.43 節にリストされている。

OPTIONS リクエストに対する 200(OK)レスポンスのメッセージボディと同じように、INVITE の Accept ヘッダフィールド(または存在しなければ application/sdp)にしたがってフォーマットされた、メディア能力の記述を含むメッセージボディがレスポンス中に存在してもよい[MAY]。

ネゴシエーションが頻繁に必要とされることがないことが望まれるので、新規ユーザが既存のカンファレンスへの参加を招待されるときにはネゴシエーションが可能ではないかもしれない。606(Not Acceptable)レスポンスに対処するかどうか決定するのは招待の起動者次第である。

このステータスレスポンスは、他のどのエンドポイントもそのリクエストに答えないことをクライアントが知っている場合にのみ返送される。

２２．HTTP 認証の使用法 (Usage of HTTP Authentication)

SIP は HTTP の認証に基づく認証のためのステートレスなチャレンジベースのメカニズムを提供する。プロキシサーバまたは UA がリクエスト(22.1 節で与えられることを例外とする)を受け取るといつでも、それは起動者のアイデンティティの保証を提供するためにリクエストの起動者をチャレンジしてもよい[MAY]。開始者が識別されるとすぐにリクエストの受信者は、このユーザが当該のリクエストをするために認可されているかどうか確認するべきである[SHOULD]。このドキュメントではどの認可システムも推奨もしくは議論しない。

本節で述べられる「ダイジェスト」認証メカニズムは、メッセージの完全性や機密性がない、メッセージ認証とリプレイ防御のみを提供する。能動的な攻撃者が SIP リクエストおよびレスポンスを修正することを

防ぐために、ダイジェスト認証で提供される上記の防御手段およびそれ以上の防御手段がとられる必要がある。

「ベーシック」認証はその貧弱なセキュリティのために、使用が反対されているということに注意すること。サーバは「ベーシック」認可スキームを使用する信用証明書を受け入れてはいけない[MUST NOT]。また、サーバは「ベーシック」でチャレンジしてもいけない[MUST NOT]。これは RFC2543 からの変更である。

22.1 フレームワーク (Framework)

SIP 認証のフレームワークは HTTP のもの(RFC2617 参考文献[17])と非常に似通っている。特に、auth-scheme、auth-param、challenge、realm、realm-value、および credentials の BNF は同一である(スキームとして「Basic」を使用することは許可されていないが)。SIP では、UAS は UAC のアイデンティティをチャレンジするために 401(Unauthorized)レスポンスを使用する。それに加えて、レジストラサーバとリダイレクトサーバは認証のために 401(Unauthorized)レスポンスを使用してもよい[MAY]、プロキシは使用してはいけなく[MUST NOT]、その代わりに 407(Proxy Authentication Required)レスポンスを使用してもよい[MAY]。様々なメッセージに Proxy-Authenticate、Proxy-Authorization、WWW-Authenticate、および Authorization を含める要求は、RFC2617(参考文献[17])に述べられているものと同じである。

SIP は標準化したルート URL(canonical root URL)の概念を持たないので、防御空間(protection space)の考えは SIP では異なって解釈される。realm 文字列だけで防御ドメイン(protection domain)を定義する。これは Request-URI と realm が共に防御ドメインを定義していた RFC2543 からの変更である。

> 防御ドメインの以前のこの定義は、UAC が送った Request-URI とチャレンジしているサーバが受け取った Request-URI が異なるかも知れず、実際に、UAC は最終的な Request-URI の形を知らないかもしれないので、若干の混乱を生じた。また、以前の定義は Request-URI 中の SIP URI の存在に依存しており、代替の URI スキーム(たとえば、tel URL)を考慮に入れていないように思えた。

受け取ったリクエストを認証するユーザエージェントやプロキシサーバのオペレータは、オペレータのサーバのための realm 文字列の生成のための以下のガイドラインを遵守しなくてはならない[MUST]。

- realm 文字列はグローバルに一意でなくてはならない[MUST]。realm 文字列が RFC2617(参考文献[17])の 3.2.1 節の推奨にしたがってホスト名もしくはドメイン名を含むことが推奨される[RECOMMENDED]。

- realm 文字列はユーザに対して表示することができる、人間が読むことのできる識別子を提示するべきである[SHOULD]。

例:

```
INVITE sip:bob@biloxi.com SIP/2.0
Authorization: Digest realm="biloxi.com", <...>
```

一般的に、SIP 認証は特定の realm(防御ドメイン)に対して意味を持つ。したがって、ダイジェスト認証では、そのような各防御ドメインはそれ自身のユーザ名とパスワードのセットを持つ。サーバが特定のリク

エストに対して認証を要求しない場合、それはパスワードを持たない(""というパスワード)デフォルトのユーザ名である「anonymous」を受け入れてもよい[MAY]。同様に、多くのユーザを代表する PSTN ゲートウェイのような UAC は、個々のユーザのアカウントではなく、それの realm のためにそれ自身のデバイス固有のユーザ名とパスワードを持ってもよい[MAY]。

サーバはほとんどの SIP リクエストに合理的にチャレンジできるが、認証に対して特別なハンドリングを必要とする、このドキュメントで定義された 2 つのリクエストがある。それは ACK と CANCEL である。

(ダイジェスト認証のように)nonce を計算するために使用される値を伝えるためにレスポンスを用いる認証スキームの下では、ACK を含め、レスポンスを必要としないすべてのリクエストでは問題が発生する。この理由により、サーバに受け入れられた INVITE 中のいかなる信用証明書も、ACK のためにそのサーバで受け入れられなくてはならない[MUST]。ACK メッセージを生成する UAC は、ACK が対応する INVITE 中に現れたすべての Authorization と Proxy-Authorization ヘッダフィールド値を複写する。サーバは ACK に対してチャレンジを行ってはいけない[MUST NOT]。

CANCEL メソッドはレスポンス(2xx)を必要とするが、CANCEL リクエストは再提出されることができないので、サーバは CANCEL リクエストに対してチャレンジを試みてはいけない[MUST NOT]。一般的に、CANCEL リクエストは、キャンセルされることになるリクエストを送ったのと同じホップから来る場合、サーバに受け入れられるべきである[SHOULD](ある種のトランスポートレイヤまたはネットワークレイヤのセキュリティアソシエーション(26.2.1 節で述べられているように)が備えられているという条件で)。

UAC がチャレンジを受け取るとき、UAC デバイスが当該の realm のための信用証明書をまだ知らない場合は、UAC は(WWW-Authenticate ヘッダフィールドまたは Proxy-Authenticate ヘッダフィールドのいずれかに現れる)チャレンジの realm パラメータのコンテンツをユーザに表示するべきである[SHOULD]。UA にそれの realm のための信用証明書をあらかじめ設定するサービスプロバイダは、あらかじめ設定されたデバイスでチャレンジされるときに、ユーザがこの realm のためにユーザ自身の信用証明書を提示する機会を持てないということに注意するべきである。

最後に、UAC が適切な realm に関連付けられた信用証明書の場所を見つけることができたとしても、この信用証明書はもはや有効でないかもしれない、あるいはチャレンジしているサーバが何らかの理由で(特にパスワードなしの「anonymous」が提出されたとき)この信用証明書を受け入れない可能性が存在することに注意すること。この場合には、サーバはチャレンジを繰り返すか、または 403(Forbidden)でレスポンスすることができる。UAC は拒否されたばかりの信用証明書を持つリクエストを再度試みてはいけない[MUST NOT](しかし、nonce が古くなっていた場合は、リクエストを再試行できる)。

22.2 **ユーザ・ユーザ間認証** (User-to-User Authentication)

UAS が UAC からリクエストを受け取るとき、UAS はリクエストが処理される前に発信元を認証してもよい[MAY]。信用証明書がリクエスト中に(Authorization ヘッダフィールドに)提供されていない場合、UAS はそのリクエストを 401(Unauthorized)ステータスコードで拒否することで、信用証明書を提供するように発信元にチャレンジできる。

レスポンスの WWW-Authenticate ヘッダフィールドが 401(Unauthorized)レスポンスメッセージに含まれなくてはならない[MUST]。フィールド値は認証スキームを示す少なくとも一つのチャレンジとその realm に適

用可能なパラメータからなる。

以下は 401 チャレンジの WWW-Authenticate ヘッダフィールドの例である。

```
WWW-Authenticate: Digest
        realm="biloxi.com",
        qop="auth,auth-int",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

発信元の UAC が 401(Unauthorized)を受け取るとき、それは可能であれば、適切な信用証明書を持つリクエストを再度発信するべきである[SHOULD]。UAC は処理を進める前に発信元のユーザからの入力を必要とするかもしれない。認証の信用証明書が(ユーザによって直接、あるいは内部の鍵束(keyring)から探し出して)供給されるとすぐに、UA はその信用証明書を、与えられた To ヘッダフィールド値と realm のためにキャッシュして、これらの値をそのデスティネーションへの次のリクエストに再利用することを試みるべきである[SHOULD]。UA はそれが望むどのような方法ででも信用証明書をキャッシュしてもよい[MAY]。

realm に対する信用証明書の位置を特定できない場合、UAC はユーザ名「anonymous」、パスワードなし(""というパスワード)でリクエストの再試行を試みてもよい[MAY]。

信用証明書の位置が特定されるとすぐに、それ自身を UAS またはレジストラサーバで認証することを望むいかなる UA も、(必ずしもそうではないが、通常は 401(Unauthorized)レスポンスを受け取った後に)リクエストに Authorization ヘッダフィールドを含めることでそれを可能にしてもよい[MAY]。Authorization フィールド値は、リクエストされるリソースの realm のための UA の認証情報を含む信用証明書、および認証とリプレイ防御のサポートに必要とされるパラメータからなる。

以下は Authorization ヘッダフィールドの例である。

```
Authorization: Digest username="bob",
        realm="biloxi.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="sip:bob@biloxi.com",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

UAC が 401(Unauthorized)レスポンスまたは 407(Proxy Authentication Required)レスポンスの受信後に信用証明書を持つリクエストを再提出するとき、UAC はアップデートしたリクエストを送るときに通常行うように、CSeq ヘッダフィールド値をインクリメントしなくてはならない[MUST]。

22.3 **プロキシ・ユーザ間認証** (Proxy-to-User Authentication)

同様に、UAC がプロキシサーバにリクエストを送るとき、プロキシサーバはリクエストが処理される前に発信元を認証してもよい[MAY]。信用証明書がリクエスト中に(Proxy-Authorization ヘッダフィールドに)提供されていない場合、プロキシはそのリクエストを 407(Proxy Authentication Required)ステータスコードで拒否することで、信用証明書を提供するように発信元にチャレンジできる。プロキシは、リクエストされたリソースのためにそのプロキシに適用可能な Proxy-Authenticate 値を持つ 407(Proxy Authentication Required)メッセージを埋め込まなくてはならない[MUST]。

参考文献[17]で述べられている Proxy-Authenticate および Proxy-Authorization を並行して使用する方法には一つの相違点がある。プロキシは Proxy-Authorization ヘッダフィールドに値を追加してはいけない[MUST NOT]。すべての 407(Proxy Authentication Required)レスポンスは、他のすべてのレスポンスのための手順と同じ手順にしたがって、UAC に向けてアップストリームにフォワードされなくてはならない[MUST]。認証を依頼したプロキシの realm のための信用証明書を含む Proxy-Authorization ヘッダフィールド値を追加することは UAC の義務である。

> プロキシが Proxy-Authorization ヘッダフィールド値を追加してリクエストを再提出した場合、プロキシは新しいリクエストの CSeq をインクリメントする必要がある。しかしながらこれは、CSeq 値が異なるために、オリジナルリクエストを提出した UAC に UAS からのレスポンスを破棄させる原因になる。

発信元の UAC が 407(Proxy Authentication Required)を受け取るとき、それは可能であれば、適切な信用証明書を持つリクエストを再発信するべきである[SHOULD]。それは、401 に対してレスポンスするために上で与えられた realm パラメータの表示手順と同じ手順に従うべきである。

realm に対する信用証明書の位置を特定できない場合、UAC はユーザ名「anonymous」、パスワードなし(""というパスワード)でリクエストの再試行を試みてもよい[MAY]。

UAC はまた、再発信したリクエストで使用した信用証明書をキャッシュするべきである[SHOULD]。

プロキシの信用証明書キャッシュのために、以下のルールが推奨される[RECOMMENDED]。

UA が特定の Call-ID を持つリクエストに対する 401/407 レスポンスで Proxy-Authenticate ヘッダフィールド値を受け取る場合、UA はそれと同じ Call-ID を含む今後のすべてのリクエストにその realm のための信用証明書を組み込むべきである。これらの信用証明書は複数のダイアログに渡ってキャッシュされてはいけない[MUST NOT]。しかしながら、アウトバウンドプロキシが存在するときに、UA がローカルアウトバウンドプロキシの realm を設定されている場合、UA は複数のダイアログに渡ってその realm のための信用証明書をキャッシュしてもよい[MAY]。これは、ダイアログ内の以降のリクエストが、Route ヘッダのパスのどのプロキシでも必要とされない信用証明書を含むことがあり得ることを意味するということに注意すること。

プロキシサーバに対してそれ自身を認証することを望むいかなる UA も、(必ずしもそうではないが、通常は 407(Proxy Authentication Required)レスポンスを受け取った後に)リクエストに Proxy-Authorization ヘッダフィールド値を含めることでそれを可能にしてもよい[MAY]。リクエストの Proxy-Authorization ヘッダフィールド値は、認証を求めるプロキシに対してクライアント自身(あるいはそれのユーザ)を識別することを可能にする。Proxy-Authorization ヘッダフィールド値は、そのプロキシに対する UA の認証情報を含

んでいる信用証明書および(または)リクエストされるリソースの realm からなる。

Proxy-Authorization ヘッダフィールド値は、realm が realm パラメータで識別されるプロキシに対してのみ適用される(このプロキシは以前に Proxy-Authenticate フィールドを使用して認証を要求していたのかもしれない)。複数のプロキシが連鎖的に使用されるとき、realm が Proxy-Authorization ヘッダフィールド値で指定された realm パラメータにマッチしないいかなるプロキシもその値を破壊してはいけない[MUST NOT]。

realm をサポートしない認証スキームが Proxy-Authorization ヘッダフィールドで使用される場合、プロキシサーバは Proxy-Authorization ヘッダフィールドの中のどれかの値がそのプロキシサーバが有効な信用証明書であるとみなす値を持つかどうかを確定するために、Proxy-Authorization ヘッダフィールドのすべての値のパースを試みなくてはならない[MUST]。これは大きいネットワークでは非常に時間を消費する可能性があるので、プロキシサーバは Proxy-Authorization ヘッダフィールドで realm をサポートする認証スキームを使用するべきである[SHOULD]。

リクエストが(16.7 節で述べられているように)フォークされる場合、さまざまなプロキシサーバおよび(または)UA が UAC にチャレンジすることを望むかもしれない。この場合、フォークを行うプロキシサーバはこれらのチャレンジを一つのレスポンスにまとめる責任を負う。フォークされたリクエストに対するレスポンスで受け取った WWW-Authenticate および Proxy-Authenticate のそれぞれの値は、フォークを行ったプロキシによって UA に送られる一つのレスポンスの中に置かれなくてはならない[MUST]。これらのヘッダフィールド値の並び順は重要ではない。

> プロキシサーバがリクエストに答えてチャレンジを発行するとき、それは UAC が有効な信用証明書を持つリクエストで再試行するまで、リクエストをプロキシしない。フォークを行うプロキシは、認証を求める複数のプロキシサーバに同時にリクエストをフォワードするかもしれない。リクエストをフォワードされたそれぞれのプロキシは同様に、発信元の UAC が各プロキシの realm で認証されるまでそのリクエストをフォワードしない。UAC が各チャレンジに対して信用証明書を提供しない場合、チャレンジを発行したプロキシサーバはデスティネーションユーザの位置が特定されるかもしれない UA にリクエストをフォワードしない。したがって、フォークの長所の大部分は失われる。

複数のチャレンジを含む 401(Unauthorized)または 407(Proxy Authentication Required)に答えてリクエストを再提出するとき、UAC は、UAC が信用証明書を供給することを望む各 WWW-Authenticate 値に対する Authorization 値と各 Proxy-Authenticate 値に対する Proxy-Authorization 値を含んでもよい[MAY]。上述のようにリクエスト中の複数の信用証明書は、realm パラメータで区別されるべきである[SHOULD]。

同じ realm に関連付けられた複数のチャレンジが一つの 401(Unauthorized)または 407(Proxy Authentication Required)に現れることが可能である。これはたとえば、同じ管理ドメイン内の複数のプロキシ(共通の realm を使用する)に、フォークされたリクエストが到達するときに起こりうる。したがって、リクエストを再試行するとき、UAC は、同じ realm パラメータ値を持つ複数の信用証明書を Authorization ヘッダフィールドまたは Proxy-Authorization ヘッダフィールドに供給してもよい[MAY]。同じ realm に対しては同じ信用証明書が使用されるべきである[SHOULD]。

22.4 ダイジェスト認証スキーム (The Digest Authentication Scheme)

本節は、HTTP ダイジェスト認証スキームを SIP に適用するために必要とされる修正点と明確化について述べる。SIP スキームの使用法はほぼ完全に HTTP(参考文献[17])のものと同じである。

RFC2543 は RFC2069(参考文献[39])で定義されている HTTP ダイジェストに基づいているので、RFC2617 をサポートする SIP サーバは RFC2069 と下位互換があることを保証しなくてはならない[MUST]。この下位互換のための手順は RFC2617 で規定されている。しかしながら、SIP サーバはベーシック認証を受け入れたり要求したりしてはいけない[MUST NOT]ことに注意すること。

ダイジェスト認証のルールは、以下に述べる相違点と「HTTP/1.1」を「SIP/2.0」に置き換えることを除けば、参考文献[17]で定義されているものに従う。

1. チャレンジに含まれる URI は以下の BNF を持つ。

   URI = SIP-URI / SIPS-URI

2. RFC2617 の BNF は、HTTP ダイジェスト認証のための Authorization ヘッダフィールドの uri パラメータが引用符に囲まれていないという誤りを持つ。(RFC2617 の 3.5 節の例は正しい。)SIP では、uri は引用符に囲まれていなくてはならない[MUST]。

3. digest-uri-value の BNF は以下である。

   digest-uri-value = Request-URI ; 25 節で定義されているものと同じ

4. Etag に基づいて nonce を選択する手順の例は SIP ではうまくいかない。

5. キャッシュオペレーションに関する RFC2617(参考文献[17])の文章は SIP に適用しない。

6. RFC2617(参考文献[17])は、サーバが、リクエスト行(request line)の URI と Authorization ヘッダフィールドに含まれる URI が同じリソースを指しているかチェックすることを要求する。SIP のコンテキストでは、いくつかのプロキシでのフォワードのために、これら 2 つの URI は異なるユーザを参照するかもしれない。そのため SIP では、サーバは、Authorization ヘッダフィールド値の Request-URI がユーザ(そのユーザに対するフォワードされた、または直接の呼を、サーバは受け入れる)に対応する事をチェックしてもよい[MAY]が、それら 2 つのフィールドが等価でないことは必ずしも失敗ではない。

7. ダイジェスト認証スキームにおけるメッセージ完全性を保証するための A2 の計算を明確にするためとして、実装者は、entity-body が空のときは(すなわち、SIP メッセージがボディを持たないとき)、entity-body のハッシュが解決されると空文字列の MD5 ハッシュになると仮定するべきである。すなわち以下のようになる。

   H(entity-body) = MD5("") = "d41d8cd98f00b204e9800998ecf8427e"

8．RFC2617 では、qop コマンド(directive)が送られていなければ、cnonce 値が Authorization ヘッダフィールドで(およびその延長として Proxy-Authorization によって)送られてはいけない[MUST NOT]ということに特に言及している。したがって、cnonce に依存するどのようなアルゴリズムも(「MD5-Sess」を含む)qop コマンド(directive)が送られることを要求する。「qop」パラメータの使用は、RFC2069 との下位互換のために RFC2617 ではオプションである。RFC2543 は RFC2069 が基になったので、クライアントやサーバが受け取ることに対して「qop」パラメータは残念ながらオプションのままとしなければならない。しかしながら、サーバは WWW-Authenticate ヘッダフィールド値と Proxy-Authenticate ヘッダフィールド値で常に「qop」パラメータを送らなくてはならない[MUST]。クライアントがチャレンジのヘッダフィールドで「qop」パラメータを受け取る場合は、その結果として生じるすべての authorization ヘッダフィールドでその「qop」パラメータを送らなくてはならない[MUST]。

RFC2543 は Authentication-Info ヘッダフィールドの使用を許可しなかった(RFC2543 は RFC2069 を事実上使用していた)。しかしながら、これはボディ上の完全性チェックと相互認証を提供するので、これからはこのヘッダフィールドの使用を許可する。RFC2617(参考文献[17])は、リクエストで qop 属性を使用した下位互換のためのメカニズムを定義する。これらのメカニズムは、クライアントが RFC2069 で規定されていなかった RFC2617 の新しいメカニズムをサポートするかどうか確定するためにサーバによって使用されなくてはならない[MUST]。

２３．S/MIME

SIP メッセージは MIME ボディを伝える。そして MIME スタンダードは完全性と機密性の双方を確保するために MIME コンテンツを保護するためのメカニズムを包含する(「multipart/signed」と「application/pkcs7-mime」という MIME タイプを含む。RFC1847(参考文献[22])、RFC2630(参考文献[23])、および RFC2633(参考文献[24])参照)。しかしながら、実装者は、SIP メッセージのボディ(特に SDP)を見たり修正したりすることに頼るネットワーク中継媒体(通常のプロキシサーバではない)がまれに存在し、セキュア MIME はこの種の中継媒体が機能することを妨げるかもしれない、ということに注意する必要がある。

これは特に、特定のタイプのファイアウォールに当てはまる。

RFC2543 で述べられている、SIP メッセージのヘッダフィールドおよびボディを暗号化するための PGP メカニズムは、反対されている。

23.1　S/MIME の証明書 (S/MIME Certificates)

S/MIME の目的のためにエンドユーザを識別するために使用される証明書は、サーバによって使用されるものと一つの重要な点が異なる。所有者のアイデンティティが特定のホスト名に対応することを表明するのではなく、これらの証明書は所有者がエンドユーザのアドレスで識別されることを表明する。このアドレスは SIP URI または SIPS URI の「userinfo」、「@」、および「domainname」部分を結合して構成される(言い換えると、bob@biloxy.com 形式の E メールアドレス)。これは通常、ユーザの address-of-record(AOR)に対応する。

これらの証明書は、SIP メッセージのボディに署名したり暗号化したりするために使用される鍵にも関連付けられる。ボディは送信者(送信者は必要に応じてメッセージとともに公開鍵を含めるかもしれない)の秘密鍵で署名されるが、ボディは目的とする受信者の公開鍵で暗号化される。明らかに、送信者はメッセージ

ボディを暗号化するために受信者の公開鍵をあらかじめ知っていなければならない。公開鍵は UA のバーチャルな鍵束に保存できる。

S/MIME をサポートする各ユーザエージェントは、エンドユーザの証明書に限定した鍵束を持たなくてはならない[MUST]。この鍵束は、address-of-record(AOR)とそれに対応する証明書の間でマッピングを行うべきである。長い時間の間に、ユーザは、シグナリングの発信元の URI(From ヘッダフィールド)に同じ address-of-record(AOR)を埋め込むときは同じ証明書を使用するべきである[SHOULD]。

エンドユーザの証明書の存在に依存するいかなるメカニズムも、エンドユーザアプリケーションに証明書を提供する一元化された機関が今日ほとんど存在しないので、非常に制限される。しかしながら、ユーザは既知の公共の認証機関から証明書を取得するべきである[SHOULD]。代替として、ユーザは自己署名証明書を生成してもよい[MAY]。自己署名証明書の意味するところは、26.4.2 節でさらに検討される。実装は、設置時にあらかじめ設定された証明書も使用できる。その中には、すべての SIP エンティティ間に存在する以前の信頼関係が存在する。エンドユーザの証明書を取得する問題に加えて、エンドユーザの証明書を配布するよく知られた集中化されたディレクトリはほとんどない。しかしながら、証明書の所有者は必要に応じて公共のディレクトリに証明書を公開するべきである[SHOULD]。同様に、UAC は、公共のディレクトリで発見された SIP リクエストのターゲット URI に対応する証明書を(手動または自動で)インポートするためのメカニズムをサポートするべきである[SHOULD]。

23.2 S/MIME の鍵交換 (S/MIME Key Exchange)

SIP それ自身は、以下の方法で公開鍵を配布する手段としても利用され得る。

SIP のために S/MIME で CMS SignedData メッセージが使用されるときはいつでも、それは署名を検証するために必要な公開鍵を持つ証明書を含まなくてはならない[MUST]。

UAC がダイアログを開始する S/MIME ボディを含むリクエストを送るとき、あるいはダイアログのコンテキスト外で非 INVITE リクエストを送るときは、ボディを S/MIME'multipart/signed' CMS SignedData のボディとして構築するべきである[SHOULD]。希望する CMS サービスが EnvelopedData (そしてターゲットユーザの公開鍵が既知)の場合、UAC は SignedData メッセージの中にカプセル化された EnvelopedData メッセージを送るべきである[SHOULD]。

UAS が証明書を含む S/MIME CMS ボディを持つリクエストを受け取るとき、UAS は最初に、可能であれば認証局のための利用可能なすべてのルート証明書とともに、証明書を検証するべきである[SHOULD]。UAS はまた、証明書のサブジェクトを確定(S/MIME では、SubjectAltName が適切なアイデンティティを含む)して、この値をリクエストの From ヘッダフィールドと比較するべきである[SHOULD]。証明書が自己署名されているか未知の機関によって署名されているためにそれを検証できない場合、あるいは検証可能だがそれのサブジェクトがリクエストの From ヘッダフィールドに一致しない場合、UAS はそれのユーザに証明書のステータス(証明書のサブジェクト、それの署名者、およびすべての鍵指紋情報を含む)を通知して、処理を進める前に明示的な許可を求めなくてはならない[MUST]。証明書がうまく検証できて証明書のサブジェクトが SIP リクエストの From ヘッダフィールドに一致する場合、あるいはユーザが(通知のあとに)明示的にその証明書の使用を認める場合、UAS はこの証明書を、その証明書の所持者の address-of-record(AOR)でインデックス付けして、ローカルの鍵束に追加するべきである[SHOULD]。

UAS が、ダイアログ中の最初のリクエストに答える S/MIME ボディを含むレスポンス、あるいはダイアログのコンテキスト外の非 INVITE リクエストに対するレスポンスを送るとき、UAS は S/MIME 'multipart/signed' CMS SignedData ボディとしてそのボディを構築するべきである[SHOULD]。希望する CMS サービスが EnvelopedData の場合、UAS は SignedData メッセージの中にカプセル化された EnvelopedData メッセージを送るべきである[SHOULD]。

UAC が証明書を含む S/MIME CMS ボディを持つレスポンスを受け取るとき、UAC は最初に、可能であれば利用可能なすべてのルート証明書とともに、証明書を検証するべきである[SHOULD]。UAC はまた、証明書のサブジェクトを確定して、この値をレスポンスの To フィールドと比較するべきである[SHOULD]。その 2 つは明確に異なるかもしれないが、これは必ずしもセキュリティの侵害を示すものではない。証明書が自己署名されているか未知の機関によって署名されているためにそれを検証できない場合、UAC はそれのユーザに証明書のステータス(証明書のサブジェクト、それの署名者、およびすべての鍵指紋情報を含む)を通知して、処理を進める前に明示的な許可を求めなくてはならない[MUST]。証明書がうまく検証できて証明書のサブジェクトがレスポンスの To ヘッダフィールドに一致する場合、あるいはユーザが(通知のあとに)明示的にその証明書の使用を認める場合、UAC はこの証明書を、その証明書の所持者の address-of-record(AOR)でインデックス付けして、ローカルの鍵束に追加するべきである[SHOULD]。UAC がそれ以前のどのトランザクションでも UAS に対してそれ自身の証明書を送信していなかった場合、それは次のリクエストまたはレスポンスに CMS SignedData ボディを使用するべきである[SHOULD]。

以降のいつかの時点で、UA が鍵束内の値に一致する From ヘッダフィールドを含むリクエストまたはレスポンスを受け取るとき、その UA はこれらのメッセージで提示された証明書を鍵束内の既存の証明書と比較するべきである[SHOULD]。食い違いがある場合、UA はそのユーザに証明書の変更を(望ましくは、これが潜在的なセキュリティの侵害であることを示す言葉で)通知して、シグナリングの処理を継続する前にユーザの許可を得なくてはならない[MUST]。ユーザがこの証明書を認める場合、この address-of-record(AOR)に対するそれ以前の値と共にそれが鍵束に追加されるべきである[SHOULD]。

しかしながら、この鍵交換メカニズムは、自己署名証明書または無名の機関によって署名された証明書が使用されるときは、安全な鍵交換を保証しないということに十分注意すること。それはよく知られた攻撃に対して無防備である。著者の意見として、それが提供するセキュリティは一般に何もないよりもましという程度のものであるが、実際には、広く使用されている SSH アプリケーションと同等である。これらの制限は 26.4.2 節でさらに詳しく検討される。

UA が受信者にとって未知の公開鍵で暗号化された S/MIME ボディを受け取る場合、それはそのリクエストを 493(Undecipherable)レスポンスで拒否しなくてはならない[MUST]。このレスポンスは(可能であれば、拒否されるリクエストの To ヘッダフィールドで与えられたすべての address-of-record(AOR)に対応する)レスポンス者にとって有効な証明書を「certs-only」という「smime-type」パラメータを持つ MIME ボディの中に含むべきである[SHOULD]。

いかなる証明書も持たずに送られた 493(Undecipherable)は、レスポンス者が、S/MIME 署名をサポートするにもかかわらず、S/MIME で暗号化されたメッセージを利用できないか利用するつもりがないことを示す。

オプションではない S/MIME ボディを含むリクエストを(「required」という Content-Disposition ヘッダの「handling」パラメータとともに)受け取るユーザエージェントは、MIME タイプを理解できない場合はそ

のリクエストを 415(Unsupported Media Type)レスポンスで拒否しなくてはならない[MUST]ということに注意すること。S/MIME が送られるときにそのようなレスポンスを受け取るユーザエージェントは、それのユーザにリモートデバイスが S/MIME をサポートしないことを通知するべきであり[SHOULD]、引き続き必要に応じてそのリクエストを S/MIME なしで再度送ってもよい[MAY]。しかしながら、この 415 レスポンスはダウングレード攻撃(downgrade attack)の構成要素となるかもしれない。

ユーザエージェントがリクエストで S/MIME ボディを送るが、安全が確保されていない MIME ボディを含むレスポンスを受け取る場合、その UAC はそれのユーザにセッションの安全が確保されないことを通知するべきである[SHOULD]。しかしながら、S/MIME をサポートするユーザエージェントが安全ではないボディを持つリクエストを受け取る場合、それは安全が確保されたボディでレスポンスするべきではない[SHOULD NOT]。しかし、それが送信者から S/MIME を期待する場合は(たとえば、送信者の From ヘッダフィールド値が鍵束のアイデンティティに一致するために)、その UAS はそれのユーザにセッションの安全が確保されないことを通知するべきである[SHOULD]。

前述のテキストで持ち上がる多くの状況は、変則的な証明書管理イベントが起こるときにユーザへの通知を要求する。ユーザはこれらの状況下で何をすべきかたずねるだろう。あくまでも、証明書の予期せぬ変更あるいはセキュリティが期待されるときにセキュリティが確保されていないことが警告の理由であって、必ずしも攻撃が起こっていることを示さない。ユーザはすべてのコネクションの試みを中止するかもしれない、あるいは受け取ったコネクションリクエストを拒否するかもしれない。テレフォニーの用語では、彼らは電話を切り、そしてかけなおす。ユーザは他のパーティにコンタクトする代替手段を見つけて他のパーティの鍵が正当に変更されていることを確認することを望むかもしれない。ユーザは、たとえば、秘密鍵の機密性が危うくなったと疑うときなど、ときには証明書の変更をせざるを得ないということに注意すること。ユーザの秘密鍵がもはや秘密でなくなったとき、ユーザは新しい鍵を正当に生成して、そのユーザの古い鍵を保持するいかなるユーザとも信頼関係を再度確立しなければならない。

最後に、ダイアログの過程で、UA がダイアログ中で以前に交換した証明書と一致しない証明書を CMS SignedData メッセージ中で受け取る場合、UA はそれのユーザに、望ましくはこれが潜在的なセキュリティの侵害であることを示す言葉で、変更を通知しなくてはならない[MUST]。

23.3 MIME ボディの安全確保 (Securing MIME bodies)

SIP に関係する 2 つのタイプの Secure MIME ボディがある。これらのボディの使用は、わずかな変更とともに S/MIME 仕様(参考文献[24])にしたがうべきである。

- 「multipart/signed」は CMS 分離署名とともに使用されなくてはならない[MUST]。

  これは非 S/MIME 準拠の受信者との下位互換を可能にする。

- S/MIME ボディは Content-Disposition ヘッダフィールドを持つべきであり[SHOULD]、「handling」パラメータの値は「required」であるべきである[SHOULD]。

- UAC がその鍵束に、リクエストを送りたいと思う相手の address-of-record(AOR)と関連付けられた証明書を持たない場合、UAC は暗号化された「application/pkcs7-mime」MIME メッセージを送ることができない。UAC は、リモート側の証明書を請願するために、CMS 分離署名を持つ

OPTIONS メッセージのような最初のリクエストを送ってもよい[MAY](署名は 23.4 節で述べられているタイプの「message/sip」ボディ上にあるべきである[SHOULD])。

S/MIME の将来の標準化作業が証明書によらない鍵を定義するかもしれないことに注意すること。

- S/MIME ボディの送信者は、更なるコミュニケーションのために能力とプリファレンスを表現するために「SMIMECapabilities」(参考文献[24]の 2.5.2 節参照)属性を使用するべきである[SHOULD]。送信者は受信者が CMS SignedData メッセージでレスポンスすることを促すために「preferSignedData」能力を使ってもよい[MAY](たとえば、上記のように OPTIONS リクエストを送るとき)ということに特に注意すること。

- S/MIME の実装は、デジタル署名アルゴリズムとして SHA1 を、暗号化アルゴリズムとして 3DES を最低限サポートしなくてはならない[MUST]。他のすべての署名と暗号化のアルゴリズムもサポートしてもよい[MAY]。実装は「SMIMECapabilities」属性で、これらのアルゴリズムに対するサポートをネゴシエートできる。

- SIP メッセージの各 S/MIME ボディはただ一つの証明書で署名されるべきである[SHOULD]。UA が複数の署名を持つメッセージを受け取る場合、最外部の署名がこのボディのための一つの証明書であるとして扱われるべきである。パラレル署名(parallel signature)は使用されるべきではない[SHOULD NOT]。

以下は SIP メッセージ中の暗号化された S/MIME SDP ボディの例である。

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Contact: <sip:alice@pc33.atlanta.com>
Content-Type:application/pkcs7-mime; smime-type=enveloped-data;
     name=smime.p7m
Content-Disposition: attachment; filename=smime.p7m
     handling=required

                  **************************************************
* Content-Type: application/sdp                                   *
*                                                                 *
* v=0                                                             *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com             *
* s=-                                                             *
* t=0 0                                                           *
```

```
* c=IN IP4 pc33.atlanta.com                          *
* m=audio 3456 RTP/AVP 0 1 3 99                      *
* a=rtpmap:0 PCMU/8000                               *
*****************************************************
```

23.4 S/MIME を使用した SIP ヘッダのプライバシーと完全性: SIP トンネリング (SIP Header Privacy and Integrity using S/MIME: Tunneling SIP)

エンド・トゥ・エンド認証、SIP ヘッダフィールドの完全性または機密性をある程度提供する手段として、S/MIME は「message/sip」タイプの MIME ボディ内に SIP メッセージ全体をカプセル化して、通常の SIP ボディと同じ方法でこれらのボディに対して MIME セキュリティを適用することができる。これらのカプセル化された SIP リクエストおよびレスポンスは独自のダイアログやトランザクションを構成しない。それらは、完全性を検証するため、あるいは付加情報を供給するために使用される「外部」メッセージのコピーである。

UAS がトンネルした「message/sip」S/MIME ボディを含むリクエストを受け取る場合、それはトンネルした「message/sip」ボディを同じ smime-type でレスポンスに含めるべきである[SHOULD]。

従来のすべての MIME ボディ(たとえば SDP)は、それらも S/MIME のセキュリティの恩恵を受けられるように「内部」メッセージにアタッチされるべきである[SHOULD]。「message/sip」ボディは、リクエストで安全ではない何らかの MIME タイプも送信されるべき場合には、「multipart/mixed」MIME ボディの一部として送ることができる、ということに注意すること。

23.4.1 SIP ヘッダの完全性と機密性の特性 (Integrity and Confidentiality Properties of SIP Headers)

S/MIME の完全性または機密性のメカニズムが使用されるとき、「内部」メッセージの値と「外部」メッセージの値の間に食い違いが出てくるかもしれない。このドキュメントで記述されているすべてのヘッダフィールドのためにそのような違いをハンドリングするためのルールは、本節で与えられる。

ルースタイムスタンプの目的のために、「message/sip」をトンネルするすべての SIP メッセージは「内部」と「外部」の両方のヘッダに Date ヘッダを含むべきである[SHOULD]。

23.4.1.1 完全性 (Integrity)

完全性チェックが実行されるときはいつでも、ヘッダフィールドの完全性は 20 節で述べられている SIP の比較ルールを使用して、署名されたボディ中のヘッダフィールドの値を「外部」メッセージのそれとマッチングすることによって決定されるべきである。

プロキシサーバによって正当に修正できるヘッダフィールドは、Request-URI、Via、Record-Route、Route、Max-Forwards、および Proxy-Authorization である。これらのヘッダフィールドがエンド・トゥ・エンドで元の状態を保っていない場合、実装はこれをセキュリティの侵害とみなすべきではない[SHOULD NOT]。このドキュメントで定義されているその他のすべてのヘッダフィールドに対する変更は完全性侵害の構成要素になる。ユーザは食い違いを通知されなくてはならない[MUST]。

23.4.1.2 機密性 (Confidentiality)

メッセージが暗号化されるとき、「外部」メッセージに存在しないヘッダフィールドが暗号化されたボディに含まれるかもしれない。

いくつかのヘッダフィールドはリクエストおよびレスポンスで必須であるため、常にプレーンテキストの形態(version)を持たなければならない。これらのヘッダフィールドには、To、From、Call-ID、CSeq、Contactが含まれる。Call-ID、CSeq、または Contact の暗号化された代替を提供することはおそらく有用でないとはいえ、「外部」の To または From に情報として代替を提供することは許可されている。暗号化されたボディ中の値は、トランザクションやダイアログを識別する目的で使用されないことに注意すること。それらは単に通知目的のためにある。暗号化されたボディ中の From ヘッダフィールドが「外部」メッセージの値と異なる場合、暗号化されたボディ内の値がユーザに対して表示されるべきである[SHOULD]が、いかなる以降のメッセージの「外部」ヘッダフィールド中でも使用されてはいけない[MUST NOT]。

ユーザエージェントは主としてエンド・トゥ・エンドの意味合い(semantics)を持つヘッダフィールドの暗号化を望むだろう。それには Subject、Reply-To、Organization、Accept、Accept-Encoding、Accept-Language、Alert-Info、Error-Info、Authentication-Info、Expires、In-Reply-To、Require、Supported、Unsupported、Retry-After、User-Agent、Server、および Warning が含まれる。これらのヘッダフィールドのどれかが暗号化されたボディ中に存在する場合、そのヘッダフィールド値をユーザに対して表示するかあるいは UA の内部状態を設定する必要があったとしても、「外部」のヘッダフィールドの代わりにそれらが使用されるべきである。しかしながらそれらは、いかなる以降のメッセージの「外部」ヘッダでも使用するべきではない[SHOULD NOT]。

Date ヘッダフィールドが存在する場合、それは常に「内部」と「外部」のヘッダで同じでなくてはならない[MUST]。

MIME ボディは「内部」メッセージにアタッチされるので、実装は通常 MIME 固有のヘッダフィールドを暗号化する。それには MIME-Version、Content-Type、Content-Length、Content-Language、Content-Encoding、および Content-Disposition が含まれる。「外部」メッセージは S/MIME ボディのための適切な MIME ヘッダフィールドを持つ。これらのヘッダフィールドは(およびそれらに続くいかなる MIME ボディも)SIP メッセージで受け取った普通の MIME ヘッダフィールドおよびボディとして扱われるべきである。

以下のヘッダフィールドを暗号化することは特に有用ではない。Min-Expires、Timestamp、Authorization、Priority、および WWW-Authenticate このカテゴリはプロキシサーバによって変更されることがあるヘッダフィールドも含む(すぐ前の節で述べられている)。UA は、これらが「外部」メッセージに含まれないのであれば、これらを決して「内部」メッセージに含めるべきでない[SHOULD]。暗号化されたボディでこれらのどれかのヘッダフィールドを受け取る UA は、暗号化されている値を無視するべきである[SHOULD]。

SIP の拡張は追加のヘッダフィールドを定義するかもしれないということに注意すること。それらの拡張の著者は、そのようなヘッダフィールドの完全性と機密性の特性について述べるべきである。SIP UA が完全性を侵害する未知のヘッダフィールドに遭遇する場合、それはそのヘッダフィールドを無視しなくてはならない[MUST]。

23.4.2 **トンネリングの完全性と認証** (Tunneling Integrity and Authentication)

S/MIME ボディ内の SIP メッセージをトンネリングすることは、送信者が安全確保を望むヘッダフィールドが CMS 分離署名で署名されて「message/sip」MIME ボディ中に複製(replicate)されている場合、その SIP ヘッダフィールドに対して完全性を提供する。

「message/sip」ボディが少なくとも基本的なダイアログ識別子(To、From、Call-ID、CSeq)を含むなら、署名された MIME ボディは限定された認証を提供できる。ボディに署名するために使用された証明書が受信者に知られていないために検証できない場合、最低限、ダイアログ中で後から来るリクエストがそのダイアログを開始したのと同じ証明書の所有者によって送信されたことを確認するためにその署名が使用できる。署名された MIME ボディの受信者が証明書を信用するにたる何らかの強い誘因を持つ場合(それを検証することができた、信頼できるリポジトリからそれを取得した、あるいはそれをよく使用する)、証明書のサブジェクトのアイデンティティをより強く断定するものとして、署名を利用することができる。

ヘッダフィールド全体の追加や削減に関して起こりうる混乱を取り除くために、送信者はリクエストからすべてのヘッダフィールドを署名されたボディに複写するべきである[SHOULD]。完全性の保護を要求するいかなるメッセージボディも、「内部」メッセージにアタッチされなくてはならない[MUST]。

署名されたボディを持つメッセージ中に Date ヘッダが存在する場合、受信者は必要に応じてそのヘッダフィールドの値を自身の内部時計と比較するべきである[SHOULD]。大幅な時間の不一致(1 時間単位あるいはそれ以上)が検出される場合、ユーザエージェントはユーザに異常を警告し、これが潜在的なセキュリティの侵害であることを注意するべきである[SHOULD]。

受信者によってメッセージに完全性の侵害が検知される場合、それがリクエストであればそのメッセージを 403(Forbidden)レスポンスで拒否してもよい[MAY]。あるいはいかなる既存のダイアログも終了してもよい[MAY]。UA はユーザにこの状況を通知し、どのように続行するかについての明確な支持を請うべきである[SHOULD]。

以下は、トンネルされた「message/sip」ボディの使用例である。

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0
```

```
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
  handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-
```

23.4.3 トンネリング暗号化 (Tunneling Encryption)

このメカニズムを CMS EnvelopedData メッセージ S/MIME ボディ内の「message/sip」MIME ボディを暗号化するために使用することも望ましいかもしれないが、実際にはほとんどのヘッダフィールドが少なくともいくらかはネットワークに使用されている。S/MIME での暗号化の一般的な用途はメッセージヘッダというよりもむしろ SDP のようなメッセージボディの安全を確保することである。Subject や Organization のようないくつかの通知ヘッダフィールドは、おそらくはエンド・トゥ・エンドの安全性を保証する。将来の SIP アプリケーションで定義されるヘッダも不明瞭さを要求するかもしれない。

ヘッダフィールド暗号化の、他の見込みある応用例は選択的な匿名(selective anonymity)である。リクエストは、個人情報を含まない From ヘッダフィールド(たとえば、sip:anonymous@anonymizer.invalid)を用いて構築されるかもしれない。しかしながら、発信元の本当の address-of-record(AOR)を含む別の From

ヘッダフィールドは、ダイアログのエンドポイントに対してのみ可視である「message/sip」MIME ボディ内に暗号化されるかもしれない。

このメカニズムが匿名のために使用される場合、From ヘッダフィールドは、送信者に関連付けられた適切な S/MIME 鍵を検索するための証明書の鍵束へのインデックスとして、もはやメッセージの受信者が利用することはできない、ということに注意すること。メッセージはまず復号され、そして「内部の」From ヘッダフィールドがインデックスとして使用されなくてはならない[MUST]。

エンド・トゥ・エンドの完全性を提供するために、暗号化された「message/sip」MIME ボディは送信者によって署名されるべきである[SHOULD]。これは、双方共にタイプが「application/pkcs7-mime」である、暗号化されたボディと署名を含む「multipart/signed」MIME ボディを生成する。

以下の暗号化されて署名されたメッセージの例では、アスタリスク(*)で囲まれたテキストは暗号化されている。

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Anonymous <sip:anonymous@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:pc33.atlanta.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type:application/pkcs7-mime;
  smime-type=enveloped-data;name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
  handling=required
Content-Length: 231

**********************************************************
* Content-Type: message/sip                              *
*                                                        *
* INVITE sip:bob@biloxi.com SIP/2.0                      *
* Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 *
* To: Bob <bob@biloxi.com>                               *
```

```
* From: Alice <alice@atlanta.com>;tag=1928301774          *
* Call-ID: a84b4c76e66710                                 *
* CSeq: 314159 INVITE                                     *
* Max-Forwards: 70                                        *
* Date: Thu, 21 Feb 2002 13:02:03 GMT                     *
* Contact: <sip:alice@pc33.atlanta.com>                   *
*                                                         *
* Content-Type: application/sdp                           *
*                                                         *
* v=0                                                     *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com     *
* s=Session SDP                                           *
* t=0 0                                                   *
* c=IN IP4 pc33.atlanta.com                               *
* m=audio 3456 RTP/AVP 0 1 3 99                           *
* a=rtpmap:0 PCMU/8000                                    *
***********************************************************

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
  handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-
```

２４．例 (Examples)

以下の例では、簡潔さのためにメッセージボディとそれに対応する Content-Length および Content-Type ヘッダフィールドを省略することが多い。

24.1 登録 (Registration)

開始時に Bob が登録を行う。メッセージフローは図 9 に示されている。登録に通常要求される認証は簡略化のために示されていないということに注意すること。

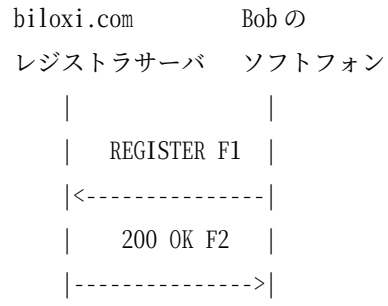

図 9: SIP の登録の例

F1 REGISTER Bob -> Registrar

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

登録は 2 時間後に期限が切れる。レジストラサーバは 200 OK でレスポンスする。

F2 200 OK Registrar -> Bob

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

24.2 **セッションセットアップ** (Session Setup)

この例は 4 節のセッションセットアップ例の全詳細を含む。メッセージフローは図 1 に示されている。これらのフローは最低限要求されるヘッダフィールドを示すということに注意すること。Allow や Supported のような他のいくつかのヘッダフィールドが通常は存在する。

F1 INVITE Alice -> atlanta.com proxy

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

(Alice の SDP は示されていない)

F2 100 Trying atlanta.com proxy -> Alice

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

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

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

(Alice の SDP は示されていない)

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

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

F5 INVITE biloxi.com proxy -> Bob

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

(Alice の SDP は示されていない)

F6 180 Ringing Bob -> biloxi.com proxy

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

```
        From: Alice <sip:alice@atlanta.com>;tag=1928301774
        Call-ID: a84b4c76e66710
        Contact: <sip:bob@192.0.2.4>
        CSeq: 314159 INVITE
        Content-Length: 0

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

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

F8 180 Ringing atlanta.com proxy -> Alice

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

F9 200 OK Bob -> biloxi.com proxy

        SIP/2.0 200 OK
        Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
          ;received=192.0.2.3
        Via:SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
          ;received=192.0.2.2
        Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
          ;received=192.0.2.1
        To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
        From: Alice <sip:alice@atlanta.com>;tag=1928301774
```

```
        Call-ID: a84b4c76e66710
        CSeq: 314159 INVITE
        Contact: <sip:bob@192.0.2.4>
        Content-Type: application/sdp
        Content-Length: 131

     (Bob の SDP は示されていない)

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

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

     (Bob の SDP は示されていない)

F11 200 OK atlanta.com proxy -> Alice

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

     (Bob の SDP は示されていない)

F12 ACK Alice -> Bob

        ACK sip:bob@192.0.2.4 SIP/2.0
```

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

Alice と Bob の間のメディアセッションは今確立された。

Bob が最初に電話を切る。Bob の SIP フォンはそれ自身の CSeq 番号空間(この例では、231 で始まる)を保持することに注意すること。Bob がリクエストを行っているので、To と From の URI および tag が交換された。

```
F13 BYE Bob -> Alice

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

F14 200 OK Alice -> Bob

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

SIP コールフローのドキュメント(参考文献[40])が SIP メッセージの更なる例を含んでいる。

２５．SIP プロトコルのための拡張 BNF (Augmented BNF for the SIP Protocol)

このドキュメントで規定されているすべてのメカニズムは、文と RFC2234(参考文献[10])で定義されている拡張 BNF 形式の両方で記述されている。RFC 2234 の 6.1 節で、この仕様で使用する中心的なルールセットを定義しているので、ここでは繰り返さない。実装者は、この仕様を理解するためにその表記法と RFC2234 の内容を熟知している必要がある。SP、LWS、HTAB、CRLF、DIGIT、ALPHA などの特定の基本ルールは大文字で表記される。ルール名を明確にするために定義中でカギ括弧が使用される。

構文的には角括弧の使用は冗長である。それは、特定のパラメータの使用がオプションであることの意味論的なヒントとして使用される。

25.1 基本ルール (Basic Rules)

以下のルールは、パースする基本構成物を記述するために、この仕様全体を通して使用される。US-ASCII でコード化されたキャラクターセットは ANSI X3.4-1986 で定義されている。

```
alphanum  =  ALPHA / DIGIT
```

いくつかのルールが RFC2396(参考文献[5])から組み入れられるが、それらが RFC2234(参考文献[10])に準拠するように更新される。これには以下のものが含まれる。

```
reserved          =    ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
                        / "$" / ","
unreserved        =  alphanum / mark
mark              =  "-" / "_" / "." / "!" / "~" / "*" / "'"
                        / "(" / ")"
escaped           =  "%" HEXDIG HEXDIG
```

SIP ヘッダフィールド値は、継続行がスペースまたは水平タブで始まるなら複数行に折りたためる。折りたたみを含むすべての linear white space は、SP と同じ意味合い(semantics)を持つ。受信者は、フィールド値を解釈する前あるいはメッセージをダウンストリームにフォワードする前に、いかなる linear white space でも一つの SP に置き換えてもよい[MAY]。これは RFC2616(参考文献[8])で述べられているように、HTTP/1.1 と正確に同じ動作をすることを意図されている。

```
LWS  =  [*WSP CRLF] 1*WSP ; linear whitespace
SWS  =  [LWS] ; sep whitespace
```

ヘッダ名を値の残りの部分から分離するために、コロンが使用される。コロンは上記のルールにより、その前にホワイトスペース(改行は認められない)、後にホワイトスペース(改行も含む)が認められている。HCOLON はこの構成概念を定義する。

```
HCOLON  =  *( SP / HTAB ) ":" SWS
```

TEXT-UTF8 ルールは、メッセージパーサーによって解釈されることを意図されていない記述的フィールドと値でのみ使用される。*TEXT-UTF8 という語は、UTF-8 キャラクターセット(RFC2279 参考文献[7])の文字を含む。TEXT-UTF8-TRIM ルールは引用符で囲まれた文字列ではない(先行する LWS と後に続く LWS が意味を持たない)、記述的フィールドのコンテンツに使用される。これに関しては、SIP は ISO8859-1 キャラクターセットを使用する HTTP と異なる。

```
TEXT-UTF8-TRIM        =   1*TEXT-UTF8char *(*LWS TEXT-UTF8char)
TEXT-UTF8char         =   %x21-7E / UTF8-NONASCII
```

```
UTF8-NONASCII  =       %xC0-DF 1UTF8-CONT
                        /  %xE0-EF 2UTF8-CONT
                        /  %xF0-F7 3UTF8-CONT
                        /  %xF8-Fb 4UTF8-CONT
                        /  %xFC-FD 5UTF8-CONT
UTF8-CONT        =      %x80-BF
```

CRLF はヘッダフィールド継続の一部としてのみ、TEXT-UTF8-TRIM の定義で認められている。TEXT-UTF8-TRIM の値を解釈する前に、折りたたみ LWS が一つの SP で置き換えられることが期待されている。

いくつかのプロトコルエレメントにおいて、16 進数文字が使用される。いくつかのエレメント(認証)は 16 進数のアルファベット文字が小文字であることを強いる。

```
LHEX  =  DIGIT / %x61-66 ;小文字の a-f
```

多くの SIP ヘッダフィールド値は、LWS または特別な文字で区切られた複数の語から成る。特に指定されなければ、トークンは大文字小文字を区別しない。これらの特別な文字は、パラメータ値で使用されるために、引用符で囲まれた文字列中になくてはならない[MUST]。この語の構成は、ほとんどの分離記号の使用を可能にするために Call-ID で使われる。

```
token       =  1*(alphanum / "-" / "." / "!" / "%" / "*"
                 / "_" / "+" / "`" / "'" / "~" )
separators  =  "(" / ")" / "<" / ">" / "@" /
                "," / ";" / ":" / "\" / DQUOTE /
                "/" / "[" / "]" / "?" / "=" /
                "{" / "}" / SP / HTAB
word        =  1*(alphanum / "-" / "." / "!" / "%" / "*" /
               "_" / "+" / "`" / "'" / "~" /
               "(" / ")" / "<" / ">" /
               ":" / "\" / DQUOTE /
               "/" / "[" / "]" / "?" /
               "{" / "}" )
```

エレメント間でトークンまたは分離記号が使用されるとき、これらの文字の前後にホワイトスペースが認められることが多い。

```
STAR            =  SWS "*" SWS ; アスタリスク
SLASH           =  SWS "/" SWS ; スラッシュ
EQUAL           =  SWS "=" SWS ; 等号
LPAREN          =  SWS "(" SWS ; 左括弧
RPAREN          =  SWS ")" SWS ; 右括弧
RAQUOT          =  ">" SWS ; 右角括弧
LAQUOT          =  SWS "<"; 左角括弧
```

```
COMMA           =  SWS "," SWS ; カンマ
SEMI            =  SWS ";" SWS ; セミコロン
COLON           =  SWS ":" SWS ; コロン
LDQUOT          =  SWS DQUOTE; 開き二重引用符
RDQUOT          =  DQUOTE SWS ; 閉じ二重引用符
```

コメント文字を括弧で括ることによって、いくつかのSIPヘッダフィールドにコメントを含めることができる。コメントは、フィールド値定義の一部に「comment」を含むフィールドにのみ許可されている。他のすべてのフィールドでは、括弧はフィールド値の一部とみなされる。

```
comment         =  LPAREN *(ctext / quoted-pair / comment) RPAREN
ctext           =  %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII / LWS
```

ctextは、左括弧、右括弧、バックスラッシュを除くすべての文字を含む。文字列は、二重引用符で囲まれている場合、一語としてパースされる。引用符で囲まれた文字列中では、引用符(")とバックスラッシュ(\)はエスケープされる必要がある。

```
quoted-string   =  SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
qdtext          =  LWS / %x21 / %x23-5B / %x5D-7E / UTF8-NONASCII
```

バックスラッシュ文字(\ )は、引用符で囲まれた文字列およびコメント構成物の中でのみ、一文字引用メカニズムとして使用してもよい[MAY]。HTTP/1.1とは異なり、CR文字とLF文字は行の折りたたみとヘッダ分割の衝突を避けるためにこのメカニズムでエスケープできない。

```
quoted-pair     =  "\" (%x00-09 / %x0B-0C / %x0E-7F)

SIP-URI         =  "sip:" [ userinfo ] hostport
                   uri-parameters [ headers ]
SIPS-URI        =  "sips:" [ userinfo ] hostport
                   uri-parameters [ headers ]
userinfo        =  ( user / telephone-subscriber ) [ ":" password ] "@"
user            =  1*( unreserved / escaped / user-unreserved )
user-unreserved =  "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password        =  *( unreserved / escaped /
                   "&" / "=" / "+" / "$" / "," )
hostport        =  host [ ":" port ]
host            =  hostname / IPv4address / IPv6reference
hostname        =  *( domainlabel "." ) toplabel [ "." ]
domainlabel     =  alphanum
                   / alphanum *( alphanum / "-" ) alphanum
toplabel        =  ALPHA / ALPHA *( alphanum / "-" ) alphanum

IPv4address     =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
```

```
IPv6reference     =  "[" IPv6address "]"
IPv6address       =  hexpart [ ":" IPv4address ]
hexpart           =  hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq            =  hex4 *( ":" hex4)
hex4              =  1*4HEXDIG
port              =  1*DIGIT
```

電話加入者のための BNF は RFC2806(参考文献[9])でみつけることができる。しかしながら、SIP URI のユーザ部分で許可されていないのにそこで認められているいかなる文字もエスケープされなくてはならない[MUST]ことに注意すること。

```
uri-parameters    =  *( ";" uri-parameter)
uri-parameter     =  transport-param / user-param / method-param
/ ttl-param / maddr-param / lr-param / other-param
transport-param=  "transport="
                     ( "udp" / "tcp" / "sctp" / "tls" / other-transport)
other-transport   =  token
user-param        =  "user=" ( "phone" / "ip" / other-user)
other-user        =  token
method-param      =  "method=" Method
ttl-param         =  "ttl=" ttl
maddr-param       =  "maddr=" host
lr-param          =  "lr"
other-param       =  pname [ "=" pvalue ]
pname             =  1*paramchar
pvalue            =  1*paramchar
paramchar         =  param-unreserved / unreserved / escaped
param-unreserved      =  "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers           =  "?" header *( "&" header )
header            =  hname "=" hvalue
hname             =  1*( hnv-unreserved / unreserved / escaped )
hvalue            =  *( hnv-unreserved / unreserved / escaped )
hnv-unreserved    =  "[" / "]" / "/" / "?" / ":" / "+" / "$"
SIP-message       =  Request / Response
Request           =  Request-Line
                     *( message-header )
                     CRLF
                     [ message-body ]
Request-Line      =  Method SP Request-URI SP SIP-Version CRLF
Request-URI       =  SIP-URI / SIPS-URI / absoluteURI
absoluteURI       =  scheme ":" ( hier-part / opaque-part )
hier-part         =  ( net-path / abs-path ) [ "?" query ]
```

```
net-path          =  "//" authority [ abs-path ]
abs-path          =  "/" path-segments

opaque-part       =  uric-no-slash *uric
uric              =  reserved / unreserved / escaped
uric-no-slash     =  unreserved / escaped / ";" / "?" / ":" / "@"
                      / "&" / "=" / "+" / "$" / ","
path-segments     =  segment *( "/" segment )
segment           =  *pchar *( ";" param )
param             =  *pchar
pchar             =  unreserved / escaped /
                      ":" / "@" / "&" / "=" / "+" / "$" / ","
scheme            =  ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )
authority         =  srvr / reg-name
srvr              =  [ [ userinfo "@" ] hostport ]
reg-name          =  1*( unreserved / escaped / "$" / ","
                      / ";" / ":" / "@" / "&" / "=" / "+" )
query             =  *uric
SIP-Version       =  "SIP" "/" 1*DIGIT "." 1*DIGIT

message-header    =  (Accept
                  /  Accept-Encoding
                  /  Accept-Language
                  /  Alert-Info
                  /  Allow
                  /  Authentication-Info
                  /  Authorization
                  /  Call-ID
                  /  Call-Info
                  /  Contact
                  /  Content-Disposition
                  /  Content-Encoding
                  /  Content-Language
                  /  Content-Length
                  /  Content-Type
                  /  CSeq
                  /  Date
                  /  Error-Info
                  /  Expires
                  /  From
                  /  In-Reply-To
                  /  Max-Forwards
                  /  MIME-Version
```

```
                / Min-Expires
                / Organization
                / Priority
                / Proxy-Authenticate
                / Proxy-Authorization
                / Proxy-Require
                / Record-Route
                / Reply-To
                / Require
                / Retry-After
                / Route
                / Server
                / Subject
                / Supported
                / Timestamp
                / To
                / Unsupported
                / User-Agent
                / Via
                / Warning
                / WWW-Authenticate
                / extension-header) CRLF

INVITEm         =  %x49.4E.56.49.54.45 ; 大文字の INVITE
ACKm            =  %x41.43.4B ; 大文字の ACK
OPTIONSm        =  %x4F.50.54.49.4F.4E.53 ; 大文字の OPTIONS
BYEm            =  %x42.59.45 ; 大文字の BYE
CANCELm         =  %x43.41.4E.43.45.4C ; 大文字の CANCEL
REGISTERm       =  %x52.45.47.49.53.54.45.52 ; 大文字の REGISTER
Method          =  INVITEm / ACKm / OPTIONSm / BYEm
                   / CANCELm / REGISTERm
                   / extension-method
                   extension-method      =  token
Response        =  Status-Line
                   *( message-header )
                   CRLF
                   [ message-body ]

Status-Line     =  SIP-Version SP Status-Code SP Reason-Phrase CRLF
Status-Code     =  Informational
                /   Redirection
                /   Success
                /   Client-Error
```

```
                / Server-Error
                / Global-Failure
                / extension-code
extension-code  = 3DIGIT
Reason-Phrase   = *(reserved / unreserved / escaped
                  / UTF8-NONASCII / UTF8-CONT / SP / HTAB)

Informational   = "100" ; Trying (試行中)
                / "180" ; Ringing (呼び出し中)
                / "181" ; Call Is Being Forwarded
                      (呼がフォワードされている)
                / "182" ; Queued (キューイングされた)
                / "183" ; Session Progress (セッションの進捗状況)

Success         = "200" ; OK

Redirection     = "300" ; Multiple Choices (複数の選択肢がある)
                / "301" ; Moved Permanently (恒久的に移動した)
                / "302" ; Moved Temporarily (一時的に移動した)
                / "305" ; Use Proxy (プロキシを使用せよ)
                / "380" ; Alternative Service (代替サービス)

Client-Error    = "400" ; Bad Request (不正なリクエスト)
                / "401" ; Unauthorized (認証されていない)
                / "402" ; Payment Required (料金支払いが必要)
                / "403" ; Forbidden (禁止)
                / "404" ; Not Found (見つからない)
                / "405" ; Method Not Allowed
                          (メソッドが許可されていない)
                / "406" ; Not Acceptable (受け入れできない)
                / "407" ; Proxy Authentication Required
                (プロキシ認証が必要)
                / "408" ; Request Timeout (リクエストがタイムアウトした)
                / "410" ; Gone (リソースがもう存在しない)
                / "413" ; Request Entity Too Large
                          (リクエストのエンティティが大きすぎる)
                / "414" ; Request-URI Too Large
                          (Request-URI が長すぎる)
                / "415" ; Unsupported Media Type
                          (サポートされていないメディアタイプ)
                / "416" ; Unsupported URI Scheme
                          (サポートされていない URI スキーム)
                / "420" ; Bad Extension (不正な拡張)
```

```
                / "421" ; Extension Required (拡張が必要)
                / "423" ; Interval Too Brief (間隔が短すぎる)
                / "480" ; Temporarily not available
                          (一時的に利用不可)
                / "481" ; Call Leg/Transaction Does Not Exist
                          (コールレグまたはトランザクションが存在しない)
                / "482" ; Loop Detected (ループが検知された)
                / "483" ; Too Many Hops (ホップが多すぎる)
                / "484" ; Address Incomplete (アドレスが不完全)
                / "485" ; Ambiguous (不明瞭)
                / "486" ; Busy Here (ここはビジー)
                / "487" ; Request Terminated(リクエストが終了させられた)
                / "488" ; Not Acceptable Here (ここでは受け入れ不能)
                / "491" ; Request Pending (リクエストペンディング)
                / "493" ; Undecipherable (解読不能)

Server-Error    = "500" ; Internal Server Error (内部サーバエラー)
                / "501" ; Not Implemented (実装されていない)
                / "502" ; Bad Gateway (不正なゲートウェイ)
                / "503" ; Service Unavailable (サービスを利用できない)
                / "504" ; Server Time-out (サーバタイムアウト)
                / "505" ; SIP Version not supported
                          (サポートされていない SIP バージョン)
                / "513" ; Message Too Large (メッセージが大きすぎる)

Global-Failure  = "600" ; Busy Everywhere (どの場所もビジー)
                / "603" ; Decline (辞退)
                / "604" ; Does not exist anywhere (どこにも存在しない)
                / "606" ; Not Acceptable (受け入れ不能)

Accept          = "Accept" HCOLON
                  [ accept-range *(COMMA accept-range) ]
accept-range    = media-range *(SEMI accept-param)
media-range     = ( "*/*"
                  / ( m-type SLASH "*" )
                  / ( m-type SLASH m-subtype )
                  ) *( SEMI m-parameter )
accept-param    = ("q" EQUAL qvalue) / generic-param
qvalue          = ( "0" [ "." 0*3DIGIT ] )
                  / ( "1" [ "." 0*3("0") ] )
generic-param   = token [ EQUAL gen-value ]
gen-value       = token / host / quoted-string
```

```
Accept-Encoding=  "Accept-Encoding" HCOLON
                      [ encoding *(COMMA encoding) ]
encoding          =  codings *(SEMI accept-param)
codings           =  content-coding / "*"
content-coding    =  token

Accept-Language     =  "Accept-Language" HCOLON
                       [ language *(COMMA language) ]
language          =  language-range *(SEMI accept-param)
language-range    =  ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) / "*" )

Alert-Info      = "Alert-Info" HCOLON alert-param *(COMMA alert-param)
alert-param     =  LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Allow             =  "Allow" HCOLON [Method *(COMMA Method)]

Authorization     =  "Authorization" HCOLON credentials
credentials       =  ("Digest" LWS digest-response)
                      / other-response
digest-response   =  dig-resp *(COMMA dig-resp)
dig-resp          =  username / realm / nonce / digest-uri
                      / dresponse / algorithm / cnonce
                      / opaque / message-qop
                      / nonce-count / auth-param
username          =  "username" EQUAL username-value
username-value    =  quoted-string
digest-uri        =  "uri" EQUAL LDQUOT digest-uri-value RDQUOT
digest-uri-value  =  rquest-uri ; HTTP/1.1で規定されている request-uri と同じ
message-qop       =  "qop" EQUAL qop-value

cnonce            =  "cnonce" EQUAL cnonce-value
cnonce-value      =  nonce-value
nonce-count       =  "nc" EQUAL nc-value
nc-value          =  8LHEX
dresponse         =  "response" EQUAL request-digest
request-digest    =  LDQUOT 32LHEX RDQUOT
auth-param        =  auth-param-name EQUAL
                       ( token / quoted-string )
auth-param-name      =  token
other-response    =  auth-scheme LWS auth-param
*(COMMA auth-param)
auth-scheme       =  token
```

```
Authentication-Info   =  "Authentication-Info" HCOLON ainfo
                          *(COMMA ainfo)
ainfo             =  nextnonce / message-qop
                      / response-auth / cnonce
                      / nonce-count
nextnonce         =  "nextnonce" EQUAL nonce-value
response-auth     =  "rspauth" EQUAL response-digest
response-digest   =  LDQUOT *LHEX RDQUOT

Call-ID           =  ( "Call-ID" / "i" ) HCOLON callid
called            =  word [ "@" word ]

Call-Info         =  "Call-Info" HCOLON info *(COMMA info)
info              =  LAQUOT absoluteURI RAQUOT *( SEMI info-param)
info-param        =  ( "purpose" EQUAL ( "icon" / "info"
                      / "card" / token ) ) / generic-param

Contact           =  ("Contact" / "m" ) HCOLON
                      ( STAR / (contact-param *(COMMA contact-param)))
contact-param     =  (name-addr / addr-spec) *(SEMI contact-params)
name-addr         =  [ display-name ] LAQUOT addr-spec RAQUOT
addr-spec         =  SIP-URI / SIPS-URI / absoluteURI
display-name      =  *(token LWS)/ quoted-string

contact-params    =  c-p-q / c-p-expires / contact-extension
c-p-q             =  "q" EQUAL qvalue
c-p-expires       =  "expires" EQUAL delta-seconds
contact-extension  =  generic-param
delta-seconds     =  1*DIGIT

Content-Disposition   =  "Content-Disposition" HCOLON
                         disp-type *( SEMI disp-param )
disp-type         = "render" / "session" / "icon" / "alert" / disp-extension-token

disp-param        =  handling-param / generic-param
handling-param    =  "handling" EQUAL
                      ( "optional" / "required"/ other-handling )
other-handling    =  token
disp-extension-token=  token

Content-Encoding    =  ( "Content-Encoding" / "e" ) HCOLON
                      content-coding *(COMMA content-coding)
```

```
Content-Language=  "Content-Language" HCOLON
                        language-tag *(COMMA language-tag)
language-tag       =  primary-tag *( "-" subtag )
primary-tag        =  1*8ALPHA
subtag             =  1*8ALPHA

Content-Length     =  ( "Content-Length" / "l" ) HCOLON 1*DIGIT

Content-Type       =  ( "Content-Type" / "c" ) HCOLON media-type
media-type         =  m-type SLASH m-subtype *(SEMI m-parameter)
m-type             =  discrete-type / composite-type
discrete-type      =  "text" / "image" / "audio" / "video"
                       / "application" / extension-token
composite-type     =  "message" / "multipart" / extension-token
extension-token    =  ietf-token / x-token
ietf-token         =  token
x-token            =  "x-" token
m-subtype          =  extension-token / iana-token
iana-token         =  token
m-parameter        =  m-attribute EQUAL m-value
m-attribute        =  token
m-value            =  token / quoted-string

Cseq               =  "CSeq" HCOLON 1*DIGIT LWS Method

Date               =  "Date" HCOLON SIP-date
SIP-date           =  rfc1123-date
rfc1123-date       =  wkday "," SP date1 SP time SP "GMT"
date1              =  2DIGIT SP month SP 4DIGIT
                       ; 日 月 年 (例: 02 Jun 1982)
time               =  2DIGIT ":" 2DIGIT ":" 2DIGIT
                       ; 00:00:00 - 23:59:59
wkday              =  "Mon" / "Tue" / "Wed" / "Thu" / "Fri" / "Sat" / "Sun"
month              =  "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" / "Jul" /
                           "Aug"/ "Sep" / "Oct" / "Nov" / "Dec"

Error-Info         =  "Error-Info" HCOLON error-uri *(COMMA error-uri)

error-uri       =  LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Expires            =  "Expires" HCOLON delta-seconds
From               =  ( "From" / "f" ) HCOLON from-spec
from-spec          =  ( name-addr / addr-spec )
```

```
                  *( SEMI from-param )
from-param        =  tag-param / generic-param
tag-param         =  "tag" EQUAL token

In-Reply-To       =  "In-Reply-To" HCOLON callid *(COMMA callid)

Max-Forwards      =  "Max-Forwards" HCOLON 1*DIGIT

MIME-Version      =  "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

Min-Expires       =  "Min-Expires" HCOLON delta-seconds

Organization      =  "Organization" HCOLON [TEXT-UTF8-TRIM]

Priority          =  "Priority" HCOLON priority-value
priority-value    =  "emergency" / "urgent" / "normal"
                      / "non-urgent" / other-priority
other-priority    =  token

Proxy-Authenticate    =  "Proxy-Authenticate" HCOLON challenge
challenge         =  ("Digest" LWS digest-cln *(COMMA digest-cln))
                      / other-challenge
other-challenge   =  auth-scheme LWS auth-param
                      *(COMMA auth-param)
digest-cln        =  realm / domain / nonce / opaque / stale / algorithm
                      / qop-options / auth-param
realm             =  "realm" EQUAL realm-value
realm-value       =  quoted-string
domain            =  "domain" EQUAL LDQUOT URI
                      *( 1*SP URI ) RDQUOT
URI               =  absoluteURI / abs-path
nonce             =  "nonce" EQUAL nonce-value
nonce-value       =  quoted-string
opaque            =  "opaque" EQUAL quoted-string
stale             =  "stale" EQUAL ( "true" / "false" )
algorithm         =  "algorithm" EQUAL ( "MD5" / "MD5-sess" / token )
qop-options       =  "qop" EQUAL LDQUOT qop-value
                      *("," qop-value) RDQUOT
qop-value         =  "auth" / "auth-int" / token

Proxy-Authorization=  "Proxy-Authorization" HCOLON credentials

Proxy-Require     =  "Proxy-Require" HCOLON option-tag
```

```
                    *(COMMA option-tag)
option-tag        =  token

Record-Route      = "Record-Route" HCOLON rec-route *(COMMA rec-route)
rec-route         =  name-addr *( SEMI rr-param )
rr-param          =  generic-param

Reply-To          =  "Reply-To" HCOLON rplyto-spec
rplyto-spec       =  ( name-addr / addr-spec ) *( SEMI rplyto-param )
rplyto-param      =  generic-param
Require           =  "Require" HCOLON option-tag *(COMMA option-tag)

Retry-After       =  "Retry-After" HCOLON delta-seconds
                     [ comment ] *( SEMI retry-param )
retry-param       =  ("duration" EQUAL delta-seconds)
                   / generic-param

Route             =  "Route" HCOLON route-param *(COMMA route-param)
route-param       =  name-addr *( SEMI rr-param )

Server            =  "Server" HCOLON server-val *(LWS server-val)
server-val        =  product / comment
product           =  token [SLASH product-version]
product-version =  token

Subject           =  ( "Subject" / "s" ) HCOLON [TEXT-UTF8-TRIM]

Supported         =  ( "Supported" / "k" ) HCOLON
                   [option-tag *(COMMA option-tag)]

Timestamp         =  "Timestamp" HCOLON 1*(DIGIT)
                   [ "." *(DIGIT) ] [ LWS delay ]
delay             =  *(DIGIT) [ "." *(DIGIT) ]

To                =  ( "To" / "t" ) HCOLON ( name-addr
                     / addr-spec ) *( SEMI to-param )
to-param          =  tag-param / generic-param

Unsupported       = "Unsupported" HCOLON option-tag *(COMMA option-tag)

User-Agent        =  "User-Agent" HCOLON server-val *(LWS server-val)

Via               =  ( "Via" / "v" ) HCOLON via-parm *(COMMA via-parm)
```

```
via-parm          =  sent-protocol LWS sent-by *( SEMI via-params )
via-params        =  via-ttl / via-maddr / via-received / via-branch
                      / via-extension
via-ttl           =  "ttl" EQUAL ttl
via-maddr         =  "maddr" EQUAL host
via-received      =  "received" EQUAL (IPv4address / IPv6address)
via-branch        =  "branch" EQUAL token
via-extension     =  generic-param
sent-protocol     =  protocol-name SLASH protocol-version
                      SLASH transport
protocol-name     =  "SIP" / token
protocol-version  =  token
transport         =  "UDP" / "TCP" / "TLS" / "SCTP" / other-transport
sent-by           =  host [ COLON port ]
ttl               =  1*3DIGIT ; 0 から 255

Warning           = "Warning" HCOLON warning-value *(COMMA warning-value)
warning-value     =  warn-code SP warn-agent SP warn-text
warn-code         =  3DIGIT
warn-agent        =  hostport / pseudonym
                     ;  デバッグで使用するために Warning ヘッダ
                     ;  に追加されるサーバの名前または匿名
warn-text         =  quoted-string
pseudonym         =  token

WWW-Authenticate      =  "WWW-Authenticate" HCOLON challenge

extension-header=  header-name HCOLON header-value
header-name       =  token
header-value      =  *(TEXT-UTF8char / UTF8-CONT / LWS)

message-body      =  *OCTET
```

２６．セキュリティの考慮: 危機(Threat)モデルとセキュリティ利用の推奨 (Security Considerations: Threat Model and Security Usage Recommendations)

SIP は安全を確保するのが容易ではないプロトコルである。中継媒体を使用すること、多面的な信頼関係、信頼がまったくないエレメント間での予想される利用、およびユーザ・トゥ・ユーザでのオペレーションが、セキュリティを些細な問題とは程遠いものにしている。大規模な調整がなく、多様な環境と用途で現在実施可能なセキュリティのソリューションが必要とされる。これらの種々の要求にあうように、SIP の異なる側面と用途に適用可能ないくつかの別個のメカニズムが要求されるだろう。

SIP シグナリングそれ自体のセキュリティは、RTP のように SIP と協調して使用されるプロトコルのセキュリティ、または SIP が伝えるかもしれないすべての固有のボディのセキュリティの意味あいには何ら影響

しない(SIP の安全確保において MIME セキュリティが重要な役割を演じるけれども)、ということに注意すること。セッションに関連付けられたいかなるメディアも、関連付けられるすべての SIP シグナリングに依存せずにエンド・トゥ・エンドで暗号化できる。メディアの暗号化はこのドキュメントの適用範囲外である。

この後に続く検討事項ではまず始めに、SIP のセキュリティの必要性を広く認識する、よく知られた危機(threat)モデル一式を吟味する。これらの危機(threat)に対処するために要求されるセキュリティサービスのセットがそれに続いて詳述され、その後にこれらのサービスを提供するために使用できるいくつかのセキュリティメカニズムの説明が続く。次に、SIP のセキュリティを向上するためにこれらのセキュリティメカニズムが使用される典型的な配備とともに、SIP の実装者に対する要求が列挙される。プライバシーに関するいくつかの注意事項で本節を終える。

26.1 攻撃と危機(Threat)モデル (Attacks and Threat Models)

本節では、多くの SIP の配備に共通するはずのいくつかの危機(threat)について詳述する。これらの危機(threat)は SIP が要求する各セキュリティサービスを説明するために特に選ばれている。

以下の例は SIP に対する危機(threat)の完全なリストを提供するものではない。そうではなくて、これらは全種類の危機(threat)を防ぐ可能性が高い特定のセキュリティサービスの必要性を明らかにする「よく知られている」危機(threat)である。

これらの攻撃は、攻撃者がネットワーク上のすべてのパケットを読むことができる可能性がある環境を想定している。SIP はパブリックインターネット上で多く使用されることが予想される。ネットワーク上の攻撃者はパケットを修正できるかもしれない(おそらく何らかの進入された中継媒体で)。攻撃者は、サービスを盗み取り、通信の盗聴、あるいはセッションを混乱させることを望むかもしれない。

26.1.1 登録の乗っ取り (Registration Hijacking)

SIP の登録メカニズムはユーザエージェントがレジストラサーバに、(address-of-record(AOR)で指定される)ユーザの位置が特定される場所にあるデバイスとして、自身を認識することを可能にする。レジストラサーバは REGISTER メッセージの From ヘッダフィールドで主張されているアイデンティティを評価して、このリクエストが To ヘッダフィールドの address-of-record(AOR)に関連付けられているコンタクトアドレスを修正できるかどうか確定する。これら 2 つのフィールドは同じであることが多いとはいえ、第三者(third-party)がユーザの代わりにコンタクト先を登録することができる多くの有効な配備がある。

その一方、SIP リクエストの From ヘッダフィールドは UA の所持者によって任意に修正されることがありうる。そしてこれは悪意ある登録に対してドアを開くことになる。address-of-record(AOR)に関連付けられたコンタクト先の変更を認可されたパーティにうまくなりすました攻撃者は、たとえば URI に対する既存のすべてのコンタクト先の登録を取り消して攻撃者自身のデバイスを適切なコンタクトアドレスとして登録できる。それによって、なりすまされたユーザに対するすべてのリクエストを攻撃者のデバイスに向かわせる。

この危機(threat)は、リクエスト発信元の暗号的保証の欠如に頼る危機(threat)の仲間に属する。有用なサービスに相当するすべての SIP UAS(たとえば、SIP リクエストとトラディショナルな電話通話に相互に作用するゲートウェイ)は、それが受け取るリクエストを認証することでそれのリソースへのアクセスを制御することを望むかもしれない。たとえば SIP フォンといったエンドユーザの UA であっても、リクエストの

発信元のアイデンティティを確かめることに関心を持つ。

この危機(threat)は、SIP エンティティがリクエストの発信元を認証することを可能にするセキュリティサービスの必要性を明らかにする。

26.1.2 **サーバになりすます** (Impersonating a Server)

リクエストが行くことになるドメインは一般的に Request-URI で指定される。UA はリクエストを配送するために、通常このドメインのサーバに直接コンタクトする。しかしながら、攻撃者がリモートサーバになりすまして、UA のリクエストが別のパーティに途中で傍受/妨害される可能性が常に存在する。

たとえば、chicago.com ドメインのリダイレクトサーバが別のドメイン(biloxi.com)のリダイレクトサーバになりすます場合について考えてみる。ユーザエージェントは biloxi.com にリクエストを送るが、chicago.com のリダイレクトサーバが偽造されたレスポンスで答える。そのレスポンスは biloxi.com からのレスポンスに対する適切な SIP ヘッダフィールドを持つ。リダイレクトレスポンス中の偽造されたコンタクトアドレスは発信元の UA を不適切なあるいは安全でないリソースに導く、または biloxy.com に対するリクエストが成功することを単に妨害するかもしれない。

この危機(threat)の仲間には大量のメンバーがおり、その多くはクリティカルである。登録の乗っ取り危機(threat)とは逆に、biloxi.com に送られた登録が chicago.com によって傍受/妨害される場合について考えてみる。chicago.com は傍受/妨害した登録に対して偽造した 301(Moved Permanently)レスポンスで返答する。このレスポンスは biloxy.com から来るように見えるかもしれず、さらに chicago.com を適切なレジストラサーバとして指定する。発信元 UA からの以降のすべての REGISTER リクエストは chicago.com に行くことになるだろう。

この危機(threat)の防御には、UA がリクエストを送るサーバを UA が認証できる手段が要求される。

26.1.3 **メッセージボディの改竄** (Tamper with Massege Bodies)

当然のことだが、SIP UA は信頼できるプロキシサーバ経由でリクエストをルーティングする。その信頼がどのように確立されたかに関係なく(プロキシの認証については本節の別の場所で議論する)、UA はプロキシがリクエストをルーティングするが、そのリクエストに含まれるボディを検査あるいはおそらく修正しないことということを信用してもよい。

SIP メッセージボディをメディアセッションのためのセッション暗号化鍵を伝えるために使用する UA を考えてみる。それはシグナリングを適切に配送するためにコンタクトするドメインのプロキシサーバを信用するが、そのドメインの管理者が今後のメディアセッションを復号化できることを望まないかもしれない。さらに悪いことに、そのプロキシサーバに悪意がある場合、それは、仲介者(man-in-the-middle)として動作するかあるいは発信元 UA が要求したセキュリティ特性を変更するかのいずれかで、セッション鍵を修正するかもしれない。

この危機(threat)の仲間はセッション鍵にのみ適用するのではなく、SIP においてエンド・トゥ・エンドで伝えられるもっとも考えられる形のコンテンツにも適用する。これらにはとりわけ、ユーザに対して表示/再生されるべき MIME ボディ、SDP、あるいはカプセル化されたテレフォニーシグナルが含まれる。攻撃者は、たとえばそれ以降の音声通話を盗聴するために、RTP メディアストリームを盗聴装置に向けるために、

SDP ボディの修正を試みるかもしれない。

たとえば Subject といった SIP のいくつかのヘッダフィールドは、エンド・トゥ・エンドで意味を持つということにも注意すること。UA はボディと同様にこれらのヘッダフィールドも保護するかもしれない(たとえば、Subject ヘッダフィールドを変更する悪意のある中継媒体は重要なリクエストをスパムに見えるようにするかもしれない)。しかしながら、多くのヘッダフィールドはプロキシによって正当に検査されるか変更されるかするので、すべてのヘッダフィールドがエンド・トゥ・エンドで安全確保されるべきではない。

これらの理由により、UA は SIP メッセージボディと、ある限られた場合にはヘッダフィールドを、エンド・トゥ・エンドで安全確保することを望むかもしれない。ボディに対して要求されるセキュリティサービスには、機密性、完全性、および認証が含まれる。これらのエンド・トゥ・エンドサービスは、プロキシサーバのような中間媒体とのやりとりを安全にするために使用される手段から独立しているべきである。

26.1.4 **セッションの破壊** (Tearing Down Sessions)

最初のメッセージングによってダイアログが確立されるとすぐに、ダイアログおよび(または)セッションの状態を修正するそれ以降のリクエストが送られる。セッション中のプリンシパルが、そのようなリクエストが攻撃者によって偽造されないことを確実にできることが重要である。

第三者(third-party)パーティの攻撃者が、セッションのパラメータ(To tag、From tag など)を知るために、2 つのパーティによって共有されているダイアログのいくつかの最初のメッセージを捕らえ、セッションに BYE リクエストを挿入する場合について考えてみる。攻撃者は、どちらかの参加者から来たように見えるリクエストを偽造することを選ぶかもしれない。ターゲットから BYE を受け取るとすぐに、セッションは完了前に破壊される。

同様のセッション途中の危機(mid-session threat)には、セッションを変更する、偽造された re-INVITE の送信が含まれる(おそらく、セッションの安全性を低下させるか、盗聴攻撃の一部としてメディアストリームをリダイレクトする)。

この危機(threat)に対する最も効果的な対策は、BYE の送信者の認証である。この場合には、受信者は、(送信者の完全なアイデンティティを確認するのではなく)対応するダイアログが確立されていたのと同じパーティから BYE が来たということだけを知る必要がある。また、機密性のために攻撃者がセッションのパラメータを知ることができない場合、攻撃者は BYE を偽造できない。しかしながら、(プロキシサーバのような)ある種の中継媒体は、セッションが確立されるときにそれらのパラメータを検査する必要がある。

26.1.5 **サービス拒否および増幅** (Denial of Service and Amplification)

サービス拒否(DoS)攻撃は、通常特定のネットワークエレメントのインターフェースに過剰な量のネットワークトラフィックを送ることで、それを利用不可能にすることに焦点を当てている。分散サービス拒否(DDoS)攻撃は、一人のネットワークユーザが複数のネットワークホストにターゲットホストを多量のネットワークトラフィックであふれさせることを可能にする。

多くのアーキテクチャにおいて、SIP プロキシサーバは、世界中の IP エンドポイントからリクエストを受け入れるためにパブリックインターネットに面している。SIP は、SIP システムの実装者またはオペレータに認識されて対処されなければならない DDoS 攻撃の多くの潜在的な機会を作り出す。攻撃者は、変造され

たソース IP アドレス、およびリクエストの発信元としてターゲットにされたホストを識別する Via ヘッダフィールドを含む偽のリクエストを生成し、このリクエストを多くの SIP ネットワークエレメントに送る。それによって、不運な SIP UA またはプロキシを、ターゲットを狙った DoS トラフィックを生成するために使用する。

同様に、攻撃者はリクエスト中でターゲットホストを識別する変造された Route ヘッダフィールド値を使い、そのようなメッセージを、ターゲットに送られるメッセージングを増幅するフォークを行うプロキシに送るかもしれない。

リクエストによって開始される SIP ダイアログが、反対方向に向けて発信される多数のトランザクションを生み出すことになると攻撃者が確信するときは、同様な効果を出すために Record-Route が使用されることがありうる。

レジストラサーバによって REGISTER リクエストが適切に認証され認可されなければ、多くの DoS 攻撃が利用できるようになる。攻撃者は管理ドメインの何人かあるいはすべてのユーザの登録を削除でき、それによってそれらのユーザが新規セッションに招待されることを妨げる。攻撃者はまた、レジストラサーバおよび関連するいかなるプロキシサーバでも DoS 攻撃における増幅器として使えるように、与えられた address-of-record(AOR)に対して同一のホストを指定する多量のコンタクト先を登録することがありうる。攻撃者はまた、非常に多くのバインディングを登録することによってレジストラサーバの利用可能なメモリおよびディスクリソースを枯渇することを試みるかもしれない。

SIP リクエストを送信するためにマルチキャストを使用することは、DoS 攻撃の可能性を劇的に増大させる可能性がある。

これらの問題は、DoS のリスクを最小化するアーキテクチャを定義する一般的な必要性、およびこのクラスの攻撃のセキュリティメカニズムのための推奨事項を心に留める必要性を明らかにする。

26.2 セキュリティメカニズム (Security Mechanisum)

上述の危機(threat)から、SIP プロトコルに要求される基本的なセキュリティサービスは、メッセージングの機密性と完全性を保持すること、リプレイ攻撃あるいはメッセージのなりすましを防ぐこと、セッション参加者の認証とプライバシーを提供すること、および DoS 攻撃を防ぐこと、であることをまとめた。SIP メッセージのボディは、機密性、完全性、および認証のセキュリティサービスを単独で要求する。

SIP 固有の新しいセキュリティメカニズムを定義するのではなく、SIP は可能であればどのような場合でも、HTTP と SMTP 空間に由来する既存のセキュリティモデルを再利用する。

メッセージの完全な暗号化は、シグナリングの機密性を保持するのに最適な手段を提供する。それはまた、悪意のあるいかなる中継媒体によってもメッセージが修正されないことを保証する。しかしながら、Request-URI、Route、そして Via のようなメッセージフィールドは、SIP リクエストが正しくルーティングされるようにほとんどのネットワークアーキテクチャにおいてプロキシに対して可視である必要があるので、SIP のリクエストとレスポンスはその全体を単純にエンド・トゥ・エンドで暗号化できない。プロキシサーバは、SIP が機能するためにメッセージのいくつかの機能も修正する必要があることにも注意すること(たとえば、Via ヘッダフィールド値を追加する)。したがってプロキシサーバは、SIP UA にある程度信頼さ

れなければならない。この目的のために、SIP のための下位レイヤのセキュリティメカニズムが推奨される。それは、SIP のリクエストまたはレスポンス全体をホップバイホップで回線上で暗号化し、エンドポイントがリクエストを送るプロキシサーバのアイデンティティを検証することを可能にする。

SIP エンティティはまた、安全な方法でお互いを認識する必要もある。SIP エンドポイントが相手 UA またはプロキシサーバに対してそれのユーザのアイデンティティを表明するとき、そのアイデンティティは何らかの方法で検証可能であるべきである。SIP ではこの要求に対応するために、暗号化認証メカニズムが提供される。

SIP メッセージボディのための独立したセキュリティメカニズムは、エンド・トゥ・エンドの相互認証の代替手段を提供するとともに、ユーザエージェントが中継媒体を信用しなければならないというある程度の制限を与える。

26.2.1 トランスポートレイヤとネットワークレイヤのセキュリティ (Transport and Network Layer Security)

トランスポートレイヤまたはネットワークレイヤのセキュリティは、メッセージの機密性と完全性を保証しながら、シグナリングトラフィックを暗号化する。

下位レイヤのセキュリティの確立時にはしばしば証明書が使用され、これらの証明書は多くのアーキテクチャにおいて認証手段を提供するためにも使用される。

トランスポートレイヤとネットワークレイヤでセキュリティを提供するよく知られている 2 つの選択肢は、それぞれ TLS(参考文献[25])と IPSec(参考文献[26])である。

IPSec はネットワークレイヤプロトコルツールのセットであり、伝統的な IP(Internet Protocol)の安全な代替として使用できる。IPSec は、ホストの組または管理ドメインが互いに既存の信頼関係を持つアーキテクチャにおいて、もっとも普通に使用される。IPSec は、ホストの OS レベルで、または(VPN アーキテクチャでのように)特定のインターフェースから受け取るすべてのトラフィックに対して機密性と完全性を提供するセキュリティゲートウェイ上で、通常は実装される。IPSec はまた、ホップバイホップでも使用できる。

多くのアーキテクチャにおいて IPSec は SIP アプリケーションとの統合を要求しない。IPSec はおそらく、SIP ホストに直接セキュリティを追加するのが困難であろう配備に最適である。最初のホップのプロキシサーバと、あらかじめ共有されている鍵関係を持つ UA も、IPSec を使用する候補である。SIP のための IPSec のいかなる配備も、SIP の安全を確保するために要求されるプロトコルツールを記述する IPSec プロファイルを要求する。このドキュメントではそのようなプロファイルは与えていない。

TLS は、コネクション指向のプロトコル上で(このドキュメントでは TCP)トランスポートレイヤのセキュリティを提供する。「tls」(TCP 上の TLS をあらわす)を Via ヘッダフィールド値または SIP URI の中で、希望するトランスポートプロトコルとして指定できる。TLS は、既存の信頼関係がないホスト間にホップバイホップのセキュリティが要求されるアーキテクチャに最適である。たとえば、Alice は彼女のローカルプロキシサーバを信頼している。そのプロキシサーバは証明書の交換後に Bob のローカルプロキシサーバ(Bob はこれを信頼している)を信頼することを決定する。したがって、Bob と Alice は安全に通信できる。

TLS は SIP アプリケーションと緊密に結び付けられなければならない。トランスポートメカニズムは SIP ではホップバイホップで定められるので、プロキシサーバに TLS 上でリクエストを送る UA はエンド・トゥ・エンドで TLS が使用されることの保証をなんら持たない。

SIP アプリケーションで TLS が使用されるときは、TLS_RSA_WITH_AES_128_CBC_SHA 暗号スイート[6]が実装者によって最低限サポートされなくてはならない[MUST]。下位互換のために、プロキシサーバ、リダイレクトサーバ、およびレジストラサーバは TLS_RSA_WITH_3DES_EDE_CBC_SHA をサポートするべきである[SHOULD]。実装者はまた、他のいかなる暗号スイートもサポートしてもよい[MAY]。

26.2.2 SIPS URI スキーム (SIPS URI Scheme)

SIPS URI スキームは、スキーム文字列が「sip」ではなく「sips」であるが、SIP URI の構文(19 で述べられている)を遵守する。しかしながら、SIPS の意味合い(semantics)は SIP URI と大きく異なる。SIPS は、リソースに安全に到達するべきであることをそのリソースが指定することを可能にする。

SIPS URI は特定のユーザのための address-of-record(AOR)(そのユーザが(ユーザの名刺、ユーザのリクエストの From ヘッダフィールド、REGISTER リクエストの To ヘッダフィールドで)標準的(canonically)に知られている URI)として使用できる。リクエストの Request-URI として使用されるとき、SIPS スキームは、Request-URI のドメイン部分に対して責任を負う SIP エンティティに到達するまで、リクエストがフォワードされる各ホップが TLS で安全を確保されなければならないことを表す。それが当該のドメインに到達するとすぐに、ローカルセキュリティとルーティングポリシーにしたがって、かなりの可能性で UAS への最後のホップのために TLS を使用して、ハンドリングされる。リクエストの発信元によって使用されるとき(リクエストの発信元が SIPS URI をターゲットの address-of-record(AOR)として使用する場合のように)、SIPS は、ターゲットドメインへのリクエストパス全体が安全確保されることを決定する。

SIPS スキームは、Request-URI に加えて、address-of-record(AOR)、コンタクトアドレス(Contact ヘッダのコンテンツ(REGISTER メソッドのそれも含む))、および Route ヘッダの中に含めることで、今日 SIP URI が SIP で使用されるその他の多くのやり方に適用できる。それぞれの場合において、SIPS URI スキームはこれら既存のフィールドが安全なリソースを指定することを可能にする。SIPS URI がこれらのいかなるコンテキストで参照される方法も、参考文献[4]で詳述されているそれ自身のセキュリティ特性を持つ。

SIPS の使用は、暗号スイート TLS_RSA_WITH_AES_128_CBC_SHA が使用されるべき[SHOULD]ことと同様に、相互 TLS 認証が使用されるべきである[SHOULD]ことを必然とする。認証処理で受け取った証明書は、クライアントが保持するルート証明書で検証されるべきである[SHOULD]。証明書の検証失敗は、リクエストの失敗という結果になるべきである[SHOULD]。

> SIPS URI スキームにおいて、トランスポートは TLS に依存せず、したがって「sips:alice@atlanta.com;transport=tcp」と「sips:alice@atlanta.com;transport=sctp」はともに有効である(そうではあるが、UDP は SIPS にとって有効なトランスポートではないということに注意すること)ということに注意すること。「transport=tls」の使用は、リクエストのひとつのホップにはっきりと限定されることを理由のひとつとして、結果的に反対された。これは RFC2543 からの変更である。

SIPS URI を address-of-record(AOR)として配布するユーザは、安全ではないトランスポート上でリクエストを拒否するデバイスをオペレートすることを選択するかもしれない。

26.2.3 HTTP 認証 (HTTP Authentication)

SIP は HTTP 認証の基づくチャレンジ能力を提供する。それはチャレンジと信用証明書を伝えるヘッダフィールドと、401 および 407 レスポンスコードに依存する。SIP における HTTP ダイジェスト認証スキームの再利用は、大きな修正をすることなしに、リプレイ防御と一方向認証を可能にする。

SIP におけるダイジェスト認証の使用方法は 22 節で詳述されている。

26.2.4 S/MIME

上で議論したように、機密性のためにエンド・トゥ・エンドで SIP メッセージ全体を暗号化することは、ネットワークの中継媒体(プロキシサーバのような)がメッセージを正しくルーティングするために特定のヘッダフィールドを見る必要があるので適切ではなく、これらの中継媒体がセキュリティのつながりから除外される場合には、SIP メッセージは本質的にルート不可能になる。

しかしながら、S/MIME は SIP UA が SIP 内で MIME ボディを暗号化し、メッセージヘッダに影響を及ぼすことなくこれらのボディをエンド・トゥ・エンドで安全にすることを可能にする。S/MIME は、相互認証と共に、メッセージボディのためのエンド・トゥ・エンドの機密性と完全性を提供できる。S/MIME を、SIP メッセージトンネリングを介して SIP ヘッダフィールドのための完全性と機密性のフォームを提供するために使用することも可能である。

SIP における S/MIME の使用方法は 23 節で詳述されている。

26.3 セキュリティメカニズムの実装 (Implemnting Security Mechanisms)

26.3.1 SIP の実装者に対する要求事項 (Requirements for Implementers of SIP)

プロキシサーバ、リダイレクトサーバ、およびレジストラサーバは TLS を実装しなくてはならず[MUST]、相互認証と一方向認証の双方をサポートしなくてはならない[MUST]。UA が TLS を開始する能力があることが強く推奨される[RECOMMENDED]。UA はまた、TLS サーバとして動作する能力を持ってもよい[MAY]。プロキシサーバ、リダイレクトサーバ、およびレジストラサーバは、それらの標準的な(canonical)ホスト名に対応する subject を持つサイト証明書を所持するべきである[SHOULD]。UA は TLS での相互認証のためにそれ自身の証明書を持ってもよい[MAY]が、このドキュメントではそれらの使用法についての規定を示さない。TLS をサポートするすべての SIP エレメントは、TLS ネゴシエーション中に受け取った証明書を検証するためのメカニズムを持たなくてはならない[MUST]。これには認証機関(望むべくは、Web ブラウザのためのルート証明書を発行するディストリビューターに匹敵する、よく知られたサイト証明書ディストリビューター)で発行された一つ以上のルート証明書を所持することが必要になる。

TLS をサポートするすべての SIP エレメントは、SIPS URI スキームもサポートしなくてはならない[MUST]。

プロキシサーバ、リダイレクトサーバ、レジストラサーバ、および UA は、IPSec もしくは他の下位レイヤのセキュリティプロトコルも実装してもよい[MAY]。

UA がプロキシサーバ、リダイレクトサーバ、またはレジストラサーバにコンタクトを試みるとき、その UAC は TLS コネクションを開始するべきである[SHOULD](その TLS コネクション上で SIP メッセージを送る)。いくつかのアーキテクチャでは、UAS もまたそのような TLS コネクションでもリクエストを受け取ってもよ

い[MAY]。

プロキシサーバ、リダイレクトサーバ、レジストラサーバ、および UA は、22 節で要求されているすべてのアスペクトを包含するダイジェスト認証を実装しなくてはならない[MUST]。プロキシサーバ、リダイレクトサーバ、およびレジストラサーバは、少なくとも一つのダイジェスト領域(Digest realm)を設定されるべき[SHOULD]であり、与えられたサーバでサポートされる「realm」文字列はそのサーバのホスト名またはドメイン名に一致するべきである[SHOULD]。

UA は MIME ボディの署名と暗号化、および 23 節で述べられているような S/MIME での信用証明書の移動をサポートしてもよい[MAY]。UA が TLS または IPSec のための証明書を検証するために一つ以上の認証機関のルート証明書を所持する場合、それは必要に応じて S/MIME の証明書を検証するためにこれらを再利用することが可能であるべきである[SHOULD]。UA は S/MIME の証明書を検証することだけのためにルート証明書を所持してもよい[MAY]。

S/MIME 実装が出てきて問題がさらにわかってきた時点で、将来のセキュリティ拡張が S/MIME に関する規範的な強度をアップグレードするかもしれないことが予想される、ということに注意すること。

26.3.2 セキュリティソリューション (Security Solutions)

これらのセキュリティメカニズムを合わせたオペレーションは、既存の Web および E メールのセキュリティモデルにある程度したがうことができる。高いレベルでは、UA はダイジェストのユーザ名とパスワードでそれ自身をサーバ(プロキシサーバ、リダイレクトサーバ、およびレジストラサーバ)に対して認証し、サーバは TLS で配送されたサイト証明書でそれ自身をホップ一つ先の UA に対してまたはホップ一つ先の別のサーバに対して(あるいはその逆で)認証する。

ピア・トゥ・ピアレベルでは、通常 UA はお互いを認証するためにネットワークを信頼する。しかしながら、ネットワークが直接認証(direct authentication)を提供しないか、あるいはネットワーク自体が信用できない場合、直接認証を提供するために S/MIME も使用できる。

以下は、26.1 節で述べられている種類の危機(threat)を防ぐために、これらのセキュリティメカニズムが様々な UA とサーバによって使用される実例である。実装者とネットワーク管理者は本節の残りの部分で与えられる規範となるガイドラインに従ってもよい[MAY]とはいえ、これらは単に実装の例としてのみ提供されている。

26.3.2.1 登録 (Registrations)

UA がオンラインになり、それのローカル管理ドメインに登録するとき、それはレジストラサーバと TLS コネクションを確立するべきである[SHOULD](10 節で、UA がどのようにしてレジストラサーバに到達するかについて述べている)。レジストラサーバは UA に証明書を提供するべき[SHOULD]であり、その証明書によって識別されるサイトは UA が登録しようとするドメインに一致しなくてはならない[MUST]。たとえば、UA が「alice@atlanta.com」という address-of-record(AOR)の登録をしようとする場合、サイト証明書は atlanta.com ドメイン内のホストを識別しなければならない(たとえば sip.atlanta.com)。それが TLS の Certificate メッセージを受け取るとき、UA は証明書を検証して証明書によって識別されるサイトを検査するべきである[SHOULD]。証明書が無効であったり、取り消されている、またはそれが適切なパーティを識別しない場合、UA は REGISTER メッセージを送ってはいけない[MUST NOT]。そうでなければ登録処理を進める。

レジストラサーバから有効な証明書が提供されたとき、UA はそのレジストラサーバが UA をリダイレクト、パスワードを盗む、またはそれに似た何らかの攻撃を試みる攻撃者ではないことがわかる。

UA はそれから、レジストラサーバから受け取ったサイト証明書に一致する Request-URI に宛てられるべき[SHOULD]REGISTER リクエストを生成する。UA が既存の TLS コネクション上で REGISTER リクエストを送るとき、レジストラサーバは 401(Proxy Authentication Required)レスポンスでリクエストにチャレンジするべきである[SHOULD]。レスポンスの Proxy-Authentication ヘッダフィールド中の「realm」パラメータは、サイト証明書で以前に与えられたドメインと一致するべきである[SHOULD]。UAC がチャレンジを受け取るとき、それはユーザに信用証明書を促すか、またはチャレンジの「realm」パラメータに一致する鍵束から適切な信用証明書を取り出すべきである[SHOULD]。この信用証明書のユーザ名は REGISTER リクエストの To ヘッダフィールド中の URI の「userinfo」部分に一致するべきである[SHOULD]。ダイジェストの信用証明書が適切な Proxy-Authorization ヘッダフィールドに挿入されるとすぐに、REGISTER はレジストラサーバに対して再提出されるべきである。

レジストラサーバはユーザエージェントにそれ自身を認証することを要求するので、攻撃者にとってユーザの address-of-record(AOR)のための REGISTER リクエストを偽造することは難しいだろう。REGISTER は機密性がある TLS コネクション上で送られるので、攻撃者はリプレイ攻撃のために信用証明書を記録するために REGISTER を傍受することができないということにも注意すること。

いったん登録がレジストラサーバに受け入れられると、レジストラサーバがこの管理ドメインのユーザに対するリクエストが送られるプロキシサーバとしても動作する場合には、UA はこの TLS コネクションをオープンしたままにするべきである[SHOULD]。既存の TLS コネクションは、ちょうど登録を完了した UA に対してやってくるリクエストを配送するために再利用される。

UA はすでに TLS コネクションの反対側のサーバを認証しているので、このコネクション上をやってくるすべてのリクエストはそのプロキシサーバを通ってきたことがわかる。攻撃者は、そのプロキシサーバを通して送られたように見える偽のリクエストを生成できない。

26.3.2.2 **ドメイン間リクエスト** (Interdomain Requests)

Alice の UA が、bob@biloxi.com というリモート管理ドメインのユーザとのセッションを開始することを望むとする。また、ローカル管理ドメイン(atlanta.com)はローカルアウトバウンドプロキシをもっているものとする。

管理ドメインに対するインバウンドリクエストをハンドリングするプロキシサーバは、ローカルアウトバウンドプロキシとしても動作してもよい[MAY]。便宜上、atlanta.com の場合はこうであるとする(そうでなければ、この時点でユーザエージェントは別のサーバと新規 TLS コネクションを開始するだろう)。クライアントが前の節で述べられている登録処理を完了していると仮定すると、クライアントは他のユーザに INVITE リクエストを送るときに、ローカルプロキシサーバとの TLS コネクションを再利用するべきである[SHOULD]。UA はユーザに不必要に信用証明書を促すことを避けるために、INVITE 中でキャッシュされた信用証明書を再利用するべきである[SHOULD]。

ローカルアウトバウンドプロキシサーバが、UA によって INVITE で提示された信用証明書を検証したとき、

それはそのメッセージをどのようにルーティングするか決定するためにRequest-URIを検査するべきである[SHOULD](参考文献[4]参照)。Request-URIの「domainname」部分がbiloxi.comではなくローカルドメイン(atlanta.com)に一致した場合、プロキシサーバはどのようにしたら最もうまくリクエストされたユーザに到達できるかを決定するために、自身のロケーションサービスを検索する。

「alice@atlanta.com」がたとえば「alex@atlanta.com」にコンタクトを試みていたら、ローカルプロキシはAlexが登録を行ったときにレジストラサーバと確立したTLSコネクションに対してリクエストをプロキシしただろう。Alexはこのリクエストを彼の認証済みチャンネル上で受け取るので、Aliceのリクエストがローカル管理ドメインのプロキシサーバによって認可されていることを、彼は保証されている。

しかしながら、この場合にはRequest-URIがリモートドメインを指定する。atlanta.comのローカルアウトバウンドプロキシサーバは、したがってbiloxi.comのリモートプロキシサーバとTLSコネクションを確立するべきである[SHOULD]。このTLSコネクション中の双方の参加者はサイト証明書を所持するサーバなので、相互TLS認証が起こるべきである[SHOULD]。コネクションのそれぞれの側は、証明書にあらわれるドメイン名をSIPメッセージのヘッダフィールドと比較するということに注意して、他方の証明書を検証し検査するべきである[SHOULD]。たとえば、atlanta.comのプロキシサーバはこの時点で、リモート側から受け取った証明書がbiloxi.comのドメインと一致することを検証するべきである[SHOULD]。それがそのように行われてTLSネゴシエーションが完了し、2つのパーティ間で安全なチャンネルが確立されるとすぐに、atlanta.comのプロキシはbiloxi.comにINVITEリクエストをフォワードできる。

次にbiloxi.comのプロキシサーバは、atlanta.comのプロキシサーバの証明書を検査して、その証明書が主張するドメインをINVITEリクエストのFromヘッダフィールドの「domainname」部分と比較するべきである[SHOULD]。biloxiのプロキシは、プロキシ元の管理ドメインにマッチしないリクエストを拒否することを要求する、厳格なセキュリティポリシーを持ってもよい[MAY]。

スパムを生成するために利用されることが多いSMTPの「オープンリレー」に等価なものをSIPで防ぐために、このようなセキュリティポリシーを設けてもよい。

しかしながら、このポリシーは、リクエストがそれ自身のものだとするドメインから来たことのみを保証する。それは、atlanta.comがどのようにAliceを認証したかということをbiloxi.comが確認することを認めない。biloxi.comがatlanta.comの認証ポリシーを知る他の何らかの方法を持つ場合に限り、Aliceが彼女のアイデンティティをどのように証明したかを確認できる可能性がある。biloxi.comはそれから、biloxi.comと共通の認証ポリシーを共有するために、管理上知らないドメインから来るリクエストを禁止するさらに厳格なポリシーを制定できる。

INVITEがbiloxiのプロキシによって承認されるとすぐに、プロキシサーバは、このリクエストのターゲットであるユーザ(この場合は「bob@biloxi.com」)に関連付けられている既存のTLSチャンネル(もしあれば)を識別するべきである[SHOULD]。INVITEはこのチャンネルを通してBobにプロキシされるべきである。リクエストは以前にbiloxiのプロキシであるとして認証されたTLSコネクション上で受け取られるので、BobはFromヘッダフィールドが改竄されておらず、atlanta.comはAliceを検証済みであることを知る(Aliceのアイデンティティを信じるかどうかは必ずしも確かではないが)。

それらがリクエストをフォワードする前に、双方のプロキシサーバは、このダイアログの以降のすべてのリクエストがこれらのプロキシサーバを通過するように、リクエストに Record-Route ヘッダフィールドを追加するべきである[SHOULD]。それによってそのプロキシサーバはこのダイアログの存続する間、セキュリティサービスを提供し続けることができる。そのプロキシサーバが Record-Route にそれら自身を追加しない場合、以降のメッセージは Alice と Bob の間をいかなるセキュリティサービスもなしで(S/MIME のようなんらかの独立したエンド・トゥ・エンドのセキュリティに 2 つのパーティが合意していなければ)、エンド・トゥ・エンドで直接渡る。この点に関して、SIP トラペゾイドモデルは、サイトプロキシ間の合意の約束事が Alice と Bob の間で十分に安全なチャンネルを提供できる、良い仕組みを提供する。

このアーキテクチャ上で獲物を狙う攻撃者は、たとえば、BYE リクエストを偽造してそれを Bob と Alice の間のシグナリングストリームに挿入することができない。その理由は、攻撃者がセッションのパラメータを確認する方法がなく、完全性メカニズムが Alice と Bob の間のトラフィックを推移的に(transitively)防御するからである。

26.3.2.3 **ピア・トゥ・ピアのリクエスト** (Peer-to-Peer Requests)

それとは別に、ローカルアウトバウンドプロキシを持たない、carol@chicago.com というアイデンティティを主張する UA を考えてみる。Carol が bob@biloxi.com に INVITE を送ることを望むとき、彼女の UA は biloxi のプロキシと(与えられた Request-URI にどうしたら最もうまく到達できるかを決定するために、参考文献[4]で述べられているメカニズムを使用して)直接 TLS コネクションを開始するべきである[SHOULD]。彼女の UA が biloxi のプロキシから証明書を受け取るとき、彼女が TLS コネクションで INVITE を送る前にそれが検証されるべきである[SHOULD]。Carol は biloxi のプロキシに彼女のアイデンティティを提供 する手段をもたないが、彼女は INVITE の「message/sip」ボディ上に CMS 分離署名を持っている。この場合、Carol は biloxi.com と正式な関係を持たないので、彼女が biloxi.com の realm に何らかの信用証明書を持つことはおそらくないだろう。biloxi のプロキシは、From ヘッダフィールドの「domainname」部分に biloxi.com を持たないリクエストにチャレンジすることさえ不可能にする厳格なポリシーを持ってもよい[MAY]。それはこれらのユーザを認可されていない(unauthenticated)として処理する。

biloxi のプロキシは、認可されていないすべてのリクエストが bob@biloxy.com(すなわち、<sip:bob@192.0.2.4>)に対して登録されている適切なコンタクトアドレスにリダイレクトされるべきである、という Bob のためのポリシーを持つ。Carol は彼女が biloxi のプロキシと確立した TLS コネクション上でリダイレクトレスポンスを受け取るので、彼女はそのコンタクトアドレスの真実性を信用する。

Carol はそれから、指定されたアドレスと TCP コネクションを確立して、受け取ったコンタクトアドレスを含む Request-URI を持つ新規 INVITE を送るべきである[SHOULD](リクエストが万全になるようにボディ中の署名を再計算して)。Bob はこの INVITE を安全ではないインターフェース上で受け取るが、彼の UA はリクエストの From ヘッダフィールドを検査し、(この場合には)それを認識する。そして引き続いてローカルにキャッシュされている信用証明書と INVITE のボディの署名中の信用証明書をマッチする。彼は同様の方法でレスポンスし、彼自身を Carol に対して認証し、そして安全なダイアログが開始する。

時折、管理ドメインのファイアウォールや NAT が、UA に対する直接の TCP コネクションの確立を不可能にすることがある。この場合、プロキシサーバーはもしかすると信頼という意味合いを持たない方法で、ローカルポリシーが指示するようにリクエストを UA にリレーできるかもしれない

(たとえば、既存の TLS コネクションの使用を見合わせてリクエストを平文の TCP 上でフォワードして)。

26.3.2.4 DoS 防御 (DoS Protection)

これらのセキュリティソリューションを利用するアーキテクチャに対するサービス拒否攻撃のリスクを最小限に抑えるため、実装者は以下のガイドラインに注意するべきである。

SIP プロキシサーバが運用されているホストがパブリックインターネットからルート可能なとき、それは防御的な運用ポリシー(ソースルーティングされたトラフィックをブロックして、望ましくは ping トラフィックをフィルターして)で管理ドメインに配備されるべきである[SHOULD]。TLS と IPSec の双方は管理ドメインの末端で、安全なトンネルとソケットを集約するためにセキュリティアソシエーションに参加する要塞ホストも使用できる。これらの要塞ホストは、管理ドメイン内の SIP ホストが不要なメッセージングに邪魔されないことを保証しながら、サービス拒否攻撃の矢面に立つこともできる。

どのようなセキュリティソリューションが配備されるかに関係なく、プロキシサーバに向けられた膨大な量のメッセージはプロキシサーバのリソースを動かなくし、望ましいトラフィックがデスティネーションに到達することを妨げることがある。プロキシサーバで SIP トランザクションを処理することに関連する計算コストが存在し、そのコストはステートレスプロキシサーバに対するものよりもステートフルプロキシサーバに対するものの方が大きい。したがって、ステートフルプロキシはステートレスプロキシサーバよりも膨大な量のメッセージの影響を受けやすい。

UA とプロキシサーバは、疑わしいリクエストに、通常のレスポンスの再送アルゴリズムを使わないで一つの 401(Unauthorized)または 407(Proxy Authentication Required)のみでチャレンジするべきであり[SHOULD]、結果として認証されていないリクエストにはステートレスに動作する。

401(Unauthorized)または 407(Proxy Authentication Required)ステータスレスポンスの再送は、トラフィックを第三者(third-party)パーティに向かわせるために変造したヘッダフィールド値(たとえば Via)を使用する攻撃者の問題を増大する。

まとめると、TLS のようなメカニズムを介したプロキシサーバの相互認証は、性質の悪い中継媒体がサービスを拒否することができる変造されたリクエストやレスポンスを持ち込む可能性を劇的に減らす。これは攻撃者が無害な SIP ノードを増幅を仲介するものにすることを十分に難しくする。

26.4 制限事項 (Limitations)

これらのセキュリティメカニズムは、慎重に適用されるときには多くの危機(threat)を防ぐが、実装者とネットワークオペレータが理解しておかなければならない、メカニズムの適用範囲における制限事項がある。

26.4.1 HTTP ダイジェスト認証 (HTTP Digest)

SIP で HTTP ダイジェスト認証を使用する際の主要な制限の一つは、ダイジェスト認証の完全性メカニズムが SIP ではあまり良く機能しないということである。具体的には、それは Request-URI とメッセージのメソッドの保護は提供するが、UA がもっとも安全確保を望む可能性が高い、いかなるヘッダフィールドの保護も提供しない。

RFC2617 で述べられている既存のリプレイ防御メカニズムも SIP ではいくつかの制限がある。たとえば、next-nonce メカニズムはパイプラインリクエストをサポートしない。リプレイ防御には nonce-count メカニズムが使用されるべきである。

HTTP ダイジェスト認証の別の制限は、realm の適用範囲である。ダイジェスト認証は、ユーザがリソース(ユーザが契約しているサービスプロバイダのように、それとの間にあらかじめ存在する関係がある(これはとても一般的なケースであり、よってダイジェスト認証は非常に有用な機能を提供する))に対してユーザ自身を認証することを望むときに有用である。それとは対照的に、TLS の適用範囲は、証明書がグローバルに検証可能であることが多いので、ドメイン間あるいは複数 realm におよび、そのため UA はあらかじめ存在する関係を持たないサーバを認証できる。

26.4.2 S/MIME

S/MIME メカニズムの最大の未解決の欠陥は、エンドユーザに対して広く普及している公開鍵インフラの欠如である。自己署名証明書(すなわちダイアログ中の一方の参加者が検証できない証明書)が使用される場合、23.2 節で述べられている SIP ベースの鍵交換メカニズムは、攻撃者が S/MIME ボディを検査して修正する可能性がある中間一致攻撃(man-in-the-middle attack)の影響を受けやすい。攻撃者はダイアログの 2 つのパーティ間の最初の鍵交換を傍受し、既存の CMS 分離署名をリクエストとレスポンスから取り除き、攻撃者が供給する証明書(正しい address-of-record(AOR)に対する証明書であるように見える)を含む別の CMS 分離署名を挿入する必要がある。それぞれのパーティは、実際には攻撃者の公開鍵を持つが、相手と鍵を交換したと思うだろう。

攻撃者は、2 つのパーティ間の最初の鍵交換のときにのみこの脆弱性を利用できるということに注意することが重要である。それ以降の場合には、鍵の改変は UA にとって検知可能である。攻撃者が長い間(可能性として、何日も、何週間も、あるいは何年も経過して)2 つのパーティ間の以降のすべてのダイアログのパスに留まりつづけることも難しいだろう。

SSH は最初の鍵交換時に同じ中間一致攻撃(man-in-the-middle attack)の影響を受けやすい。SSH は完全でないけれども、その一方でコネクションのセキュリティを向上させることは広く認められている。鍵指紋の使用は、SSH と同じように SIP にもいくらかの支援を提供するだろう。たとえば、2 つのパーティが音声通話セッションを確立するために SIP を使用する場合、各々は相手から受け取った鍵の指紋を読み取るだろう。そしてそれはオリジナルと比較される。間にいる者(man-in-the-middle)にとって、参加者のシグナリングをエミュレートする(クリッパーチップベースの安全な電話で使用された行為)よりも音声をエミュレートするほうがはるかに難しいに違いない。

S/MIME メカニズムは、UA が暗号化されたリクエストを前置きなしに送ることを可能にする(UA が鍵束にデスティネーションの address-of-record(AOR)に対する証明書を所持する場合)。しかしながら、address-of-record(AOR)のために登録されたどの個別のデバイスも、デバイスの現在のユーザによって前回採用された証明書を保持しないことがありうる。そしてそれゆえに、暗号化されたリクエストを適切に処理することはできない。このことは結局、回避可能な何らかのエラーシグナリングとなる。これは特に暗号化されたリクエストがフォークされたときに起こる可能性が高い。

S/MIME に関連する鍵は、デバイス(UA)ではなく特定のユーザ(address-of-record(AOR))に関連付けられたときにもっとも有用である。ユーザがデバイス間を移動するとき、UA 間で秘密鍵を安全に送信することは難

しいかもしれない。そのような鍵がどのようにデバイスに取得されるかはこのドキュメントの適用範囲外である。

そのほかの S/MIME メカニズムのより平凡な問題は、巨大なメッセージが、特に 23.4 節で述べられている SIP トンネリングメカニズムが使用されるときに、生成されることになるということである。そのため、S/MIME トンネリングが使用されるときはトランスポートプロトコルとして TCP を使用することを推奨する[RECOMMENDED]。

26.4.3 TLS

TLS に関して最も一般的に表明される懸念は、TLS が UDP 上で走ることができないということである。すなわち、TLS は基礎となるコネクション指向のプロトコルを必要とする(それはこのドキュメントでは TCP を意味する)。

ローカルアウトバウンドプロキシサーバおよび(または)レジストラサーバにとって、多くの UA と長時間存続する TLS コネクションを同時にたくさん維持することも困難かもしれない。これはいくつかのスケーラビリティに関する妥当な懸念を、特に集約的な暗号スイートに対して、もたらす。長時間存続する TLS コネクションの冗長性を維持することも、特に UA が単独でそれらの確立に責任を負うときに、厄介かもしれない。

TLS は、SIP エンティティがそれの隣接するサーバを認証することのみを認める。TLS は厳密にホップバイホップのセキュリティを提供する。TLS あるいはこのドキュメントで規定されているいかなるメカニズムも、クライアントがそれが直接の TCP コネクションを形成することができないプロキシサーバを認証することを認めない。

26.4.4 SIPS URI

TLS をリクエストパスのすべてのセグメントで実際に使用することは、終端の UAS が TLS 上で到達可能でなければならないことを必要とする(おそらくは、コンタクトアドレスとして SIPS URI を登録して)。これが SIPS の望ましい用法である。しかしながら、多くの有効なアーキテクチャは、リクエストパスの一部分の安全を確保するために TLS を使用し、たとえば、UAS への最後のホップに対しては他の何らかのメカニズムに頼る。したがって SIPS は、TLS の用法が真にエンド・トゥ・エンドであることを保証できない。多くの UA はやってくる TLS コネクションを受け入れないので、たとえ TLS をサポートする UA であっても上記の TLS の制限事項の節で述べられているように、UAS として TLS 上のリクエストを受け取るために、持続する TLS コネクションを維持することを要求されるかもしれない、ということに注意すること。

SIPS Request-URI に対する SIPS バインディングを提供するために、ロケーションサービスは必須とされない。ロケーションサービスは(10.2.1 節で述べられているように)一般的にユーザ登録で利用されるが、他の様々なプロトコルおよびインターフェースが address-of-record(AOR)のためのコンタクトアドレスを供給することもたぶんあり得る。そしてこれらのツールは必要に応じて SIPS URI を SIP URI に自由にマッピングする。ロケーションサービスはバインディングを問い合わせられたときには、SIPS Request-URI を持つリクエストを受け取ったかどうかに関わらず、それのコンタクトアドレスを返送する。リダイレクトサーバがロケーションサービスにアクセスしている場合、コンタクトアドレスの正当性を決定するためにリダイレクトの Contact ヘッダフィールドを処理するかどうかは、そのエンティティ次第である。

TLS がターゲットドメインまでのリクエストセグメントのすべてで使用されることを保証することは、多少面倒である。非準拠または信用できない、途中にある暗号的に認証されたプロキシサーバが SIPS に関するフォワードのルール(および 16.6 節の一般的なフォワードのルール)を無視することを選ぶかもしれないことが見込まれる。そのような悪意のある中継媒体は、たとえば、安全性を落とす(downgrade)ための試みとしてリクエストのターゲットを SIPS URI から SIP URI に変更することがあり得る。

それとは別に、中継媒体が正当にリクエストのターゲットを SIPS URI から SIP URI に変更するかもしれない。そのため、Request-URI に SIPS URI スキームを使用するリクエストの受信者は、リクエストパス全体で(クライアントから先で)SIPS が使用されたということを Request-URI だけから仮定できない。

これらの懸念に対処するため、SIP URI または SIPS URI を含む Request-URI を持つリクエストの受信者は、To ヘッダフィールド値を検査してそれが SIPS URI を含むかどうか確認することが推奨される[RECOMMENDED](Request-URI の URI が To ヘッダフィールドの URI と同じスキームを持つが等価ではない場合でも、それはセキュリティの侵害にならないことに注意すること)。クライアントはリクエストの Request-URI と To ヘッダフィールドに違う値を入れることを選択するかもしれないが、SIPS が使用されるときにはこの不一致はセキュリティ侵害の可能性があると解釈され、結果的にリクエストは受信者に拒否されることがあり得る。受信者はまた、ローカル管理ドメインに到達するまでのリクエストパス全体で TLS が使用されたかどうかをダブルチェックするために Via ヘッダの連なりも検査してもよい[MAY]。エンド・トゥ・エンドで To ヘッダフィールドがオリジナルの形で伝えられることを保証する役に立つように、発信元の UAC が S/MIME を使用することもできる。

Request-URI のスキームがトランジット中に不適切に修正されたと信じる理由を UAS が持つ場合、UA はそれのユーザにセキュリティ侵害の可能性があることを通知するべきである[SHOULD]。

ダウングレード攻撃を防御するためのさらなる対策として、SIPS リクエストのみを受け入れるエンティティは、安全ではないポート上のコネクションを拒否してもよい[MAY]。

エンドユーザは間違いなく確実に SIPS URI と SIP URI の間の違いを見分けるだろう。そして無条件反射的にそれらをマニュアルで編集するかもしれない。これはセキュリティに貢献することもセキュリティを落とすこともあり得る。たとえば、攻撃者が、プロキシサーバのすべての SIPS レコードを事実上削除する偽のレコードセットを挿入して DNS キャッシュを改竄する場合、このプロキシサーバをトラバースするすべての SIPS リクエストは失敗するかもしれない。しかしながら、ユーザが SIPS address-of-record(AOR)に対する何度もの呼び出しが失敗しているのがわかるときは、SIPS から SIP にスキームをマニュアルで変換する何らかのデバイス上に呼び出しが行っていて再試行していることがあり得る。もちろんこれに対するいくつかの防護手段はあるが(デスティネーションの UA が真に偏執的な場合はすべての非 SIPS リクエストを拒否することがあり得る)、それは注意するに値する制限事項である。良い面をあげれば、ユーザが SIP URI だけを提示されるときでさえも「SIPS」を利用できることをユーザは知るだろう。

26.5 **プライバシー** (Privacy)

SIP メッセージは送信者に関する細心の注意を要する情報を含むことが多い。それには、伝えなければならないことだけではなく、誰と通話するのか、いつどれくらいの時間通話するのか、そしてどこからセッションに参加するのかも含まれる。多くのアプリケーションとそれのユーザは、この種のプライベート情報をそれを知る必要がないいかなるパーティからも隠すことを要求する。

プライベート情報が暴かれるそれほど直接的でない方法もある、ということに注意すること。ユーザまたはサービスが、人名と組織の所属(これは address-of-record(AOR)の大部分について述べている)から推測可能なアドレスで到達可能なことを選択する場合、リストに載っていない「電話番号」を持つことでプライバシーを保証する慣例的な方法は信用できない。ユーザロケーションサービスは、発信者に対するセッション招待の受信者の具体的な所在を暴露することで、セッション招待の受信者のプライバシーを侵害することがあり得る。したがって実装は各ユーザごとに、どの種類のロケーションと有効性の情報が特定のクラスの発信者に渡されるかを制限することができるべきである[SHOULD]。これが、進行中の SIP ワークで調査されることが期待されている問題のすべてのクラスである。

ときとしてユーザは、アイデンティティを運ぶヘッダフィールド中の情報を秘密にすることを望むかもしれない。これはリクエストの発信元を表す From および関連するヘッダだけに適用するのではなく、To にも適用する。スピードダイアルのニックネームやターゲットグループのための展開されていない識別子(unexpanded identifier)を最終デスティネーションに伝えることは適切でないかもしれず、リクエストがルーティングされるときにいずれかが Request-URI から削除されるだろう。しかし、その 2 つが最初に同じであれば、To ヘッダフィールドでは変更されない。したがって、Request-URI と異なる To ヘッダフィールドを生成することがプライバシー保護のために望まれてもよい[MAY]。

２７． IANA 条項 (IANA Considerations)

SIP アプリケーションで使用されるすべてのメソッド名、ヘッダフィールド名、ステータスコード、およびオプションタグは、RFC の IANA 条項の指示にしたがって IANA に登録される。

仕様は、IANA が http://www.iana.org/assignments/sip-parameters の下に 4 つの新規サブレジストリを生成することを指示する。その 4 つとは、そこに既に存在するヘッダフィールドのサブレジストリに追加される、オプションタグ(Option Tags)、警告コード(Warning Codes (warn-codes))、メソッド(Methods)、およびレスポンスコード(Response Codes)である。

27.1 オプションタグ (Option Tags)

この仕様は http://www.iana.org/assignments/sip-parameters の下にオプションタグ(Option Tags)サブレジストリを制定する。

オプションタグは、拡張に対する SIP 互換性メカニズムをサポートするために、Require、Supported、Proxy-Require、および Unsupported のようなヘッダフィールドで使用される(19.2 節参照)。オプションタグ自体は、特定の SIP オプション(すなわち、拡張)に関連付けられた文字列である。それは SIP エンドポイントに対するオプションを識別する。

オプションタグは standards track RFC で発表されたときに IANA によって登録される。RFC の IANA 条項の節は、発表された RFC の番号と共に、IANA レジストリに現れる以下の情報を含まなければならない。

- オプションタグの名称。名称はどのような長さでもよい[MAY]が、20 文字以下であるべきである[SHOULD]。名称は alphanum 文字(25 節参照)のみから成らなくてはならない[MUST]。

- 拡張を説明する説明文。

27.2 **警告コード** (Warn-Codes)

この仕様は http://www.iana.org/assignments/sip-parameters の下に警告コード(Warn-codes)サブレジストリを制定し、20.43 節にリストされている warn-codes のそれへの追加を開始する。さらなる warn-codes は RFC 刊行物によって登録される。

warn-codes の表のための説明文は以下のようである。

警告コードは、トランザクションの失敗がセッション記述プロトコル(SDP)(RFC2327 参考文献[1])の問題に起因するとき、SIP レスポンスメッセージのステータスコードを補足する情報を提供する。

「warn-code」は 3 桁から成る。最初の桁である 3 は SIP 固有の警告を示す。将来の仕様が 3xx 以外の warn-codes の使用を記述するまで、3xx warn-codes だけを登録できる。

警告の 300 から 329 はセッション記述のキーワードの問題を示すために予約されている。330 から 339 はセッション記述で要求された基本的なネットワークサービスに関係する警告。370 から 379 はセッション記述で要求された定量的な QoS パラメータに関係する警告。そして、390 から 399 は上記のカテゴリに当てはまらないその他の警告である。

27.3 **ヘッダフィールド名** (Header Field Names)

ここで述べることは、http://www.iana.org/assignments/sip-parameters の下のヘッダサブレジストリに関する IANA の指示を旧式なものとする。

新規ヘッダフィールド名を登録するために、以下の情報が RFC 文書で提供される必要がある。

- ヘッダが登録される RFC 番号
- 登録されることになるヘッダフィールド名
- 定義されていれば、そのヘッダフィールドのコンパクトフォーム

いくつかの一般的で広く使用されているヘッダフィールドは、一文字のコンパクトフォームに割り当ててもよい[MAY](7.3.3 節)。コンパクトフォームは SIP ワーキンググループのレビュー後にのみ割り当てでき、そのあとに RFC が発行される。

27.4 **メソッドとレスポンスコード** (Method and Response Codes)

この仕様は、http://www.iana.org/assignments/sip-parameters の下にメソッド(Method)とレスポンスコード(Response-Code)サブレジストリを制定し、以下のようにそれの定義を開始する。当初のメソッドの表は以下のとおりである。

INVITE	[RFC3261]
ACK	[RFC3261]
BYE	[RFC3261]

CANCEL [RFC3261]
REGISTER [RFC3261]
OPTIONS [RFC3261]
INFO [RFC2976]

レスポンスコードの表は当初21節の、通知(Informational)、成功(Success)、リダイレクト(Redirection)、クライアントエラー(Client-Error)、サーバエラー(Server-Error)、およびグローバル失敗(Global-Failure)が使用される。表は以下の形式をもつ。

タイプ(例: Informational)
番号 デフォルトのリーズンフレーズ [RFC3261]

新規レスポンスコードまたはメソッドを登録するために、以下の情報がRFC文書で提供される必要がある。

- メソッドまたはレスポンスコードが登録されるRFC番号
- 登録されることになるレスポンスコード番号またはメソッド名
- 必要に応じて、レスポンスコードに対するデフォルトのリーズンフレーズ

27.5 「message/sip」MIME タイプ (The “message/sip” MIME type)

このドキュメントでは、SIP メッセージが SIP 内のボディとしてトンネルされることを許可するために「message/sip」MIME メディアタイプを登録する。これは主としてエンド・トゥ・エンドの安全のためである。このメディアタイプは以下の情報で定義される。

メディアタイプ名: message
メディアサブタイプ名: sip
必要なパラメータ: なし

オプションのパラメータ: version
version: 同封されたメッセージの SIP バージョン番号(例: "2.0")。存在しない場合、version はデフォルトとして"2.0"になる。

エンコードスキーム: 8 ビットのヘッダからなる SIP メッセージ。
オプションとしてバイナリの MIME オブジェクトがそれに続く。そのときは、SIP メッセージはバイナリとして処理されなければならない。通常の状態では、SIP メッセージはバイナリデータの送信が可能なトランスポートで運ばれ、特別なエンコードは必要とされない。

セキュリティの考慮: 下記参照
S/MIME と協調したセキュリティメカニズムとしての、これの用法のモチベーションと例は 23.4 で与えられている。

27.6 新規 Content-Disposition パラメータの登録 (New Content-Disposition Parameter Registrations)

このドキュメントでは Content-Disposition ヘッダの 4 つの新しい「disposition-types」も登録する。その 4 つとは、alert、icon、session、および render である。これらの値が IANA レジストリで Content-Disposition のために記録されることを著者は要求する。

モチベーションと例を含め、これらの「disposition-types」の説明は、20.11 節で与えられている。

以下は、IANA レジストリにふさわしい簡単な説明である。

Alert　ボディはユーザに注意を喚起するためのカスタムの呼び出し音である
icon　ボディはユーザにアイコンとして表示される
render　ボディはユーザに表示されるべきである
session　ボディは、たとえば RFC2327 の SDP ボディと同様に、コミュニケーション
セッションを記述する

２８．RFC2543 からの変更点 (Changed From RFC2543)

この RFC は RFC2543 を改訂する。大部分は RFC2543 との下位互換を保っている。ここで述べられている変更点は、RFC2543 で発見された多くのエラーを修正し、RFC2543 で詳述されていないケースについての情報を提供する。プロトコルはここでは、さらにはっきりとしたレイヤモデルで提示されている。

われわれは相違点を、場合によっては相互運用性や正確なオペレーションに影響を与える RFC2543 からの重要な変更である機能的動作と、RFC2543 とは異なるが相互運用性の問題が発生する原因にならない機能的動作、に分割する。その上、ここでは文章化されていない、数え切れないほどの説明もあった。

28.1 重要な機能変更 (Major Functional Changes)

- 相手が呼び出しに答える前にその呼の切断を UAC が望むときは、CANCEL を送る。オリジナルの INVITE が依然として 2xx を送信する場合、UAC はそれから BYE を送る。BYE は、RFC2543 ではいつでも送ることができたのに対して、既存のコールレグ(この RFC ではダイアログと呼ぶようになった)上でのみ送ることができる。

- SIP の BNF は、RFC2234 に準拠するように変換された。

- SIP URL の BNF は、ユーザ部分でさらに大きな文字セットを許可するように、より一般化された。さらに、比較ルールがおもに大文字小文字を区別しないように単純化され、パラメータが存在するときの比較操作が詳述された。もっとも重要な変更は、初期値を持つパラメータを含む URI が、そのパラメータを持たない URI にマッチしないことである。

- Via の隠蔽を削除した。不明瞭にする処理を実行するためにネクストホップに頼っていたので、それは重大な信用問題(trust issue)を抱えていた。その代わりに、Via の隠蔽は、ステートフルプロキシでローカルの実装の選択で行うことができる。そのため記述されなくなった。

- RFC2543 では、CANCEL トランザクションと INVITE トランザクションが混ざり合っていた。それらは分割された。ユーザが INVITE を送り、 次いで CANCEL を送るとき、それでも INVITE トラ

ンザクションは正常に終了する。UAS はオリジナルの INVITE リクエストに対して 487 レスポンスでレスポンスする必要がある。

- 同様に、CANCEL トランザクションと BYE トランザクションも混ざり合っていた。RFC2543 は、UAS が BYE を受け取ったときは INVITE に対してレスポンスを送らないことを認めていた。ここではそれは認められない。オリジナルの INVITE はレスポンスを必要とする。

- RFC2543 では、UA は UDP をサポートすることだけが必要とされていた。この RFC では、UA は UDP と TCP をサポートすることを必要とされる。

- RFC2543 では、フォークを行うプロキシは、複数のチャレンジがある場合にはダウンストリームエレメントからひとつのチャレンジだけを渡されていた。この RFC では、プロキシはすべてのチャレンジを収集して、それらをフォワードされたレスポンスに入れることになっている。

- ダイジェスト認証の信用証明書では、URI は引用符に囲まれている必要がある。これは RFC2617 と RFC2069 では双方共に一貫性がないので、それらからははっきりわからない。

- SDP 処理は別の仕様(参考文献[13])に分割され、SIP を介して有効にトンネルされる、正式なオファー/アンサー交換処理として、さらに完全に規定された。SDP は、ベースライン SIP 実装で INVITE/200 または 200/ACK に対して認められる。RFC2543 は SDP を、ひとつのトランザクション中の INVITE、200、および ACK で使用する能力をほのめかしていたが、うまく規定されていなかった。より複雑な SDP の用法が拡張で認められる。

- URI および Via ヘッダフィールドでの IPv6 の完全なサポートを追加した。Via での IPv6 のサポートは、Via ヘッダフィールドのパラメータが角括弧とコロンを許可することを要求した。これらの文字は以前は許可されていなかった。理論上これは、古い実装との間に相互運用性の問題を生ずる。しかしながら、ほとんどの実装は、これらのパラメータ中ですべての ASCII の非コントロール文字を受け入れることが観測された。

- DNS の SRV 手順が別の仕様(参考文献[4])に文書化された。この手順は SRV と NAPTR リソースレコードの双方を使用し、もはや RFC2543 で述べられているようには SRV からのデータを結合しない。

- Max-Forwards の使用を必須にすることで、ループ検知がオプションになった。RFC2543 のループ検知手順には、エラーでないときにスパイラルをエラー状態であると報告するであろう重大なバグがあった。オプションのループ検知手順はさらに完全にそして正確にここで規定された。

- 今ではタグはダイアログ識別の基本的な要素なので、タグの使用が必須になった(RFC2543 ではタグはオプションだった)。

- クライアントがどの拡張をサポートするかをサーバに示すことを可能にする、Supported ヘッダフィールドを追加した。サーバはそれらの拡張をレスポンスに適用し、レスポンスの Require でそれらの使用法を示す。

- いくつかのヘッダフィールドの BNF から拡張パラメータが欠落していたので、それらが追加された。

- Route と Record-Route の構築操作は、RFC2543 では明確さがまったく足りず、アプローチも正しくなかった。それはこの仕様で、かなり書き直された(また非常に単純化された)。これはおそらく間違いなく最大の変更点である。最初のリクエストが Record-Route の外で何らかの方法で取得した Route ヘッダフィールド値のセットを持つ「あらかじめロードされたルート(pre-loaded routes)」を使用しない配備のために、下位互換性が依然として提供される。それらの状況では、新しいメカニズムは相互運用可能ではない。

- RFC2543 では、メッセージの行は CR、LF、あるいは CRLF で区切ることができた。この仕様では CRLF だけを許可する。

- CANCEL と ACK での Route の使用は RFC2543 ではよく定義されていなかった。それがしっかりと規定された。リクエストが Route ヘッダフィールドを持っていた場合、そのリクエストへの非 2xx レスポンスに対する CANCEL または ACK は同じ Route ヘッダフィールド値を伝える必要がある。2xx レスポンスに対する ACK は、その 2xx レスポンスの Record-Route から知った Route 値を使用する。

- RFC2543 はひとつの UDP パケットで複数のリクエストを認めていた。この用法は削除された。

- Expires ヘッダフィールドとパラメータでの絶対時間の用法は削除された。それは、時間が同期されていない(これは一般的に起こる)エレメントで相互運用性の問題を生じていた。その代わりに相対時間が使用される。

- Via ヘッダフィールド値の branch パラメータは、すべてのエレメントで使用することが必須になった。それは今では一意なトランザクション識別子の役割を演じる。これは、RFC2543 からの複雑でバグを抱えたトランザクション識別ルールを回避する。前のホップがそのパラメータをグローバルに一意にしたかどうかを確定するために、パラメータ値でマジッククッキーが使用され、それが存在しないときは比較は古いルールに戻る。したがって、相互運用性が保証される。

- RFC2543 では、TCP コネクションのクローズは CANCEL と等価であるとされていた。これはプロキシ間の TCP コネクションに対しては、実装がほとんど不可能(そして間違い)だった。これは、TCP コネクション状態と SIP 処理の間につながりがないので、撤廃された。

- RFC2543 は、UA が別のトランザクションが進行中に新しいトランザクションを開始できるかどうかについて触れていなかった。それがここで規定された。それは非 INVITE リクエストでは許可され、INVITE では却下される。

- PGP が削除された。それは十分に規定されていなかった。また、さらに完成された PGP MIME と互換性がなかった。それは S/MIME で置き換えられた。

- エンド・トゥ・エンドの TLS のために「sips」URI スキームが追加された。このスキームは RFC2543 と下位互換がない。Request-URI 中に SIPS URI スキームを持つリクエストを受け取る既存のエレメントは、おそらくそのリクエストを拒否するだろう。これは実際のところ機能であるといえる。つまりそれは、SIPS URI に対する呼がパスのすべてのホップが安全確保されている場合にのみ配送されることを保証する。

- 追加のセキュリティ機能が TLS と共に追加された。これらについては、さらに長く完成されたセキュリティの考慮節で述べられている。

- RFC2543 では、アップストリームに 101 から 199 までの暫定レスポンス(provisional response)をフォワードするためにプロキシは要求されなかった。これは MUST に変更された。それ以降の多くの機能は 101 から 199 までのすべての暫定レスポンス(provisional response)の配送に依存するので、これは重要である。

- RFC2543 では 503 レスポンスコードについてほとんど語られていなかった。それは、プロキシにおいて失敗あるいは過負荷状態を示す重要な用途があることがわかった。これは何らかの特別な扱いを必要とする。具体的には、503 の受信は、DNS の SRV ルックアップの結果である次のエレメントにコンタクトを試みるトリガーになるべきである。また、503 レスポンスは特定の状況下でのみプロキシによってアップストリームにフォワードされる。

- RFC2543 は、サーバの UA 認証メカニズムについて、十分に明らかにしなかったが、定義した。それは削除された。その代わりに、RFC2617 の相互認証が認められた。

- UA は、最初の INVITE に対する ACK を受け取るまで、呼に対して BYE を送れない。潜在的な競合状態(race condition)に導くが、これは RFC2543 では許可されていた。

- UA またはプロキシは、リクエストに対する暫定レスポンス(provisional response)を受け取るまで、トランザクションに対して CANCEL を送れない。潜在的な競合状態(race condition)に導くが、これは RFC2543 では許可されていた。

- 登録における action パラメータは反対された。それはいかなる有用なサービスに対しても不適当で、アプリケーション処理がプロキシで適用されたときにコンフリクトを生じた。

- RFC2543 はマルチキャストに対して多数の特別ケースを持っていた。たとえば、特定のレスポンスが抑制されたり、タイマーが調整されたりするなどである。マルチキャストはより限定された役割を演じるようになった。そして、プロトコルオペレーションは、ユニキャストとは対照的にマルチキャストの用法には影響されない。その結果としての制限事項が記述された。

- ベーシック認証は完全に削除され、それの使用は禁止された。

- プロキシはもはや 6xx を受信したときにそれを即座にフォワードしない。その代わりに、ペンディング中の branch を即座に CANCEL する。これは、2xx が後に続く 6xx を UAC が受け取ること

になる潜在的な競合状態を回避する。この競合状態を除くすべての場合において、結果は同じになる。つまり、6xx がアップストリームにフォワードされる。

- RFC2543 はリクエストのマージの問題を解決しなかった。これは、リクエストがプロキシでフォークして、後にひとつのエレメントで再び一緒になったときに起こる。マージのハンドリングは UA でのみ行われ、手順は最初のリクエスト以外はすべて拒否するように定義された。

28.2 軽微な機能変更 (Minor Functional Changes)

- ユーザに対してオプションのコンテンツを提示するために、Alert-Info、Error-Info、および Call-Info ヘッダフィールドが追加された。

- Content-Language、Content-Disposition、および MIME-Version ヘッダフィールドが追加された。

- 両方のパーティがお互いに同時に re-INVITE を送る場合に対処するために「グレアハンドリング(glare handling)」メカニズムが追加された。これは新しい 491(Request Pending)エラーコードを使用する。

- 受け取れなかった電話またはメッセージに後ほど折り返すことをサポートするために、In-Reply-To および Reply-To ヘッダフィールドが追加された。

- 有効な SIP トランスポートとして、TLS および SCTP が追加された。

- 通話中に失敗をいつでもハンドリングするために、さまざまなメカニズムが記述されていた。それらが全般的に統一された。終了するために BYE が送られる。

- RFC2543 は、INVITE に対するレスポンスの再送を TCP 上で義務化していたがそれは 2xx に対してだけ必要であったことに気づいた。それは不十分なプロトコルのレイヤ化の結果であった。ここで定義された、さらに首尾一貫したトランザクションレイヤによって、それはもはや不要になった。INVITE に対する 2xx レスポンスだけが TCP 上で再送される。

- クライアントトランザクションマシンとサーバトランザクションマシンは、再送カウントではなくタイムアウトに基づいて駆動するようになった。これは、ステートマシーンが TCP と UDP に対して適切に規定されることを可能にする。

- Date ヘッダフィールドは、ユーザエージェントの日付を自動設定するための単純な手段を提供するために、REGISTER に対するレスポンスで使用される。

- レジストラサーバが、短すぎる期間の有効期限を持つ登録を拒否することを許可した。423 レスポンスコードと Min-Expires をこの目的のために定義した。

２９．規範的な参考文献 (Normative References)

[1] Handley, M. and V. Jacobson, "SDP: Session Description

Protocol", RFC 2327, April 1998.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Rosenberg, J. and H. Schulzrinne, "SIP: Locating SIP Servers", RFC 3263, June 2002.

[5] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[9] Vaha-Sipila, A., "URLs for Telephone Calls", RFC 2806, April 2000.

[10] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[11] Freed, F. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[12] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[13] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", RFC 3264, June 2002.

[14] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[15] Postel, J., "DoD Standard Transmission Control Protocol", RFC

761, January 1980.

[16] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[17] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[18] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

[19] Zimmerer, E., Peterson, J., Vemuri, A., Ong, L., Audet, F., Watson, M. and M. Zonoun, "MIME media types for ISUP and QSIG Objects", RFC 3204, December 2001.

[20] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[21] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[22] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[23] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[24] Ramsdell B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[25] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[26] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

３０．有益な参考文献 (Informative References)

[27] R. Pandya, "Emerging mobile and personal communication systems," IEEE Communications Magazine, Vol. 33, pp. 44--52, June 1995.

[28] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[29] Schulzrinne, H., Rao, R. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[30] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", RFC 3015, November 2000.

[31] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[32] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL scheme", RFC 2368, July 1998.

[33] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.

[34] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[35] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[36] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.

[37] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.

[38] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.

[39] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP: Digest Access Authentication", RFC 2069, January 1997.

[40] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Willis, D., Rosenberg, J., Summers, K. and H. Schulzrinne, "SIP Call

Flow Examples", Work in Progress.

[41] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," Journal of Internetworking: Research and Experience, Vol. 4, pp. 99--120, June 1993. ISI reprint series ISI/RS-93-359.

[42] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar. 1996.

[43] Floyd, S., "Congestion Control Principles", RFC 2914, September 2000.

付録１　タイマー値の表 (Table of Timer Values)

表 4 は、この仕様で使用されている各種タイマーの意味と初期値をまとめたものである。

タイマー値		節	意味
----------	T1	500ms default	17.1.1.1 節　RTT 予測値
T2	4s	17.1.2.2 節	非 INVITE リクエストおよび INVITE に対するレスポンスのための最大再送期間
T4	5s	17.1.2.2 節	メッセージがネットワーク上に残存する最大期間
Timer A	initially T1	17.1.1.2 節	UDP だけのための INVITE リクエストの再送間隔
Timer B	64*T1	17.1.1.2 節	INVITE トランザクションのタイムアウトタイマー
Timer C	> 3min	16.6 節の項目 11	プロキシの INVITE トランザクションタイムアウト
Timer D	> 32s for UDP 0s for TCP/SCTP	17.1.1.2 節	レスポンスの再送のための待ち時間
Timer E	initially T1	17.1.2.2 節	UDP だけのための非 INVITE リクエストの再送間隔
Timer F	64*T1	17.1.2.2 節	非 INVITE トランザクションのタイムアウトタイマー
Timer G	initially T1	17.2.1 節	INVITE に対するレスポンスの再送間隔
Timer H	64*T1	17.2.1 節	ACK 受信のための待ち時間
Timer I	T4 for UDP 0s for TCP/SCTP	17.2.1 節	ACK 再送のための待ち時間
Timer J	64*T1 for UDP 0s for TCP/SCTP	17.2.2 節	非 INVITE リクエストの再送のための待ち時間
Timer K	T4 for UDP 0s for TCP/SCTP	17.1.2.2 節	レスポンスの再送のための待ち時間

表 4: タイマーのまとめ

謝辞 (Acknowledgment)

コメントと提案をいただいた、IETF MMUSIC および SIP ワーキンググループのメンバーに感謝したい。詳細なコメントは、Ofir Arkin、Brian Bidulock、Jim Buller、Neil Deason、Dave Devanathan、Keith Drage、Bill Fenner、Cedric Fluckiger、Yaron Goland、John Hearty、Bernie Hoeneisen、Jo Hornsby、Phil Hoffer、Christian Huitema、Hisham Khartabil、Jean Jervis、Gadi Karmi、Peter Kjellerstedt、Anders Kristensen、Jonathan Lennox、Gethin Liddell、Allison Mankin、William Marshall、Rohan Mahy、Keith Moore、Vern Paxson、Bob Penfield、Moshe J. Sambol、Chip Sharp、Igor Slepchin、Eric Tremblay、Rick Workman らが提供してくれた。

Brian Rosen は編纂した BNF を提供してくれた。

Jean Mahoney はテクニカルライティングの支援をしてくれた。

この作業は、特に参考文献[41,42]を基にしている。

著者の連絡先 (Authors' Addresses)

著者の連絡先は、エディター、ライター、RFC2543 のオリジナルの著者の順で、アルファベット順にリストされている。リストされているすべての著者は、このドキュメントに大量の文書を積極的に寄稿してくれた。

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Ave
East Hanover, NJ 07936
USA

EMail: jdrosen@dynamicsoft.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA

EMail: schulzrinne@cs.columbia.edu

Gonzalo Camarillo
Ericsson
Advanced Signalling Research Lab.
FIN-02420 Jorvas
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Alan Johnston
WorldCom
100 South 4th Street
St. Louis, MO 63102
USA

EMail: alan.johnston@wcom.com

Jon Peterson
NeuStar, Inc
1800 Sutter Street, Suite 570
Concord, CA 94520
USA

EMail: jon.peterson@neustar.com

Robert Sparks
dynamicsoft, Inc.
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
USA

EMail: rsparks@dynamicsoft.com

Mark Handley
International Computer Science Institute
1947 Center St, Suite 600
Berkeley, CA 94704
USA

EMail: mjh@icir.org

Eve Schooler
AT&T Labs-Research
75 Willow Road
Menlo Park, CA 94025
USA

EMail: schooler@research.att.com

完全な著作権表記 (Full Copyright Statement)

謝辞

RFC 編集者の職務のための資金は、現在、インターネットソサエティによって提供されている。

付属資料　1.　SIP RFC 3261 対訳表　（単語/熟語）

原文	和訳時の統一表記	備考
[A]		
address-of-record	address-of-record(AOR)	rfc3261上,AOR,address-of-record の２表記有り。
Alert	Alert	パラメータ名のため原文のまま
alternative service	代替サービス	
answerer	アンサー側(answerer)	
atomic	非分割(atomic)	
atomically	非分割的(atomically)	
attempt(名詞)	試み、試行	
attempt(動詞)	試みる、試す、試行する	
authentication	認証	
authorization	認可	Authorization XX は原文のまま
[B]		
Back-toBack User agent	バック・トゥ・バックユーザーエージェント	
"basic" and "digest"	ベーシックおよびダイジェスト	
behavior	動作	
branch	branch	
[C]		
cache	キャッシュ	
cached	キャッシュされた	
call	呼、通話、コール	
call flow	コールフロー	
call forwarding	着信転送	
call handling fields	呼ハンドリングフィールド	
call identifier	呼識別子	
call leg	コールレグ	
call processing	呼処理	
callee	着信者	
called user agent	ユーザーエージェントと呼ばれる	
called party	着信側パーティ	
caller	発信者	
calling party	発信側パーティ	
call stateful	コールステートフル	

case-insensitive	大文字、小文字を区別しない	
certificate	証明書	
challenge	チャレンジ	
class	クラス	
client	クライアント	
conference	カンファレンス	
configuration	設定	connection　コネクション、接続
	TCPコネクション	
consult	検索する	
contact(動詞)	コンタクトする	Contactは原文のまま
contents characteristics	コンテンツ特性	
credentials	信用証明書	

[D]

definitive response	確定レスポンス(definitive response)	
delimiter	区切文字	
Destination Address	送信先アドレス　／ デスティネーションアドレス	
Dialog	ダイアログ	
digital signature	デジタル署名	
display name	ディスプレイネーム	
downstream	ダウンストリーム	

[E]

element	エレメント	
encryption	暗号化	
entity	エンティティ	
equivalent	等価	as equivalent toは、「等価なものとして」

[F]

feature	機能	
Fig. x	図x	
Final response	最終レスポンス(Final response)	
firewall	ファイアウォール	
formal	正式な	
format	フォーマット	
forward	フォワード	
forwarding loops	ループ転送	
future	以降の、将来の	future request, future callなどのようにプロトコル処理で出てくる

		る場合は「以降の」を用い、かなり先のサポートという表現では「将来の」を使う。

[G]

global	グローバル	

[H]

handshake	ハンドシェイク	
hash	ハッシュ（値）	
header	ヘッダ	
header field	ヘッダフィールド	
hop-by-hop	ホップバイホップ	

[I]

identify	識別する	
identity	アイデンティティ	
implementation	実装	
informational resonse	通知レスポンス(informational response)	
initiate	開始する	
initiator	起動者	
internet	インターネット	インターネットドラフトなど
Internet telephone calls	インターネット電話呼	
invite	招待する	
invited user	招待されたユーザ	
invitee	招待された側	
invitation	招待	

[J]

[K]

knowledge	知識	

[L]

layer	レイヤ	
legitimately	正当に	
line	行	xxx-lineの時は原文表記
linear white space	linear white space	
listen	待ち受けする	listening: 待ち受け中
location	ロケーション	

location server	ロケーションサーバ	
location service	ロケーションサービス	
loose routers	ルースルータ	

[M]

magic cookie	マジッククッキー	
malformed	不正な形式	
map（動詞）	マップする	
mapping（名詞）	マッピング	
mapping（動詞）	マッピングする	
media	メディア	media-xxxの時は原文表記
message body	メッセージボディ	message-bodyの時はmessage-body
method	メソッド	原則としてmethod-xxxの時は原文表記
mid-call features	コール内機能(mid-call features)	
multicast	マルチキャスト	

[N]

null	ヌル	
next downstream element	ネクストダウンストリームエレメント	
Next-Hop,next-hop,next hop	ネクストホップ	

[O]

offensive content	不適切なコンテンツ	
offerer	オファー側(offerer)	
on-going	進行中	
operation	オペレーション	
originator	発信元	
outbound proxy	アウトバンドプロキシ	
outcome	結果	
override	オーバーライドする	
overriding mechanism	オーバライドメカニズム	

[P]

parallel	並行して	
parallel search	パラレルサーチ	
party	パーティ	
pending	保留/未処理	Request Pendingは原文のまま
personal mobility	パーソナルモビリティ	

preloaded	プリロード	
prose	文	
proxy (proxies) (名詞)	プロキシ	複数形も単数形のproxyとする
proxy (動詞)	プロキシする	
Proxy xx	Proxy xx	
Processing (名詞)	処理	
Processing (動詞)	処理を行う／処理する	
provisional response	暫定レスポンス(provisional response)	

[Q]

query(名詞)	クエリー	queryingも同様
query(動詞)	問い合わせる	
queue(動詞)	キューイングする	

[R]

realm	realm	
reason phrase	リーズンフレーズ	
recipient	受信者	
redirect(動詞)	リダイレクトする	
redirect server	リダイレクトサーバ	
redirection	リダイレクション	
REGISTER (名詞)	REGISTER	
register(動詞)	登録する	
registration	登録	
registrar	レジストラサーバ	
registry	レジストリ	
regular transaction	レギュラートランザクション	
reject	拒否	拒絶も可だが拒否が基本
Request	リクエスト	
resource	リソース	
Response	レスポンス	
return	返送する	
ringback tone	呼び出し音	
rogue inter mediaries	悪意のある中継媒体	
route(名詞)	経路（ルート）	
route(動詞)	ルーティングする	
route set	ルートセット	

[S]

scheme	スキーム	
Section xx	ｘx節	
security mechanism	セキュリティメカニズム	
semantics	意味合い(semantics)	原文併記
separator	セパレータ	
serially	順次に	
session	セッション	
session description	セッション記述	
server	サーバ	
signature	署名	
SIP call	SIPコール	
spiral（名詞）	スパイラル	
spiral（動詞）	スパイラルする	
state	状態	
Stateless xx	ステートレス xx	
state diagram	状態遷移図	
state machine	ステートマシーン	
stateful proxy	ステートフルプロキシ	
status xx	ステータスｘｘ	ステータスコード、等
strict-routing	ストリクトルーティング	
success response	成功レスポンス(success response)	
syntax	構文	

[T]

tag	タグ	
terminate	終了（基本的に）	呼（ｾｯｼｮﾝ）を終了する、等
third-party xxx	第三者(third-party)のxxx	原文併記
threat	危機(threat)	危機(threat)モデル　等
time-of-day	時間帯別	
time-to-live	存続時間	
token	トークン	
topmost（もしくはtop-most）	先頭の	
transaction	トランザクション	
transport	トランスポート	

[U]

unique	一意の	
update(d)	アップデートする	
upstream	アップストリーム	
URL-encoded	URLエンコードされた	

user	ユーザ	
user agent (client/server)	ユーザエージェント	
[V]		
version number	バージョン番号	
[W]		
white space	ホワイトスペース	
[X]		
Xx server	Xx サーバ	プロキシサーバ　等
[Y]		
[Z]		
[他]		
(2**32)-1	$2^{32}-1$	

付属資料　2.　SIP RFC 3261 対訳表　（文章）

該当節	原文	和訳例
	…UAC proxies all requests…	UACが全てのリクエストをプロキシする
	…The string is case-insensitive, but implementations MUST use upper-case	文字列において大文字、小文字の区別はないが、実装時には大文字を使わなくてはならない。
	Will pick up the phone	受話器をとる
	connected	接続される
22.1節	canonical root URL	標準化したルートURL
20.1節	angle bracket, < >	山括弧
25.1節	parentheses, ( )	小括弧
	square bracket, [ ]	大括弧（角括弧）
	augmented Backus-Naur Form	拡張バッカス-ナウア記法
21.4.22 節	overlapped dialing	重複ダイアリング(overlap dialing)

付属資料　3.　SIP RFC 3261 対訳表　（RFC2119 関連）

要求レベルを表す語句(RFC2119)については、以下の訳文を当てることとし、その後に括弧書きで原文の語を追記することとする。

MUST	(～し)なくてはならない[MUST]
MUST NOT	(～し)てはいけない[MUST NOT]
REQIRED	(～)が要求される[REQIRED]
SHALL	(～し)なくてはならない[SHALL]
SHALL NOT	(～し)てはいけない[SHALL NOT]
SHOULD	(～する)べきである[SHOULD]
SHOULD NOT	(～する)べきではない[SHOULD NOT]
RECOMMENDED	(～)が推奨される[RECOMMENDED]
MAY	(～し)てもよい[MAY]
OPTIONAL	(～)が選択可能である[OPTIONAL]