2
O Protocolo SIP

2.1.
Introdução Histórica

O protocolo SIP teve suas origens em 1996 como um componente do conjunto de ferramentas e protocolos da Mbone, ou *Multicast backbone* [44]. A Mbone era uma rede *multicast* experimental que rodava em cima da Internet. Naquela época, suas aplicações incluíam distribuição de conteúdo multimídia, como seminários, palestras e reuniões do IETF. O SIP foi originalmente desenvolvido pelo grupo de trabalho da IETF MMUSIC (*Multi-Party Multimedia Session Control*) e sua versão 1.0 foi lançada como *Internet Draft* em 1997. Mudanças significativas foram feitas no protocolo resultando na segunda versão, submetida como *Internet Draft* em 1998. O protocolo recebeu aprovação em março de 1999 e foi publicado como RFC 2543 em abril de 1999. Um novo *Internet Draft* contendo correções de *bugs* e esclarecimentos relacionados ao SIP foi publicado em julho de 2000, como RFC 2543 "bis". Este documento evoluiu e acabou resultando na publicação da RFC 3261, que substitui a RFC 2543.

A popularidade do SIP levou a criação de vários grupos de trabalho dentro do IETF como o SIPPING (*Session Initiation Protocol Investigation*) que é responsável por investigar as aplicações do SIP, desenvolver requisitos para as extensões do SIP, e publicar documentos BCP (*Best Current Practice*) sobre o uso do SIP, e o SIMPLE (*SIP for Instant Messaging and Presence Leveraging*), que é responsável pela padronização de aplicações do SIP para presença e mensagens instantâneas. Os grupos PINT (*PSTN and Internet Internetworking*) e SPIRITS (*Service in the PSTN/IN requesting Internet Services*) também publicam documentos que tem relação com o SIP.

O SIP foi desenvolvido com alguns requisitos em mente. O primeiro é a escalabilidade. O protocolo teria que funcionar independentemente da localização física do usuário, bastando que o usuário estivesse conectado a Internet. Usuários poderiam ser convidados para muitas sessões, então o

protocolo deveria ser escalável em ambas as direções. O segundo requisito é o reuso de componentes. Em vez de inventar novas ferramentas, aquelas já desenvolvidas pelo IETF deveriam ser reutilizadas. Ferramentas como MIME [45], URLs [46] e SDP poderiam ser aproveitadas. Isto resultou em um protocolo que integra-se bem com outras aplicações IP (como *web* e *e-mail*). Interoperabilidade era outro requisito perseguido. Interoperabilidade está no coração dos processos e desenvolvimentos do IETF. Para atender este requisito, utilizou-se do princípio KISS (*Keep It Simple Stupid*) para manter o protocolo o mais simples possível.

2.2. Aspectos Gerais

O SIP é um protocolo de controle da camada de aplicação que pode estabelecer, modificar e terminar sessões multimídia como chamadas telefônicas através da Internet.

Resumidamente, o SIP tem as seguintes características:

- É baseado em texto. A estrutura das mensagens do SIP é muito mais fácil de se extender a novas aplicações do que protocolos equivalentes, como o H.323 que usa o padrão de codificação ASN.1 em vez de texto. Isso possibilita fácil implementação em linguagens de programação orientadas a objetos como Java e Perl [47], possibilitando fácil depuração de problemas e fazendo do SIP um protocolo simples, flexível e extensível.
- Envolve menos sinalização. O SIP foi desenvolvido para atender somente aos requisitos básicos de um protocolo de sinalização de chamadas (criar, modificar e terminar sessões), para que a sinalização seja o mais simples possível. O rápido estabelecimento de chamadas é crucial para uma alta qualidade de serviço (QoS). Além disso, um grande número de parâmetros usados na negociação e estabelecimento de *streams* de mídia entre participantes de uma chamada é encapsulado no corpo da mensagem do SIP através do SDP. Isso também aumenta a velocidade no estabelecimento da chamada.
- Separação entre sinalização e mídia. Com o SIP, os caminhos entre sinalização e mídia são totalmente independentes. A

sinalização e mídia podem passar por diferentes rotas através de conjuntos independentes de equipamentos em diferentes redes.

- Independe do protocolo de transporte utilizado. Embora o SIP tenha sido desenvolvido para ser independente do protocolo da camada de transporte, tipicamente ele roda sobre UDP em vez de TCP. O estabelecimento de conexões TCP e suas rotinas de confirmações introduzem atrasos, que são cômodos e devem ser evitados em transmissões de voz. Adotando o UDP, entretanto, o tempo das mensagens e suas retransmissões devem ser controlados pela camada de aplicação.
- Possibilidade de *forking* (procura paralela). Isso possibilita que várias entidades sejam associadas a um único endereço, afim de que todas ou uma seleção delas sejam contatadas simultaneamente ou seqüencialmente, dependendo da política local. A primeira entidade que atender recebe a chamada.

O SIP segue cinco etapas para estabelecimento e término de uma chamada:

- Localização do usuário – Descobrir a localização do usuário onde quer que ele esteja.
- Disponibilidade do usuário – Determinação da disponibilidade do usuário chamado em participar de uma chamada.
- Capacidades do usuário – Negociação e determinação dos formatos de mídia a serem utilizados entre o usuário chamador e usuário chamado.
- Estabelecimento da sessão – O diálogo é estabelecido e os *streams* de mídia são transportados.
- Gerenciamento da sessão – Incluindo transferência e término de sessões, modificações de parâmetros da sessão e adição de serviços.

2.2.1.
O Estabelecimento de uma Chamada

Será apresentado nesta seção um exemplo simples de estabelecimento de uma chamada através do protocolo SIP de modo a facilitar o entendimento das próximas seções.

Aline liga para Bob que trabalham na mesma companhia e são servidos pelo mesmo servidor *proxy*. Já que Aline não liga regularmente para Bob, seu *SIP phone* não possui o endereço IP do *SIP phone* de Bob. Assim, a sinalização através de SIP passará pelo servidor *proxy*. Aline disca o número privado de Bob, que é 3114-1031. Seu *SIP phone* converte o número privado em um SIP URI (*Uniform Resource Indicator*) [48] (sip: 2131141031@nice.com) e envia um INVITE para o servidor *proxy*. A Figura 1 mostra a troca de mensagens para este exemplo.

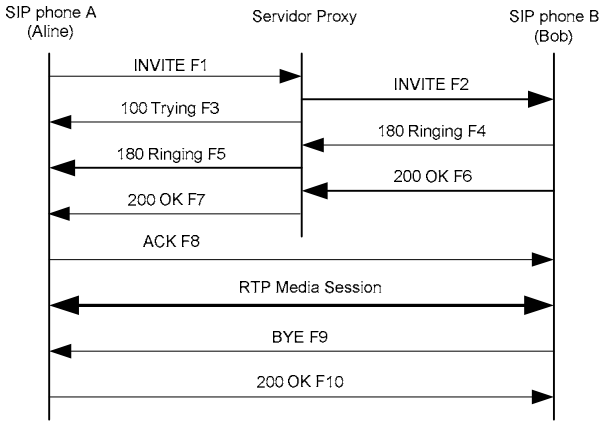

Figura 1 – Estabelecimento de uma chamada

O corpo da mensagem INVITE contém a descrição do tipo de sessão que o *SIP phone* de Aline deseja estabelecer, contendo o tipo de *codec* que será usado, o endereço IP e porta em que deseja receber os pacotes de mídia, etc.

Quando o *proxy* recebe a mensagem INVITE, ele envia uma resposta 100 Trying de volta para o *SIP phone* de Aline, indicando que o *proxy* está tentando rotear a mensagem INVITE para o *SIP phone* de Bob. O servidor *proxy* adiciona um campo de cabeçalho com seu próprio endereço IP na mensagem INVITE (campo *Via*) e encaminha-a ao *SIP phone* de Bob.

Quando o *SIP phone* de Bob recebe o INVITE, ele toca, e então Bob pode decidir se aceita ou não a chamada. Desde que o nome de Aline esteja no cabeçalho *To*, o *SIP phone* de Bob pode mostrar o nome de Aline. O *SIP phone* de Bob envia uma resposta 180 Ringing através do *proxy* de volta para o *SIP*

phone de Aline. O *proxy* usa o cabeçalho *Via* para determinar para onde enviar a resposta, e remove seu próprio endereço do topo. Quando o *SIP phone* de Aline recebe a resposta 180 Ringing, o tom de “chamando” é escutado por Aline.

Quando Bob atende a chamada, o seu *SIP phone* envia uma resposta 200 OK para indicar que ele atendeu a chamada. O corpo da mensagem “200 OK” contém a descrição de mídia do tipo de sessão que o *SIP phone* de Bob pode estabelecer para esta chamada. Assim, há uma troca nos dois sentidos de mensagens contendo o tipo de mídia, negociando as capacidades a serem utilizadas na chamada. O *SIP phone* de Aline envia um ACK diretamente ao *SIP phone* de Bob (sem passar pelo servidor *proxy*), e Aline pode falar com Bob através de uma sessão de mídia RTP (*Real Time Protocol*). Note que os pacotes de voz são roteados diretamente entre os dois *SIP phones*, e seus cabeçalhos não possuem qualquer informação sobre mensagens SIP ou servidor *proxy* que estabeleceram a sessão de mídia RTP.

Quando Bob desliga a chamada, seu *SIP phone* envia uma mensagem BYE diretamente ao *SIP phone* de Aline. O *SIP phone* de Aline responde com um 200 OK, que termina a chamada, incluindo a sessão de mídia RTP.

Existem situações em que o estabelecimento da chamada passa por vários servidores *proxy* ou *redirect*, que serão apresentados neste capítulo.

2.3. Entidades SIP

O SIP define duas classes básicas de entidades de rede: clientes e servidores. Um cliente é qualquer elemento de rede, como um PC com um microfone ou um *SIP phone*, que envia requisições SIP (*SIP requests*) e recebe respostas SIP (*SIP responses*). Um servidor é um elemento de rede que recebe requisições e envia respostas, podendo aceitar, rejeitar ou redirecionar a requisição. Logo, o SIP é um protocolo cliente-servidor. Note que os clientes e servidores são entidades lógicas. Seus papéis duram somente durante uma transação (consultar seção 2.4.6), o que significa que um cliente também pode ser um servidor.

Existem quatro tipos de servidores diferentes: *proxies*, *user agent servers* (UAS), *redirect* e *registrars*.

Note, que a diferença desses servidores SIP são somente lógicas, e não físicas. Tipicamente, o *registrar* é combinado com o *proxy* ou servidor *redirect*. Assim, o *proxy* ou servidor *redirect* pode fazer uso imediato da informação de

localização quando recebe requisições para encaminhá-las aos usuários registrados, melhorando assim o tempo de processamento, que depende da implementação e configuração.

2.3.1. User Agents (UAs)

End points (ou pontos finais) que usam o SIP para estabelecer e negociar sessões são chamados de *user agents* (UA). Geralmente, os *user agents* são chamados de *User Agent Server* (UAS) e *User Agent Client* (UAC). No entanto, UAS e UAC são entidades lógicas apenas, sendo que cada UA possui um UAC e UAS. UAC é a parte do UA responsável por enviar requisições e receber respostas. UAS é a parte do UA responsável por receber requisições e enviar respostas.

Como o UA possui ambos UAC e UAS, é comum dizer que ora ele atua como um UAC ora como um UAS. De fato, o UA do usuário chamador se comporta como um UAC quando ele envia uma requisição INVITE e recebe respostas para esta requisição. O UA do usuário chamado se comporta como um UAS quando recebe uma requisição INVITE e envia respostas.

Essa situação se inverte quando o usuário chamado decide terminar a sessão enviando uma requisição BYE. Neste caso, o UA do usuário chamado passa a ser o UAC e o UA do usuário chamador passa a ser o UAS. Esta situação é representada na Figura 2.

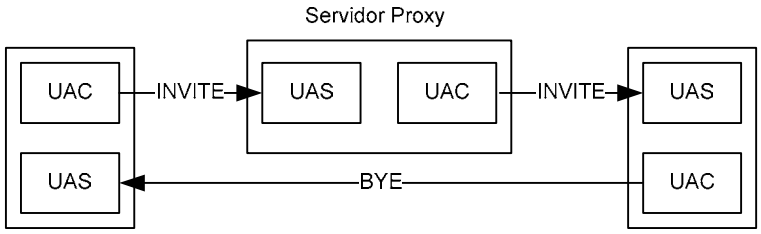

Figura 2 – UAC e UAS, entidades lógicas

2.3.2. Servidor Proxy

A RFC 3261 define servidores *proxies* como elementos responsáveis por rotear requisições SIP para UAS e respostas SIP para UAC. Uma requisição pode passar por diversos *proxies* até chegar ao UAS. Cada um deles tomará decisões sobre o roteamento, modificando a requisição antes de encaminhá-la

para o próximo elemento. As respostas serão roteadas pelo mesmo conjunto de *proxies* usados para encaminhar a requisição na ordem inversa.

A tarefa mais importante do servidor *proxy* é rotear as requisições “para mais perto” (*closer*) do usuário chamado.

Os servidores *proxy* podem ser vistos como roteadores da camada de aplicação SIP sendo responsáveis por encaminhar requisições e respostas SIP. No entanto, os *SIP proxies* empregam uma lógica de roteamento que tipicamente é mais sofisticada do que encaminhar mensagens automaticamente baseando-se em uma tabela de roteamento. De acordo com a RFC 3261, os *proxies* são responsáveis por validar as requisições, autenticar usuários, fazer uso de *forking* (procura paralela), resolver endereços, cancelar chamadas ainda não completadas, adicionar o campo *Record-Route* no cabeçalho de mensagens, e detectar *loops*. A versatilidade dos servidores *proxy* permite, por exemplo, que requisições sejam encaminhadas para apenas usuários autenticados que não possuem nenhum tipo de débito com o provedor de serviço.

O servidor *proxy* é projetado para que na maioria das vezes seja transparente para UAs. Eles podem modificar as mensagens apenas de modo específico e limitado. Por exemplo, o *proxy* não pode modificar o conteúdo do SDP de uma mensagem INVITE. Fora algumas exceções, os *proxies* não podem gerar requisições por iniciativa própria.

A especificação do SIP define dois tipos de *proxies*: *stateful* e *stateless proxy*.

2.3.2.1. Stateless Proxy

Um *stateless proxy* é um simples encaminhador de mensagens, como é descrito na especificação do SIP. Quando recebe uma requisição, o *stateless proxy* processa a requisição da mesma forma que o *stateful proxy*, mas o *stateless proxy* encaminha a mensagem sem guardar qualquer informação da transação. Eles encaminham as mensagens independentemente das outras. Ou seja, quando uma mensagem é encaminhada, o *proxy* simplesmente “esquece” de que uma vez tratou daquela mensagem.

Stateless proxies são mais simples, e por isso, mais rápidos que os *stateful proxies*. No entanto, podemos listar algumas desvantagens:

- Um *stateless proxy* não pode associar respostas às requisições encaminhadas, porque ele não possui conhecimento das requisições que ele encaminhou. Assim, o *proxy* não tem como saber se a transação foi bem sucedida ou não. Um *stateless proxy* não pode gerar uma resposta provisória 100 Trying ou qualquer outra mensagem provisória.
- Um *stateless proxy* não pode associar retransmissões de requisições e respostas com as mensagens enviadas anteriormente. Desta forma, as mensagens são processadas da mesma forma que a mensagem original recebida, isto é, o *proxy* não tem a capacidade de reconhecer que a mensagem é uma mensagem retransmitida.
- Se uma mensagem for perdida, o *proxy* não a retransmitirá. Retransmissões são responsabilidade de *stateful proxies* ou UAs.
- *Stateless proxies* não possuem a funcionalidade de *forking*.

Por causa da sua maior capacidade de processar mensagens, *stateless proxies* são comumente usados no *core* da rede de provedores de serviços ou operadoras de telecomunicações para auxiliar no roteamento das mensagens SIP pela rede.

2.3.2.2. Stateful Proxy

Stateful proxies são mais complexos. Quando recebem uma requisição, eles criam um estado e mantém este estado até que a transação termine. Isto é, eles processam transações em vez de mensagens individuais. Algumas transações, especialmente aquelas criadas pelo INVITE, podem durar muito tempo (até o usuário chamado aceitar ou rejeitar a chamada).

O comportamento do *stateful proxy* é modelado em termos de transações de clientes e servidores. Transações de servidores são responsáveis por receber requisições e retornar respostas, e transações de clientes são responsáveis por enviar requisições e receber respostas. Uma requisição é processada por uma transação de servidor e então encaminhada adiante para uma ou mais transações de clientes (pode haver mais de uma transação de cliente no caso de *forking*, por exemplo). Uma resposta é recebida pela transação de cliente adequada e então encaminhada de volta a transação de servidor. Associar

transações de clientes e servidores e gerenciar o estado completo de cada requisição é responsabilidade do *proxy core* (Figura 3). O *proxy core* determina para onde rotear a requisição, escolhendo uma ou mais transações de clientes. Ele também coleta as respostas de diferentes transações de clientes e escolhe a(s) resposta(s) que será(ão) enviada(s) de volta para a transação de servidor.

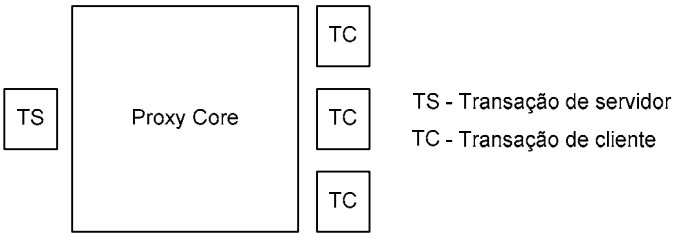

Figura 3 – Servidor stateful proxy

A capacidade de associar mensagens SIP a transações dá ao *stateful proxy* algumas características interessantes. Eles podem executar *forking*, o que significa que ao receber uma requisição, duas ou mais mensagens serão encaminhadas. São capazes de identificar retransmissões de mensagens e encaminhá-las somente em situações onde a retransmissão é necessária, enquanto o *stateless proxy* não é capaz de identificar retransmissões, encaminhando toda mensagem que passa por ele. Um *stateful proxy* também pode gerar retransmissões em caso de perda de mensagens. Além disso, podem processar localmente requisições CANCEL encaminhadas a ele e gerar requisições CANCEL quando necessário.

Possuem também a capacidade de utilizar métodos mais avançados para procurar um usuário. Por exemplo, quando um INVITE é enviado ao telefone de um usuário no seu escritório de trabalho e quando ele não atende, a chamada é redirecionada para o seu telefone celular. *Stateless proxies* não podem fazer isso, porque eles não tem conhecimento de como a transação (chamada ao telefone do escritório) terminou.

A maior parte dos *SIP proxies* de hoje são *stateful* porque suas configurações são geralmente mais complexas. Eles têm a capacidade de administrar contas de usuários, *forking*, alguma forma de ajuda à transposição de NAT (*Network Address Translation*) e tudo isso requer um *stateful proxy*.

A capacidade de “saber” o estado da transação e do histórico de mensagens, dão ao *stateful proxy* alguns desvantagens [49]:

- Maior consumo de memória – O *stateful proxy* retém mais memória por mensagem processada que o *stateless proxy*, e por uma duração de tempo maior. Isso tem um impacto negativo na capacidade máxima do *proxy* e limita a quantidade de chamadas/transações concorrentes que ele pode tratar. Algumas otimizações no código, que são específicas do SIP, podem compensar isso e baixar o consumo de memória para níveis requeridos por *proxies* de alta capacidade.
- *Throughput* – Um *stateful proxy* gasta mais ciclos de CPU por mensagem processada – mapeando mensagens em transações, gerenciando o estado das transações, processando os *timers* das transações e associando transações de cliente e servidor. Esse processamento extra reduz a capacidade do *proxy* em termos de performance (número máximo de mensagens processadas por segundo). Por esses motivos, o desenvolvimento de *stateful proxies* de alta performance se mostrou não ser trivial, e requer otimizações especiais no desenvolvimento do *proxy*, como a possibilidade de customização através da flexibilidade de configuração.
- Complexidade de implementação – Um *stateful proxy* faz mais do que apenas rotear mensagens. Algumas lógicas precisam ser empregadas de modo que se possa lidar com ações como *forking* paralelo (por exemplo, escolhendo a melhor resposta), CANCEL, recursão para mensagens de classe 3xx (consultar seção 2.4.4), e tratar mensagens ACK para respostas de classe 2xx (consultar seção 2.4.4).

2.3.2.2.1. Comportamento do Stateful Proxy

Para todas as novas requisições, um elemento que deseja atuar como *proxy* deve:

- Validar a requisição;
- Pré-processar as informações de roteamento;
- Determinar o(s) destino(s) para a requisição;
- Rotear a requisição para cada destino;
- Processar todas as respostas.

Antes de receber a requisição, um *proxy* deve validar a requisição para ter certeza de que pode dar prosseguimento no processamento da mensagem. A mensagem precisa passar pelos seguintes testes de validação:

- Verificação de sintaxe – A requisição precisar ser suficientemente bem formada para que possa ser tratada pelo *proxy*. No entanto, este teste só é aplicado nos campos específicos da mensagem que o servidor *proxy* precisa processar. Todos as outras partes não devem ser verificadas, já que o SIP é projetado para ser um protocolo extensível (um proxy não deve rejeitar mensagens cujos métodos/cabeçalhos sejam desconhecidos para ele).
- Verificação do formato da URI – O formato do URI (por exemplo, "sip:" em sip:exemplo.com) precisa ser da forma que o *proxy* entenda e saiba como rotear. Se não, o *proxy* retornará a mensagem 416 (*Unsupported URI Scheme*).
- Verificação do campo *Max-Forwards* do cabeçalho – *Max-Forwards* é um campo de cabeçalho da mensagem que indica quantos saltos (*hops*) a mensagem pode atravessar. Cada *proxy* que trata a mensagem decrementa esse número de uma unidade (similar ao campo TTL de alguns protocolos). Se a mensagem não possuir este campo de cabeçalho ou possuir um valor de *Max-Forwards* acima de zero, a mensagem é encaminhada. Se a mensagem possuir um valor de *Max-Forwards* igual à zero, o *proxy* não encaminhará a mensagem e deverá retornar uma mensagem 483 (*Too many hops*). Esse mecanismo previne que a mensagem entre em *loop* em um conjunto de *proxies*.
- Detecção de *Loop* (Opcional) – Um *proxy* pode verificar a existência de *loops* através da execução de um algorítmo no campo *Via* do cabeçalho contido na mensagem. O *proxy* verifica se já não tratou da mensagem "no passado". Se ele já tratou da mensagem, ele verifica se a mensagem contém valores diferentes em campos de cabeçalho que influenciam as decisões de roteamento (como o *Request-URI*, *From* e *To*). Se o *proxy* identifica uma condição de *loop*, ele rejeita a mensagem enviando uma mensagem 482 (*Loop Detected*). Este teste é opcional.

- Verificação do campo *Proxy-Require* do cabeçalho – O cliente pode indicar que certas extensões do SIP são necessárias que o *proxy* suporte para que a mensagem possa ser processada corretamente. O *proxy* deve ler este cabeçalho e verificar se suporta todas as extensões listadas. Se não suportar, o *proxy* deverá retornar uma mensagem 420 (*Bad Extension*) indicando qual parâmetro listado no campo do cabeçalho *Proxy-Require* não é suportado através do campo *Unsupported* do cabeçalho.
- Autenticação – Se a autenticação no *proxy* for necessária para o originador da mensagem, ele deve fornecer as credenciais necessárias para autenticar o usuário. Se a mensagem não incluir tais credenciais ou as credenciais falharem na autenticação do usuário, o *proxy* deve retornar uma mensagem 407 contendo um desafio. O procedimento de autenticação será explicado na seção 2.3.2.2.3.

Uma vez que o *proxy* tenha validado a requisição e decidido roteá-la, ele deve determinar o(s) destino(s) que a mensagem deve ser roteada antes de enviá-la. O *proxy* utiliza dois tipos de resolução de endereços, em seqüência:

- Determinação do conjunto de endereços de destino – O *proxy* resolve o endereço de destino (*Request-URI*) da requisição SIP em um conjunto de endereços SIP que são mapeados de alguma forma.
- Resolução de DNS – O *proxy* resolve cada um dos endereços de destino em um endereço de transporte da forma: {*transport_protocol, IP address, port*}

O primeiro processo na resolução de endereço, conhecido como obter o conjunto de endereços de destino (*target-set*) na especificação do SIP, resulta em produzir um conjunto de endereços SIP. Este estágio mapeia endereços SIP em endereços SIP, ou seja, um endereço sip:userA@domain.com pode ser resolvido para sip:userA@cliente1.domain.com.

O conjunto de endereços de destino pode ser obtido de uma das duas formas listadas abaixo:

- Conjunto de endereços de destino pré-definido – Este é o caso mais simples, onde o endereço de destino da requisição (*Request-URI*) é tal que o *proxy* pode rotear automaticamente para o endereço de destino sem precisar resolver para outros endereços. Um destes casos é quando o *Request-URI* está no domínio no qual o *proxy* não é responsável.
- Conjunto de endereços de destino determinado pelo *proxy* – Se o conjunto de endereços de destino não for pré-definido, significa que o *proxy* é responsável pelo domínio do *Request-URI* e deverá aplicar qualquer mecanismo necessário para determinar para onde enviar a requisição. Algumas opções são: acessar um serviço de localização criado por um servidor *registrar*, consultar um banco de dados, consultar um servidor de presença ou utilizar outros protocolos.

Antes de enviar a mensagem, o *proxy* precisa resolver o(s) endereço(s) SIP para endereço(s) de transporte. *Proxies*, como outras entidades SIP, usam o mecanismo de DNS para resolução de endereços descrito na RFC 3263 [50]. Essencialmente, este é um algoritmo que seletivamente usa interrogações do tipo SRV, NAPTR, A e AAAA DNS para mapear um dado endereço SIP na forma de endereço de transporte {*transport_protocol, IP address, port*}.

2.3.2.2.2. Forking

Depois de processar a requisição e montar a lista de endereços de destino, o *proxy* pode escolher encaminhá-la para múltiplos endereços. Esse processo é chamado de *forking* ou procura paralela e o *proxy* que suporta este recurso é chamado de *forking proxy.* O *forking* permite a implementação de funcionalidades como procurar simultaneamente por um usuário em múltiplos UAs (em casa e no escritório, por exemplo), procurar o primeiro UA disponível em um call center, e *forward-on-busy* (encaminhar para outro UA em caso de ocupado).

Um *proxy* pode escolher como usar o *forking* de várias maneiras:

- *Forking* paralelo – O *proxy* encaminha cópias da requisição para múltiplos destinos simultaneamente.

- *Forking* seqüencial – O *proxy* encaminha a requisição para um endereço de destino de cada vez, esperando por uma resposta final (de falha) antes de encaminhar para o próximo endereço.
- *Forking* misto – O *proxy* pode escolher encaminhar as requisições para alguns endereços de modo paralelo enquanto para outros de modo seqüencial.

A Figura 4 mostra o *message flow* de um *forking* paralelo.

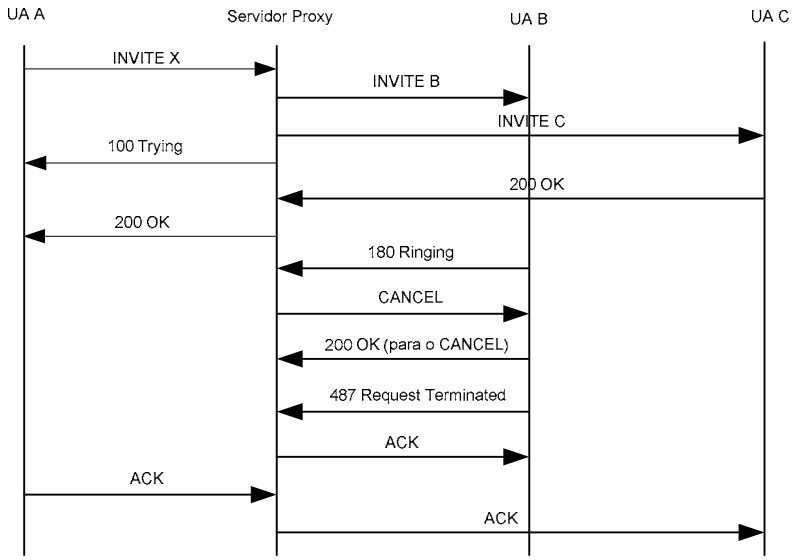

Figura 4 – Message flow de forking paralelo

A funcionalidade de *forking* paralelo aumenta a complexidade do *proxy*, já que o *proxy* tenta contatar o usuário em vários destinos simultaneamente, fazendo com que o ele trate de múltiplas transações concorrentes de clientes e possivelmente colete múltiplas respostas finais. O *proxy* precisa escolher qual a melhor resposta final e encaminhá-la ao usuário chamador. Ao receber uma resposta de classe 2xx (*Success*) ou 6xx (*Global Failure*), o *proxy* precisa cancelar as requisições pendentes (usando a requisição CANCEL) e encaminhar a resposta final ao usuário chamador.

A Figura 5 mostra o *message flow* de um *forking* seqüencial.

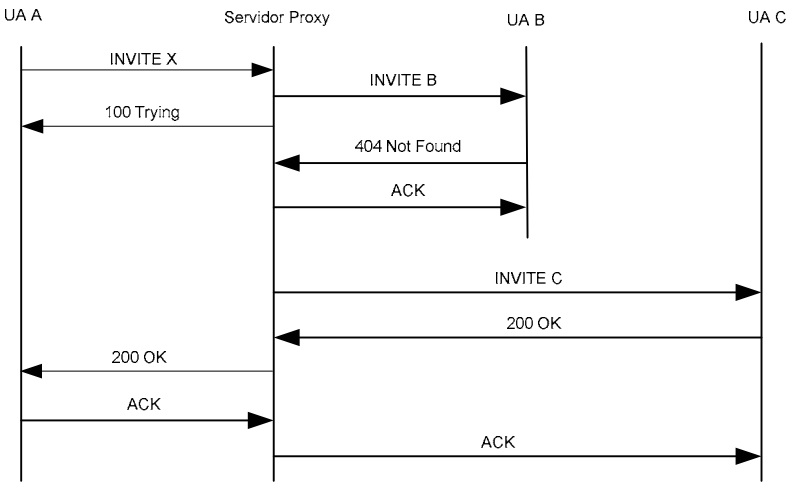

Figura 5 – Message flow de forking seqüencial

2.3.2.2.3. Autenticação

Quando um UAC envia uma requisição para um *proxy*, o *proxy* pode decidir autenticar o originador antes que a requisição seja processada. Ele pode enviar um desafio (*challenge*) para o originador através de uma mensagem 407 (*Proxy Authentication Required*) com o campo do cabeçalho *Proxy-Authorization* contendo o desafio. O cliente pode reenviar a requisição com o campo *Proxy-Authorization* do cabeçalho que provê as credenciais compatíveis com o desafio. O cliente pode prover as credenciais também antes de ser desafiado, de modo a evitar o retardo e o processamento extra da resposta 407 (as credenciais podem ser construídas de acordo com os desafios anteriores armazenados). No entanto, o desafio deve ser ainda válido para que o cliente seja autenticado. Caso tenha expirado, o *proxy* enviará novo desafio. Tanto o desafio quanto as credenciais são construídos usando um *hash* (código de autenticação da mensagem) criptografado, fazendo com que certos valores, como senhas, não sejam enviadas em texto.

Um valor de *hash* é um valor numérico de comprimento fixo derivado de uma seqüência de dados. Os valores de *hash* são usados para verificar a integridade dos dados enviados por canais não seguros. O valor do *hash* dos dados recebidos é comparado ao valor do *hash* dos dados enviados para determinar se foram alterados.

A autenticação feita pelo *proxy* é necessária para verificar que o originador da requisição é realmente um usuário autorizado a fazer uso dos serviços e garantir que certos campos da mensagem não sejam alterados por terceiros.

A Figura 6 mostra o *message flow* de uma autenticação.

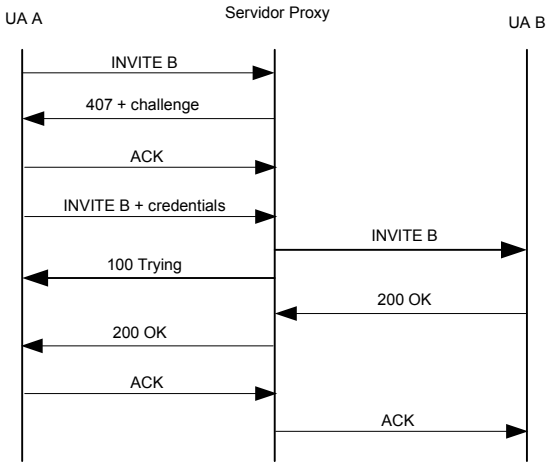

Figura 6 – Message flow de uma autenticação

2.3.2.2.4. Outbound Proxy

Outbound proxy é um *proxy* que recebe requisições de um cliente independentemente do destino das mensagens (*Request-URI*) que o cliente está enviando. Isto permite o uso de UAs mais simples, que não precisam tomar decisões de roteamento e fazer pesquisas no DNS.

2.3.3. Servidor Registrar

O servidor *registrar* é uma entidade SIP especial que recebe registros de usuários (através das mensagens REGISTER) dentro de um domínio, extrai a informação da localização atual (endereço IP, porta e nome do usuário, contido no campo *Contact*) e armazena em um serviço de localização. O serviço de localização é formado por *bindings* (associações) entre endereços da forma sip:usernameA@domainA e sip:usernameB@domainB (domainB pode ser escrito em forma de endereço IP:porta). UsernameA e usernameB podem ser iguais ou não. Este serviço de localização é então consultado pelo *proxy* do domínio que é responsável por encaminhar a mensagem.

A especificação do SIP não especifica a implementação do serviço de localização. O único requerimento é que o servidor *registrar* de um domínio precisa ler e escrever os dados em um serviço de localização, e que um *proxy* ou servidor *redirect* deste domínio seja capaz de ler estes dados.

Devido à estreita união entre *proxies* e *registrars*, usualmente eles são implementados juntos fisicamente.

Cada registro tem um tempo limitado de duração. O campo *Expires* do cabeçalho ou o parâmetro *expires* do campo *Contact* do cabeçalho determinam por quanto tempo o registro será válido. O UA pode atualizar seu registro de tempos em tempos para evitar que o mesmo expire.

A Figura 7 mostra o *registrar*, serviço de localização e *proxy* separados fisicamente.

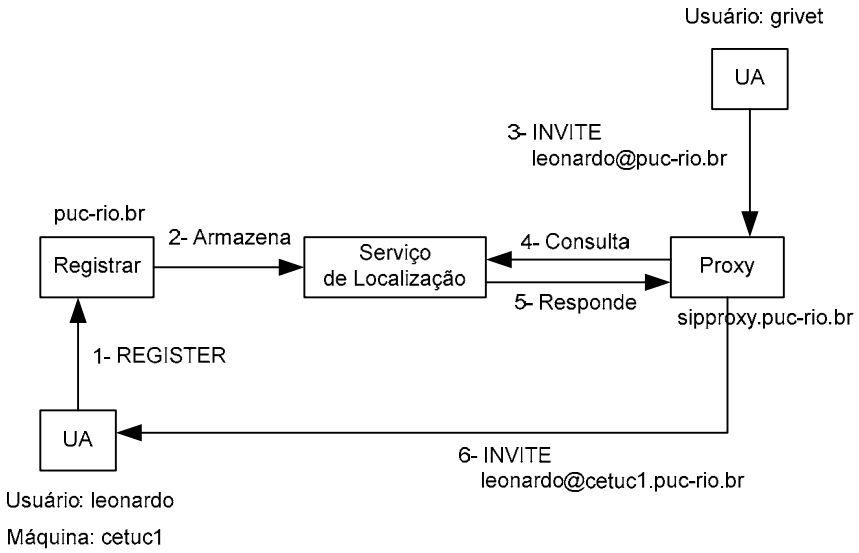

Figura 7 – Interação entre proxy, registrar e serviço de localização

2.3.4. Back-To-Back User Agent (B2BUA)

O B2BUA é uma entidade lógica que recebe uma requisição, a processa como um UAS, e de modo a determinar como a requisição será respondida, atua como um UAC e gera requisições. Diferentemente do *proxy*, ele mantém o estado do diálogo e precisa participar do envio de todas as requisições pertencentes ao diálogo que ele estabeleceu. A especificação do SIP não define o comportamento do B2BUA, mas o define como uma concatenação de um UAC e UAS.

Pode-se explicar o funcionamento do B2BUA como uma entidade que recebe uma requisição, a modifica e a envia como uma nova requisição. As respostas a esta requisição também são modificadas e encaminhadas ao destino. Como o B2BUA atua como UAS para um lado e como UAC para o outro, ele possui um controle maior sobre a chamada, já que mantém o estado

da chamada no nível de diálogo e não de transação, como no caso de *stateful proxies*. O B2BUA esconde do usuário chamado a identidade original do usuário chamador, funcionando como um *anonymizer*. Para isso, é necessário modificar a requisição alterando os cabeçalhos *From*, *Via*, *Contact*, *Call-ID*, e a informação de mídia do SDP, e modificar qualquer outro campo de cabeçalho que contenha a informação do usuário chamador. A resposta retornada pelo usuário chamado também terá o *Contact* e a informação de mídia do SDP modificada pelo B2BUA. A informação de mídia do SDP modificada apontará para o B2BUA, que encaminhará os pacotes RTP do usuário chamador para o usuário chamado e vice-versa. Desse modo, nenhum dos *end points* conhecerão a identidade um do outro (obviamente, o usuário chamador conhece o SIP URI do usuário chamado para que a chamada se estabeleça).

Abaixo, a Figura 8 [51] mostra a diferença de funcionamento entre um B2BUA e um servidor *proxy*.

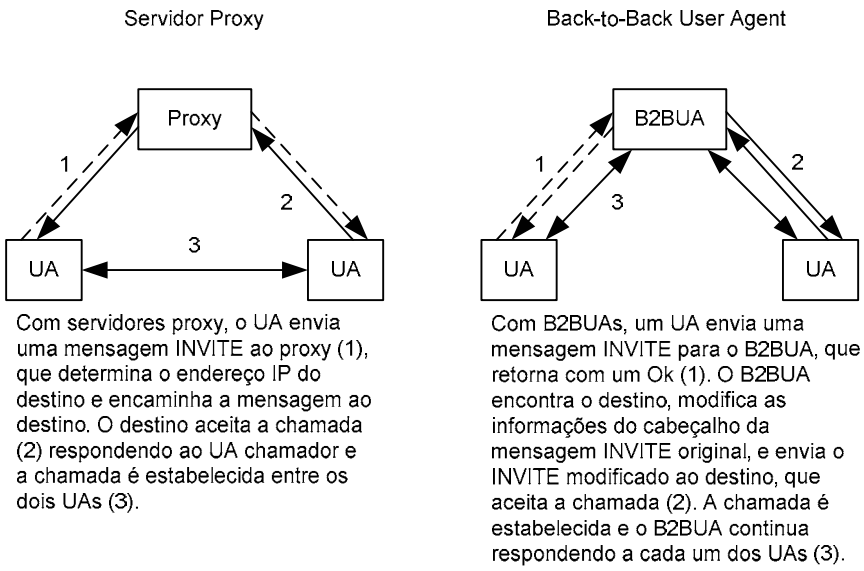

Figura 8 – Diferença de funcionamento entre um B2BUA e um servidor proxy

Algumas vezes, os B2BUA são usados para prover outros serviços. No entanto, eles contrariam a idéia do protocolo SIP de ser um protocolo *end-to-end*. O uso do B2BUA é benéfico em cenários onde um ou mais dos seguintes requisitos é necessário:

- Necessidade que a rede inicie ou modifique o estado do diálogo.
- Necessidade da rede desconectar chamadas (por exemplo, se o crédito de um telefone pré-pago acabou) ou modificar chamadas

(por exemplo, trocar de *codec* na fase de estabelecimento da chamada ou durante a chamada).

- Necessidade que a rede modifique mensagens de uma maneira em que é proíbido ao *proxy* (modificar/adicionar/remover campos do cabeçalho ou corpo da mensagem, criptografar/decriptografar a mensagem ou parte dela, compressão, etc).
- Esconder a identidade do usuário chamador.
- Manter-se atualizado em relação ao estado do diálogo (por exemplo, para fins de bilhetagem).
- Necessidade de transposição ao NAT e/ou ao *firewall*, fazendo com que o B2BUA atue como um ALG (*Application Level Gateway*).
- Necessidade de conexão de chamadas entre dois usuários através de um controlador (*Third Party Call Control* – 3PCC), definido na RFC 3725 [52].

Pode-se apontar as seguintes desvantagens no uso do B2BUA:

- É um ponto de falha único na rede, o que significa que seu uso diminui a confiabilidade das sessões SIP. Diferentemente do *proxy*, se o B2BUA falhar, todas as sessões em curso serão desconectadas, pois o B2BUA atua ativamente no diálogo.
- Há uma maior probabilidade de perda de pacotes e aumento da latência dos pacotes de mídia, já que os mesmos trafegam através do B2BUA.
- Maior consumo de memória por chamada, diminuindo a capacidade e escalabilidade comparado com o *proxy*.
- O processamento do estabelecimento e término de chamada é mais complexo (já que há necessidade de modificar a mensagem), diminuindo assim o desempenho em termos de chamadas por segundo.

2.3.5. Servidor Redirect

O servidor *redirect* é um tipo de servidor SIP que responde a requisições e não as encaminham. É um dos servidores mais simples. Ele recebe uma requisição e responde com uma mensagem de classe 3xx (redirecionamento)

informando ao UA uma lista de endereços alternativos através do campo *Contact* do cabeçalho.

Assim como o *proxy*, o servidor *redirect* faz uso do serviço de localização para gerar a resposta de classe 3xx com os endereços alternativos do usuário chamado.

O redirecionamento permite aos servidores coletar as informações de roteamento para uma requisição e enviá-las na resposta para um cliente, ajudando na localização do destino da requisição, enquanto são retirados do caminho das futuras mensagens para esta transação. Isto torna a rede mais escalável, pois propaga a informação de roteamento do *core* da rede para as bordas. Quando o originador da requisição receber a resposta de classe 3xx, ele enviará uma nova requisição (gerando uma nova transação), com mesmo *Call-ID*, *To* e *From*, mas valores diferentes de *branch ID* e *CSeq*.

A exceção para a resposta de classe 3xx é quando o servidor *redirect* recebe uma requisição CANCEL. Neste caso, ele retornará uma mensagem de classe 2xx. Essa resposta termina a transação SIP.

Os servidores *redirect* mantém o estado da transação durante toda a transação SIP, fazendo deles elementos *stateful* em transação. Eles desconhecem o estado dos diálogos.

A Figura 9 mostra um exemplo de *message flow* retirado da RFC 3665 [53].

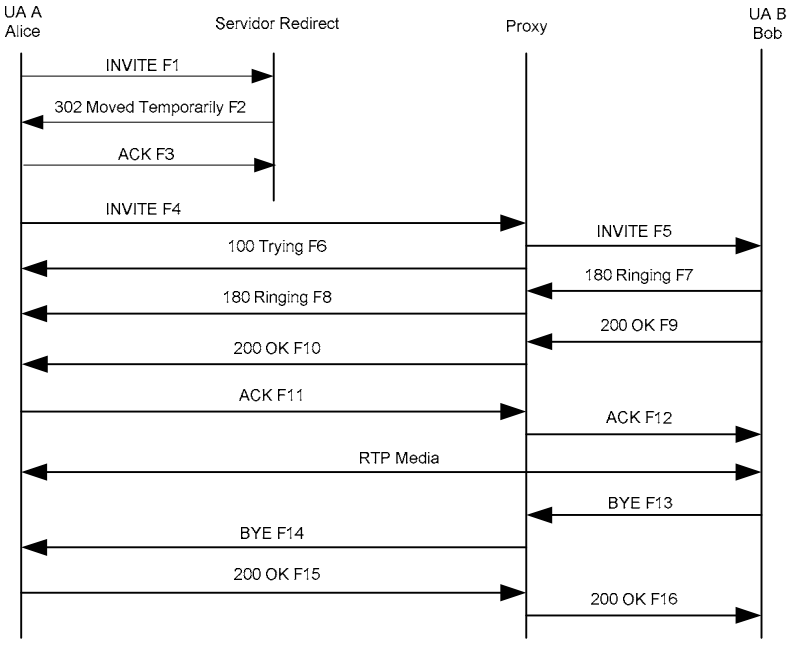

Figura 9 – Message flow de um redirecionamento de chamada

As mensagens de interesse são listadas abaixo.

F1 INVITE Alice -> Redirect Server

INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bKbf9f44
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 2xTb9vxSit55XU7p8@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com>
Content-Length: 0

F2 302 Moved Temporarily Redirect Proxy -> Alice

SIP/2.0 302 Moved Temporarily
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bKbf9f44
;received=192.0.2.101
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=53fHlqlQ2
Call-ID: 2xTb9vxSit55XU7p8@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:bob@chicago.example.com;transport=tcp>
Content-Length: 0

F3 ACK Alice -> Redirect Server

ACK sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bKbf9f44
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=53fHlqlQ2
Call-ID: 2xTb9vxSit55XU7p8@atlanta.example.com
CSeq: 1 ACK
Content-Length: 0

F4 INVITE Alice -> Proxy 3

```
INVITE sip:bob@chicago.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 2xTb9vxSit55XU7p8@atlanta.example.com
CSeq: 2 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Content-Length: 0
```

Note que o ACK reusa o mesmo *branch ID* das mensagens INVITE e 302. Isto porque, o ACK em resposta a uma mensagem de classe 3xx/4xx/5xx/6xx é considerado parte da transação do INVITE, além de ser uma resposta *hop-by-hop* e não *end-to-end*.

O campo *Contact* do cabeçalho pode conter URIs com novas localizações ou *usernames* diferentes, ou pode especificar parâmetros adicionais de transporte. Uma resposta 301 (*Moved Permanently*) ou 302 (*Moved Temporarily*) pode também prover a mesma localização e *username* que foi informada na requisição original, mas especificar parâmetros adicionais de transporte para nova tentativa como um servidor diferente ou endereço *multicast*, ou mudança no protocolo de transporte de UDP para TCP ou vice-versa.

No entanto, servidores *redirect* não podem redirecionar a requisição para um URI igual a do *Request-URI* (isto provocaria um *loop*). Ao invés disso, o servidor *redirect* deve encaminhar essa requisição como um servidor *proxy* ou retornar uma mensagem 404 (*Not Found*).

2.4. Mensagens SIP

2.4.1. Estrutura da mensagem SIP

O SIP é um protocolo baseado em texto que usa o conjunto de caracteres ISO 10646 com codificação UTF-8 (*8-bit unicode transformation format*) e usa uma sintaxe similar à do protocolo HTTP. A meta linguagem ABNF (*Augmented Backus-Naur Format*) é usada na RFC 3261 para descrever a sintaxe do SIP, assim como de outros protocolos da Internet.

Uma mensagem SIP pode ser uma requisição de um cliente para um servidor, ou uma resposta de um servidor para um cliente. Uma forma de descrever uma mensagem SIP através do formato ABNF é mostrada abaixo.

SIP-message = Request / Response

Isto significa que uma mensagem SIP pode ser uma requisição (*Request*) ou resposta (*Response*). *SIP-message* no lado esquerdo, antes do sinal de igual, representa o que está sendo definido. O que está no lado direito do sinal de igual representa a definição. O sinal "/" é usado com o significado de "OU".

Os dois tipos de mensagens consistem de um início de linha (*start-line*), um ou mais campos de cabeçalho (*header fields*), uma linha em branco representando o final do cabeçalho, e um corpo da mensagem (*message-body*) opcional.

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

O ínicio de linha, cada campo de cabeçalho e a linha em branco devem terminar com um CRFL (*Carriage-Return Line-Feed sequence*).

Na terminologia da ABNF, o símbolo "*" significa que um elemento pode ser repetido, separado no mínimo por um espaço. Os elementos entre parênteses são tratados como um único elemento. O espaço é representado por SP. Elementos opcionais são representados entre colchetes.

Uma requisição é representada por:

```
Request  =  Request-Line
            *message-header
            CRLF
            [ message-body ]
```

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

Method = "INVITE" / "ACK" / "OPTIONS" / "BYE" / "CANCEL" / "REGISTER"

E uma resposta é representada por:

Response = Status-Line
*message-header
CRLF
[message-body]

Status-Line = SIP-version SP Status-Code SP Reason-Phrase CRLF
Status-Code = Informational / Success / Redirection / Client-Error / Server-Error / Global-Failure / extension-code

A descrição das mensagens através da ABNF é explicada na seção 25 da RFC 3261.

2.4.2. SIP URI (Uniform Resource Indicator)

SIP URIs são utilizados em vários lugares incluindo *Request-URI* e os campos *To*, *From* e *Contact* do cabeçalho. Serve para identificar um recurso de comunicação e segue as regras definidas na RFC 3986 [48]. Eles contêm informações suficientes para iniciar e manter a sessão com um recurso. O formato geral de um SIP URI é mostrado abaixo.

sip:user:password@host:port;uri-parameters?headers

O campo *user* identifica o recurso em particular pertencente a um *host*. A parte do URI chamado de *userinfo* consiste dos campos *user*, *password* e do sinal de @. O *userinfo* é opcional e não é usado, por exemplo, no *Request-URI* de uma mensagem REGISTER. O campo *password* traz a senha associada ao usuário. Embora a sintaxe contenha este campo, a especificação não recomenda seu uso, já que a senha é enviada sem criptografia. O campo *host* contém o recurso SIP, podendo ser um domínio ou um endereço IPv4 ou IPv6. O campo *port* designa a porta em que a requisição será enviada. Quando não está presente, utiliza-se a porta padrão do SIP, que é a 5060. O campo *uri-parameters* é composto por parâmetros que afetam o modo como a requisição será tratada. Vários parâmetros podem ser utilizados, representados na forma

"*parameter-name* = *parameter-value*" e separados por ";". Não é permitida a repetição de parâmetros já utilizados. O parâmetro *transport* é um exemplo de *uri-parameter* e indica o protocolo de transporte (UDP, TCP ou SCTP) que será utilizado. Quando omitido, utiliza-se o protocolo padrão do SIP para transporte, que é o UDP. Outro exemplo de *uri-parameter*, é o parâmetro *user*, que pode estar presente como *user=phone* (indica que o campo *user* deve ser interpretado como um número de telefone) ou *user=ip* (que é o padrão). O campo *headers*, depois do sinal "?", indica os cabeçalhos que devem ser adicionados na construção de uma requisição a partir de um URI. É similar a operação do URL *mailto*, que possibilita o uso dos parâmetros *Subject* (assunto) e *Priority* (prioridade) na construção de uma mensagem. Mais de um parâmetro pode ser utilizado no campo *headers*, representados por "*hname* = *hvalue*" e sendo separados pelo sinal "&". Para maiores informações, consultar a seção 19 da especificação do SIP.

2.4.3. Requisições SIP

A RFC 3261 define seis tipos de requisições ou métodos que são usados para requisitar um tipo específico de ação que deve ser executada por um *proxy* ou UA. São eles: INVITE, REGISTER, BYE, ACK, CANCEL, e OPTIONS. Existem outros métodos definidos em outras RFCs que não serão tratados neste trabalho.

2.4.3.1. INVITE

O método INVITE é usado para estabelecer uma sessão de mídia entre dois UAs. É similar à mensagem *Setup* do protocolo Q.931 ou à mensagem IAM (*Initial Address Message*) do ISUP. O INVITE é sempre confirmado através de uma mensagem ACK, ou seja, o início da sessão é estabelecido após o *three-way hand-shaking*.

O INVITE possui tipicamente um corpo de mensagem (*message body*) com a descrição da sessão (através do protocolo SDP). Caso o INVITE não possua a descrição da sessão, a mensagem ACK deverá incluí-la. Caso a descrição da sessão não seja aceita pelo UAS, o mesmo deverá enviar uma mensagem BYE para terminar a sessão.

O UAC que origina o INVITE para estabelecer um diálogo gera um identificador único chamado de *Call-ID* que é usado durante todo o diálogo no cabeçalho. O campo *CSeq* do cabeçalho é usado para manter a ordem das mensagens. Como as requisições podem ser enviadas através de um protocolo não confiável, as mensagens podem chegar fora de ordem, e o número de seqüência deve estar presente nas mensagens de modo que o destino possa identificar retransmissões e mensagens fora de ordem. Os campos *From* e *To* servem para identificar o usuário chamador e usuário chamado, respectivamente (assim como no SMTP). No campo *From* do INVITE é adicionado um parâmetro *tag* pelo UAC e no campo *To* das respostas é adicionado um parâmetro *tag* pelo UAS, como será explicado na seção 2.4.7. A *tag* do campo *To* da mensagem 200 OK em resposta ao INVITE é usado no cabeçalho *To* da mensagem ACK e em todas as mensagens subseqüentes do diálogo. A combinação das *tags* do *To* e *From* e *Call-ID* formam um identificador único para o diálogo. O campo *Via* é usado para gravar o caminho da requisição. Depois ele é usado para rotear as respostas exatamente pelo mesmo caminho no sentido inverso.

Uma mensagem INVITE enviada em um diálogo já existente deve utilizar o mesmo *Call-ID* e *tags* dos campos *To* e *From*. É chamado de re-INVITE e é usado para mudar alguma característica da mídia utilizada na sessão. O cabeçalho *CSeq* deve ser incrementado para que a mensagem possa ser distinguida de uma retransmissão. Caso o re-INVITE não seja aceito, a sessão continua como antes.

Abaixo segue um exemplo de uma mensagem INVITE.

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Content-Type: application/sdp
Content-Length: 151

(SDP omitido)
```

A primeira linha identifica o método INVITE. O URI da primeira linha é chamado de *Request-URI* e expressa o próximo *hop* ou salto da mensagem.

Os campos do cabeçalho mandatórios do INVITE são: *Call-ID*, *CSeq*, *From*, *To*, *Via*, *Contact* e *Max-Forwards*.

2.4.3.2. REGISTER

O método REGISTER é utilizado para informar ao servidor *registrar* a atual localização de um UA. Essa informação está contida no campo *Contact* do cabeçalho. O servidor *registrar* correlaciona o SIP URI do campo *To*, publicamente conhecido, com o SIP URI do campo *Contact*, que revela a atual localização do usuário. Esse serviço de localização utilizado pelo *registrar* para armazenar essa correlação de endereços é utilizado pelo *proxy* para rotear as chamadas para os usuários.

Dependendo do uso dos campos *Contact* e *Expires*, o servidor *registrar* tomará ações diferentes. Se o parâmetro *expires* ou o campo *Expires* não forem utilizados, o registro do SIP URI será cancelado em uma hora.

O uso do *Request-URI* e dos campos *To*, *From* e *Call-ID* na mensagem REGISTER é um pouco diferente comparado às outras mensagens. O *Request-URI* contém apenas o domínio do servidor *registrar* (não possui o usuário). O REGISTER pode ser encaminhado por um *proxy* até chegar ao servidor *registrar* responsável pelo domínio em questão. O campo *To* contém o SIP URI que será registrada no servidor *registrar* (chamado de *address-of-record*, ou AOD). O campo *From* contém o SIP URI do originador da requisição, tipicamente igual à do cabeçalho *To*, mas podendo ser diferente no caso de um usuário realizar o registro para um outro usuário. É recomendado que o mesmo *Call-ID* seja usado para todos os registros de um UA.

Um REGISTER pode receber como resposta mensagens de classe 3xx ou 4xx contendo o endereço para o qual o registro deve ser feito no campo *Contact* do cabeçalho.

Um exemplo de mensagem REGISTER é mostrado abaixo:

```
REGISTER sips:ss2.biloxi.example.com SIP/2.0
Via: SIP/2.0/TLS client.biloxi.example.com:5061;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Bob <sips:bob@biloxi.example.com>;tag=a73kszlfl
```

To: Bob <sips:bob@biloxi.example.com>
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 REGISTER
Contact: <sips:bob@client.biloxi.example.com>
Content-Length: 0

Os campos de cabeçalho mandatórios do REGISTER são: *Call-ID*, *CSeq*, *From*, *To*, *Via* e *Max-Forwards*.

2.4.3.3. BYE

O método BYE é utilizado para terminar sessões já estabelecidas. É similar ao REL (*Release*) do ISUP. Uma sessão é considerada já estabelecida se o UAC recebeu uma mensagem de classe 2xx originada do UAS. A requisição BYE é sempre enviada por UAs que participam da sessão, nunca por *proxies*. É um método *end-to-end*, isto é, as respostas são geradas apenas pelo outro UA.

Um exemplo de mensagem da requisição BYE é mostrado abaixo.

BYE sip:alice@client.atlanta.example.com SIP/2.0
Via: SIP/2.0/TCP client.biloxi.example.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 BYE
Content-Length: 0

Os campos de cabeçalho mandatórios da requisição BYE são os mesmos da requisição REGISTER.

2.4.3.4. ACK

Este método é usado para confirmar o recebimento da resposta final do INVITE, completando assim o *three-way handshake*. Respostas finais são definidas como respostas de classes 2xx, 3xx, 4xx, 5xx e 6xx. O *CSeq* nunca é incrementado quando se envia um ACK, de modo que o UAS possa

correlacionar o INVITE com o ACK correspondente. O campo *To* da mensagem ACK é copiado da resposta final, que possui o parâmetro *tag* (isto difere do campo *To* da mensagem INVITE). O ACK deve conter apenas um campo *Via* que deve ser igual ao da requisição INVITE.

Uma mensagem ACK pode conter um corpo da mensagem, embora seja um caso especial, já que uma requisição ACK não pode ser rejeitada caso o corpo da mensagem não seja entendido. Incluir corpo na mensagem ACK em resposta a mensagem que não seja de classe 2xx (non-2xx) não é recomendado. No caso de mensagens de classe 2xx, o corpo da mensagem ACK é usado apenas se o SDP não foi enviado no INVITE.

Para respostas de classe 2xx, o ACK é *end-to-end*. Em todas as outras respostas, o ACK é *hop-by-hop* quando um *stateful proxy* é usado. A razão do envio da mensagem ACK ser *end-to-end* em resposta a mensagens de classe 2xx é devido a possibilidade do ACK conter um corpo de mensagem (o corpo deve ser gerado apenas pelo UA).

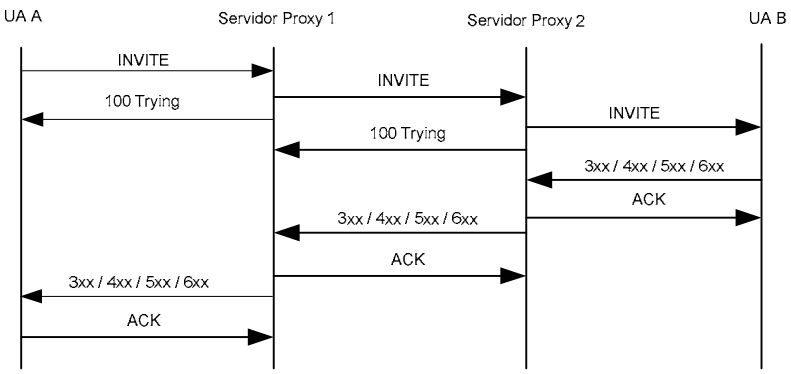

Figura 10 – ACK hop-by-hop

O ACK *hop-by-hop* (Figura 10) utiliza o mesmo *branch ID* da mensagem INVITE, já que é considerado parte da mesma transação. O ACK *end-to-end* (Figura 11) usa um novo *branch ID*, apesar de não se encaixar no conceito de transação (consultar a seção 2.4.6).

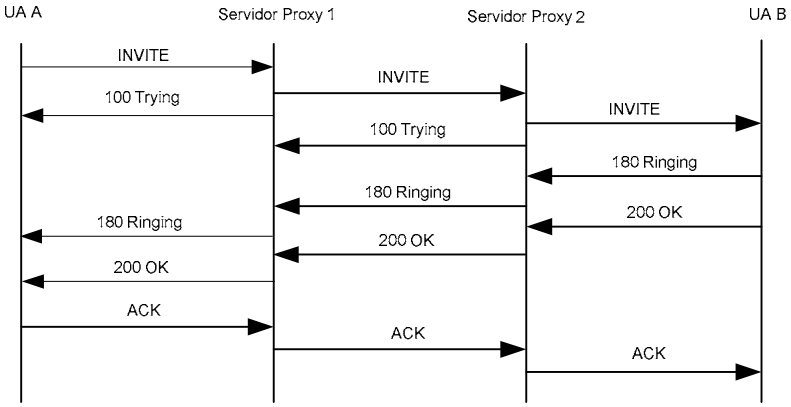

Figura 11 – ACK end-to-end com Record-Route

Um exemplo de uma mensagem ACK é mostrado abaixo.

ACK sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bd5
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 ACK
Content-Length: 0

Os campos de cabeçalho mandatórios da requisição ACK são os mesmos da requisição REGISTER.

2.4.3.5. CANCEL

O método CANCEL é usado para cancelar sessões que ainda não foram estabelecidas. É usado quando o usuário chamado ainda não enviou uma mensagem de resposta final. É comumente usado por *proxies* que fazem *forking* paralelo para cancelar mensagens INVITE pendentes quando há o primeiro atendimento. O CANCEL é um método *hop-by-hop*. O campo *CSeq* de mensagens CANCEL nunca é incrementado de modo que os *proxies* ou UAs possam correlacionar a mensagem CANCEL com a mensagem INVITE correspondente, que possui o mesmo *CSeq*. O *branch ID* também é igual ao da mensagem INVITE a ser cancelada.

Um *proxy* que recebe uma mensagem CANCEL, a encaminha para os mesmos *hops* para os quais as mensagens INVITE pendentes foram encaminhadas. O *proxy* não espera pela resposta das mensagens CANCEL encaminhadas, as responde imediatamente (com um 200 OK), mesmo sem saber se a requisição deu resultado. Caso a mensagem CANCEL tenha sido enviada no momento em que o UAS estava enviando uma resposta de classe 2xx ao INVITE, o CANCEL não funcionará e a mensagem BYE deverá ser usada pelo UAC.

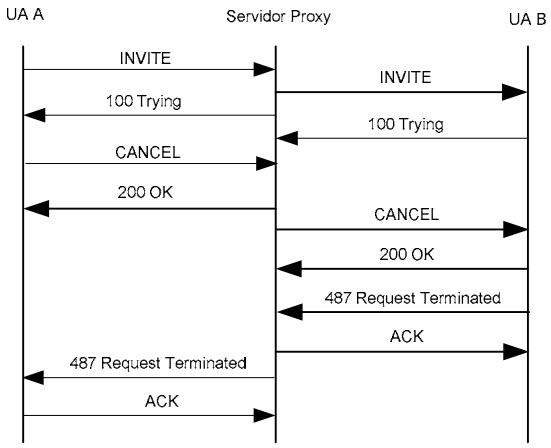

Figura 12 – Processamento da requisição CANCEL

Um UA confirma o cancelamento através de uma mensagem 200 OK e responde com uma mensagem 487 Request Terminated.

Um exemplo de uma mensagem CANCEL é mostrado abaixo.

CANCEL sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/UDP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 2xTb9vxSit55XU7p8@atlanta.example.com
CSeq: 1 CANCEL
Content-Length: 0

Os campos de cabeçalho mandatórios da requisição CANCEL são os mesmos da requisição REGISTER.

2.4.3.6. OPTIONS

O método OPTIONS é usado para interrogar um UA ou servidor sobre suas capacidades ou atual disponibilidade. Um UA ou servidor responde a uma mensagem OPTIONS da mesma maneira que a uma mensagem INVITE. Caso não esteja disponível, responderá com uma mensagem 4xx ou 6xx. A resposta lista todas as capacidades de um UA ou servidor. A mensagem OPTIONS nunca será gerada por um servidor, apenas por UAs.

Os campos de cabeçalho *Allow*, *Accept*, *Accept-Encoding*, *Accept-Language* e *Supported* devem estar na mensagem 200 OK em resposta ao OPTIONS.

Um exemplo de uma mensagem OPTIONS e de sua resposta 200 OK são mostrados abaixo.

```
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

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 omitido)

Os campos de cabeçalho mandatórios da requisição OPTIONS são os mesmos da requisição REGISTER.

2.4.4. Respostas SIP

As respostas são mensagens geradas por UAS ou servidores em reposta a uma requisição gerada por um UAC. Uma reposta pode simplesmente confirmar o recebimento de uma requisição para evitar futuras retransmissões ou pode conter informações necessárias para o estabelecimento da sessão através de campos de cabeçalho e/ou informações sobre mídia.

As respostas são divididas em seis classes. As cinco primeiras foram aproveitadas da especificação do HTTP/1.1. A sexta classe foi criada especificamente para o SIP.

As classes são diferenciadas pelo primeiro algarismo de um número de três algarismos contido na resposta, chamado de *Status-Code*. É um número que indica o resultado da tentativa de entender e satisfazer a requisição e é para uso de “máquinas”. Ao lado do *Status-Code*, é mostrado o resultado em forma de frase chamado de *Reason-Phrase* e destina-se ao uso “humano”. A especificação apenas sugere o uso de *Reason-Phrases*, as implementações não são obrigadas a seguí-las, podendo traduzi-las, por exemplo.

Nem todos os códigos foram aproveitados do HTTP/1.1, apenas os apropriados para o SIP. Geralmente, todos os códigos acima de x80 foram criados especificamente para o SIP.

Se um código contido em uma resposta enviada por um UAS ou servidor não for entendida por um cliente, esta será interpretada como uma mensagem com código x00, onde x é a classe da mensagem original.

As classes são mostradas e explicadas na Tabela 2.

Classe	Descrição	Ação
1xx	*Informational* (Informativo)	Indica que a requisição foi recebida e que o estabelecimento da sessão está em andamento. Este tipo de mensagem é *end-to-end* e pode conter um corpo. A exceção é a mensagem 100 Trying que é *hop-by-hop* e não deve conter um corpo.
2xx	*Success* (Bem sucedido)	A mensagem 200 OK pode ser usada em duas situações: 1- Em resposta a uma mensagem INVITE. Indica que a sessão foi estabelecida. Esta resposta conterá um corpo com a descrição da mídia do UAS. 2- Em resposta a outras requisições. Indica que a requisição foi recebida, processada e bem sucedida.
3xx	*Redirection* (Redirecionado)	Indica que uma ação mais adiante precisa ser tomada para que a requisição possa ser completada, isto é, uma nova requisição deve ser gerada pelo UAC ou *proxy* para o(s) endereço(s) contido(s) no cabeçalho *Contact* da resposta.
4xx	*Client Error* (Erro do cliente)	Indica que a requisição não pôde ser executada pelo servidor do modo em que foi recebida. A resposta específica do erro do cliente ou a presença de certo tipo de campo de cabeçalho deve indicar ao UAC a razão do erro e como a requisição deve ser reformulada. A nova requisição deve ser corrigida antes de ser enviada novamente.
5xx	*Server Error* (Erro do servidor)	Indica que o servidor falhou em executar uma requisição aparentemente válida. A requisição pode ser enviada a outro servidor.
6xx	*Global Error* (Falha global)	Indica que a requisição falhou. A requisição não deve ser reenviada a este ou a outros servidores. Apenas o servidor que tiver conhecimento definitivo sobre um usuário em particular deve utilizar este tipo de resposta.

Tabela 2 – Classes de respostas SIP

Todas as respostas definidas pela especificação do SIP estão listadas na Tabela 3. Na maior parte das vezes, elas são auto-explicativas. Para maiores detalhes, consultar a seção 21 da RFC 3261.

100 Trying	380 Alternative Service	410 Gone	482 Loop Detected	501 Not Implemented
180 Ringing	400 Bad Request	413 Request Entity Too Large	483 Too Many Hops	502 Bad Gateway
181 Call Is Being Forwarded	401 Unauthorized	414 Request-URI Too Long	484 Address Incomplete	503 Service Unavailable
182 Queued	402 Payment Required	415 Unsupported Media Type	485 Ambiguous	504 Server Time-out
183 Session Progress	403 Forbidden	416 Unsupported URI Scheme	486 Busy Here	505 Version Not Supported
200 OK	404 Not Found	420 Bad Extension	487 Request Terminated	513 Message Too Large
300 Multiple Choices	405 Method Not Allowed	421 Extension Required	488 Not Acceptable Here	600 Busy Everywhere
301 Moved Permanently	406 Not Acceptable	423 Interval Too Brief	491 Request Pending	603 Decline
302 Moved Temporarily	407 Proxy Authentication Required	480 Temporarily Unavailable	493 Undecipherable	604 Does Not Exist Anywhere
305 Use Proxy	408 Request Timeout	481 Call/Transaction Does Not Exist	500 Server Internal Error	606 Not Acceptable

Tabela 3 – Respostas SIP definidas na RFC 3261

2.4.5. Campos de Cabeçalho

Os campos de cabeçalho do SIP são similares aos do HTTP em semântica e sintaxe. A especificação está em conformidade com a RFC 2234 [54] e cada campo de cabeçalho é definido da forma "*field-name: field-value*". Alguns campos de cabeçalho possuem uma forma compacta de representação e podem ser *hop-by-hop* ou *end-to-end*. Os campos de cabeçalho *hop-by-hop* são os únicos em que *proxies* podem inserir, ou em alguns casos, modificar. A Tabela 4 relaciona todos os campos de cabeçalho definidos na especificação do SIP. Para maiores detalhes, consultar as seções 7.3 e 20 da especificação.

Accept	Content-Encoding	Min-Expires	Route
Accept-Encoding	Content-Language	MIME-Version	Server
Accept-Language	Content-Length	Organization	Subject
Alert-Info	Content-Type	Priority	Supported
Allow	CSeq	Proxy-Authenticate	Timestamp
Authentication-Info	Date	Proxy-Authorization	To
Authorization	Error-Info	Proxy-Require	Unsupported
Call-ID	Expires	Record-Route	User-Agent
Call-Info	From	Reply-To	Via
Contact	In-Reply-To	Require	Warning
Content-Disposition	Max-Forwards	Retry-After	WWW-Authenticate

Tabela 4 – Campos de cabeçalho definidos na RFC 3261

Alguns destes campos de cabeçalho já foram apresentados ao longo deste capítulo.

2.4.6. Transações

Embora as mensagens do SIP sejam enviadas independentemente através da rede, elas são usualmente separadas em transações pelos UAs e certos tipos de servidores *proxy*. Assim, diz-se que o protocolo SIP é um protocolo transacional.

Uma transação SIP consiste de uma única requisição e respostas desta requisição, que inclui zero ou mais respostas provisórias e uma ou mais respostas finais.

Se uma transação foi iniciada por uma requisição INVITE e se a resposta final não for da classe 2xx, então a requisição ACK fará parte da mesma transação. Se a resposta final for da classe 2xx, então a requisição ACK não é considerada parte da transação. A Figura 13 ilustra esta situação.

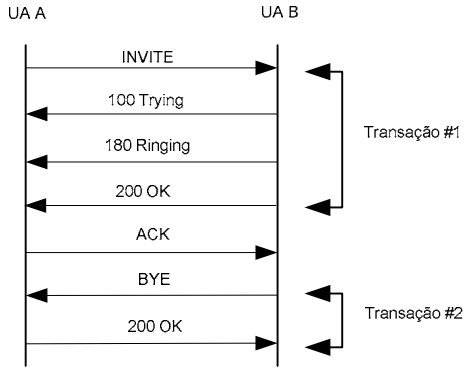

Figura 13 – Transações em uma chamada

Transações possuem um lado de cliente e um lado de servidor. O lado de cliente é conhecido como transação de cliente e o lado de servidor é conhecido como transação de servidor. A transação de cliente envia requisições, e a transação de servidor envia as respostas. As transações de cliente e de servidor são funções lógicas que estão presentes em qualquer UA ou *stateful proxy*. As transações são transparentes para *stateless proxies*. A função da transação de cliente é receber a requisição da transação de usuário (TU) e encaminhá-la confiavelmente para a transação de servidor.

A figura abaixo mostra a relação entre as transações de cliente (TC) e de servidor (TS).

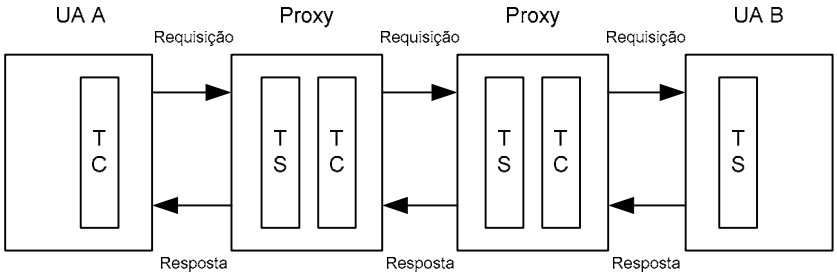

Figura 14 – Relações entre transações

A transação de cliente também é responsável por receber respostas e encaminhá-las a TU, filtrando as respostas retransmitidas e as respostas não permitidas (como uma resposta para um ACK). No caso de uma requisição INVITE, a transação de cliente é responsável por gerar a requisição ACK após receber uma resposta de classe 2xx.

Similarmente, a função da transação de servidor é receber requisições da camada de transporte e encaminhá-las a TU. A transação de servidor filtra as

retransmissões de requisições e recebe respostas da TU, encaminhando-as para a camada de transporte para transmissão pela rede.

Na RFC 2543, o identificador de transações era calculado como *hash* de todos os campos de cabeçalho importantes, incluindo *To*, *From*, *Request-URI* e *CSeq*. Isto provou ser muito lento e complexo [55].

Na RFC 3261, a maneira de calcular o identificador de transações foi completamente alterado. Agora, um parâmetro *branch* do campo de cabeçalho *Via* contém diretamente o identificador de transação. Este parâmetro é utilizado tanto por clientes como por servidores.

O valor do parâmetro *branch* precisa ser único no tempo e espaço para todas as requisições enviadas por um UA. As exceções para esta regra são as requisições CANCEL e as requisições ACK para respostas de classe *non*-2xx. No caso de requisições CANCEL, o valor do parâmetro *branch* deve ser o mesmo da requisição que ele cancelará. E no caso de requisições ACK para respostas *non*-2xx, as requisições terão o mesmo *branch ID* que a requisição INVITE que se está confirmando.

O valor do parâmetro *branch* sempre começa com os caracteres "z9hG4bK".

2.4.7. Diálogos

Um diálogo representa uma relação SIP *peer-to-peer* entre dois UAs durante um certo intervalo de tempo e é um conceito muito importante para UAs. Os diálogos facilitam o seqüenciamento e roteamento das mensagens entre UAs.

Diálogos são identificados pelo campo *Call-ID*, uma *tag* no campo *From* e outra *tag* no campo *To*. Mensagens que possuem os mesmos três identificadores pertencem ao mesmo diálogo.

O *Call-ID* é um identificador da chamada e deve ser uma *string* que identifica unicamente a chamada. Uma chamada consiste de um ou mais diálogos. No caso de *forking*, quando o UAC estabelece sessões com vários UAS, todos possuem o mesmo *Call-ID* e por isso todos fazem parte da mesma chamada.

A *tag* é um número aleatório criptografado com pelo menos 32 bits de aleatoriedade, que é adicionado como parâmetro aos campos *To* e *From* do cabeçalho para identificarem unicamente um diálogo (as tags não pertencem às

URIs do *To* e *From*, ficando sempre fora o "< >"). O campo *To* na mensagem inicial INVITE não deve conter a *tag*. O usuário chamador deve incluir a *tag* no campo *From*. Excluindo o 100 Trying, todas as respostas deverão ter uma *tag* adicionada ao campo *To* do cabeçalho. O envio ou recebimento de uma resposta contendo uma *tag* no campo *From* cria o que chamamos de *early dialog*. A *tag* do campo *To* retornada por uma mensagem 200 OK é então incorporada como identificador de diálogo e usada em todas as futuras requisições para este *Call-ID*. Resumidamente, o parâmetro *tag* do campo *To* é gerado pelo usuário chamado e o parâmetro *tag* do campo *From* é gerado pelo usuário chamador.

Caso um UAC receba respostas com diferentes *tags*, isso significa que são respostas de diferentes UASs, e assim que o INVITE foi ramificado. É decisão do UAC como lidar com esta situação. Ele poderá estabelecer sessões separadas com os UASs que responderem, e nesse caso os diálogos terão o mesmo *From* (incluindo o *tag*), *Call-ID* e *CSeq*, mas terão os campos de cabeçalho *To* diferentes por causa da *tag*. O UAC poderá também enviar um BYE para algumas ramificações e estabelecer apenas uma sessão.

De fato, o *CSeq* é usado para ordenar mensagens pertencentes a um diálogo. Seu valor deve ser monotonicamente aumentado para cada mensagem enviada dentro de um diálogo, caso contrário o UA que receber estas mensagens as tratará como mensagens fora de ordem ou retransmissões.

A Figura 15 ilustra o estabelecimento e término de um diálogo.

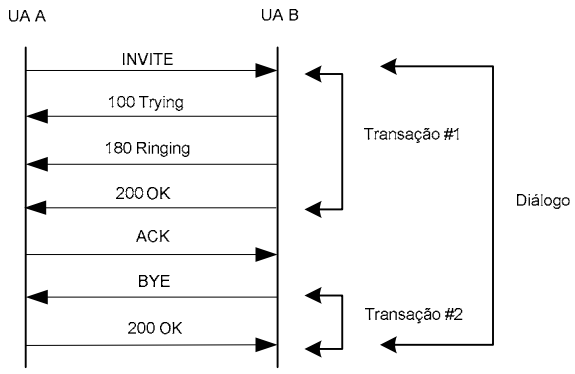

Figura 15 – Diálogo x Transação

Apenas respostas 2xx e 101-199 com parâmetro *tag* no campo *To*, onde a requisição foi um INVITE, estabelecerá um diálogo.

2.4.8. Record-Routing

Record-Routing é um mecanismo que permite que *proxies* possam ficar no caminho da sinalização para todas as mensagens futuras dentro de um determinado diálogo. Para isso, o *proxy* deve inserir o campo de cabeçalho *Record-Route* com o seu endereço na mensagem inicial de estabelecimento do diálogo (mensagem INVITE). O UAS e UAC constroem listas de roteamento baseados no cabeçalho de *Record-Route* que eles acham na requisição e enviam cada requisição subseqüente com uma lista de endereços forçando o roteamento através desses endereços com o campo de cabeçalho *Route*.

A figura abaixo mostra um *message flow* sem e com *Record-Route*.

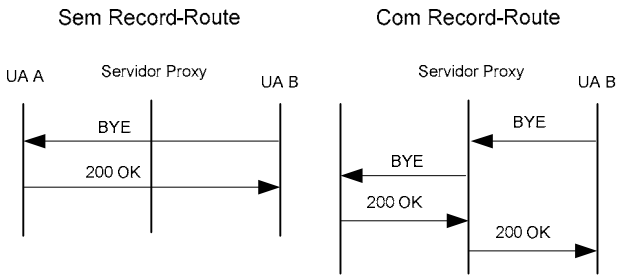

Figura 16 – Diferença no roteamento com o cabeçalho Record-Route

Um *proxy* que não utilize o recurso de *Record-Route* não deve esperar que requisições subseqüentes ao INVITE passem através dele. Existem certas situações onde o *proxy* precisa ficar no caminho da sinalização para recebimento das mensagens subseqüentes como, por exemplo, *proxies* que suportam NAT ou *proxies* responsáveis por bilhetagem.

É muito importante usar o recurso do *Record-Route* de modo seletivo, pois ele permite manter o *proxy* "dentro" de um diálogo por toda a sua duração, enquanto outros *proxies* dão assistência apenas no estabelecimento da sessão e depois se retiram do caminho da sinalização. Isto torna o SIP um protocolo mais escalável, já que apenas um pequeno número de mensagens passa pelos *proxies*. Além de evitar que *proxies* gastem recursos para "acompanhar" diálogos em que eles não tem interesse.

2.4.9. Confiabilidade do SIP

O protocolo SIP é suportado tanto pelo protocolo UDP, protocolo não confiável, como pelo TCP ou SCTP, protocolos confiáveis. Quando o SIP usa o UDP, ele tem que garantir a entrega da mensagem, já que o UDP não o faz. O UDP também não garante o seqüenciamento das mensagens, ou seja, as mensagens podem chegar fora de ordem ao destino. Logo, foram adicionados ao SIP mecanismos para prover a confiabilidade na entrega das mensagens, que são basicamente *timers* de *time out* e retransmissão, além do uso do *three-way handshake* e de cabeçalhos que identificam o seqüenciamento das mensagens. Quando o SIP usa o TCP ou SCTP, protocolos confiáveis, os *timers* de retransmissão do SIP não são utilizados, já que estes protocolos já possuem seus mecanismos próprios para garantir a entrega das mensagens ao destino em caso de perda.

A RFC 3261 (itens 17.1 e 17.2) define o esquema de retransmissões e *time outs* através de máquinas de estados ou *state machines*. Aqui, para simplificar, relacionou-se abaixo apenas os *timers* definidos na RFC 3261.

- *Timer* T1 – É uma estimativa do valor de RTT (*Round-Trip Time*) e seu valor padrão é de 500ms. Elementos (clientes ou servidores) podem usar um valor menor que 500ms (embora não recomendado), como por exemplo, em redes privadas. No entanto, em caso de redes com alta latência (com RTT maior que 500 ms), é recomendado que T1 seja aumentado.
- *Timer* T2 – Representa o intervalo de tempo máximo em que um elemento demora para responder a uma requisição *non*-INVITE em caso de perda e é também usado como intervalo de tempo para retransmissão de respostas ao INVITE de classe 300-699. O valor padrão é fixado em 4 segundos.
- *Timer* T4 – Representa o tempo máximo que uma mensagem trafega na rede. O valor padrão é fixado em 5 segundos.
- *Timer* A – Representa o intervalo de tempo das retransmissões do INVITE. Em caso de recebimento de uma resposta de classe 1xx ou 2xx, o *timer* é finalizado. O valor inicial começa em T1, dobrando de valor até atingir 64*T1 (totalizando o envio de 7 requisições

INVITE). É utilizado apenas por protocolos de transporte não confiáveis.

- *Timer* B – Representa o *time out* da transação iniciada pelo INVITE. É disparado sempre que um INVITE é enviado, independente do protocolo de transporte. Seu valor é fixado em 64*T1.
- *Timer* C – Representa o *time out* da transação iniciada por um *proxy* através da requisição INVITE. Valor deve ser superior a 3 minutos.
- *Timer* D – É disparado após o envio de um ACK confirmando o recebimento de uma resposta de classe 300-699. Representa um tempo de espera para recebimento de possíveis retransmissões de mensagens de classe 300-699. O valor é de pelo menos 32 segundos para protocolos de transporte não confiáveis e 0 segundo para protocolos de transporte confiáveis.
- *Timer* E – Representa o intervalo de tempo de retransmissões de requisições *non*-INVITE. O timer é finalizado quando se recebe uma resposta de classe 200-699. O valor do *Timer* E começa em T1, depois passa para MIN(2*T1,T2) e assim por diante até atingir T2, quando esse tempo é fixado. Quando se recebe uma resposta de classe 1xx, e o *Timer* E expira, seu valor é fixado em T2. É usado apenas para protocolos de transporte não confiáveis.
- *Timer* F – Representa o *time out* da transação iniciada por uma requisição *non*-INVITE. É disparado sempre que uma requisição é enviada, independente do protocolo de transporte. Seu valor é fixado em 64*T1.
- *Timer* G – É usado para retransmissões de respostas ao INVITE de classe 300-699. Usado apenas para protocolos de transporte não confiáveis. O valor do *Timer* G começa em T1, depois passa para MIN(2*T1,T2) e assim por diante até atingir T2, quando esse tempo é fixado.
- *Timer* H – Tempo de espera para recebimento do ACK. Representa o tempo em que a transação de servidor desiste de retransmitir uma resposta de classe 300-699. É usado para todos os protocolos de transporte. O valor do *Timer* G é igual ao do *Timer* B.
- *Timer* I – É disparado quando já houve o recebimento de um ACK e representa um tempo de espera para recebimento de possíveis

retransmissões do ACK. Seu valor é de T4 para protocolos de transporte não confiáveis e 0 segundo para protocolos de transporte confiáveis.

- *Timer* J – É disparado quando já houve o recebimento de uma resposta final (classe 200-699) e representa um tempo de espera para recebimento de possíveis retransmissões de requisições *non-*INVITE. O valor é fixado em 64*T1 para protocolos de transporte não confiáveis e 0 segundo para protocolos de transporte confiáveis.
- *Timer* K – É disparado quando já houve o recebimento de uma resposta final (classe 200-699) e representa um tempo de espera para recebimento de possíveis retransmissões de respostas de classe 200-699. O valor é fixado em T4 para protocolos de transporte não confiáveis e 0 segundo para protocolos de transporte confiáveis.

É apresentado abaixo um resumo dos algoritmos de retransmissão para o caso de mensagens INVITE, BYE, ACK, 100 Trying/180 Ringing (classe 1xx) e 200 OK.

- INVITE – Em caso de perda, as retransmissões obedecem ao algoritmo definido pelo *Timer* A. O *time out* ocorre quando o *Timer* B expira.
- BYE – Em caso de perda, as retransmissões obedecem ao algoritmo definido pelo *Timer* E. O *time out* ocorre quando o *Timer* F expira.
- ACK – Em caso de perda, o ACK não é retransmitido. A resposta 200 OK é que será retransmitida.
- 100 Trying/180 Ringing (classe 1xx) – As respostas de classe 1xx não são transmitidas confiavelmente pelo protocolo SIP, já que as mesmas são mensagens meramente informativas. No entanto, no caso de uso de um protocolo de transporte confiável, como o TCP, caso uma mensagem 1xx seja perdida, haverá retransmissão da mesma.
- 200 OK – Em caso de perda, quando a resposta 200 OK for em resposta a uma requisição BYE, ela não será retransmitida. No

caso de resposta ao INVITE, a perda da resposta 200 OK obedecerá a um algoritmo igual ao do *Timer* G, inicia em T1, depois passa para MIN(2*T1,T2) e assim por diante até atingir T2, quando esse tempo é fixado. Este *timer* não possui nome e é utilizado independente do protocolo de transporte. Isto, porque, como o 200 OK é retransmitido *end-to-end*, podem existir *hops* em que o protocolo de transporte pode ser não confiável. Logo, para garantir a confiabilidade em todos os *hops*, o 200 OK é retransmitido periodicamente mesmo se o transporte for confiável.

No caso do SIP sobre TCP, uma mensagem dá *time out* quando não há recebimento de um TCP ACK, ou seja, para cada mensagem enviada, há uma confirmação, aumentando assim o tráfego de mensagens na rede. Os valores do *time out* são determinados pela medição do RTT pelo TCP. No entanto, o valor padrão de *time out* usado no TCP para o estabelecimento inicial da conexão depende da implementação, podendo chegar a seis segundos [3]. Assim, quando o TCP for utilizado para VoIP, há a necessidade de regular estes valores. Ainda no TCP, o valor de *time out* aumenta exponencialmente, tipicamente por um fator de dois, cada vez que uma dada mensagem é retransmitida [3].

Um dos problemas do TCP é o HOL (*Head of Line*) *blocking* [56]. O problema acontece ao utilizar a mesma conexão TCP para transmitir mensagens de sinalização associadas a múltiplas sessões entre dois servidores. Um cliente TCP envia duas mensagens de sinalização através de uma conexão TCP: a primeira mensagem (mensagem A) corresponde a uma sessão entre dois UAs e a segunda mensagem (mensagem B) corresponde a outra sessão entre outros dois UAs. Se a mensagem B chegar com sucesso ao par da conexão TCP e a mensagem A se perder, o TCP não encaminhará a mensagem B a aplicação até a mensagem A ser retransmitida e finalmente recebida. Logo, a sessão em que a mensagem B faz parte é afetada pela perda de uma mensagem que pertence a outra sessão. A utilização de um único TCP *socket* (endereço IP e porta) para transmissão de mensagens associadas a múltiplas sessões (ou chamadas) para um mesmo destino (endereço IP e porta) é chamado de TCP *persistent connection* (ou conexão TCP persistente).