Integração do protocolo SIP na Media Framework da plataforma Android

André Filipe Ferreira Araújo Barbosa

Dissertação na área de informática com vista na obtenção do grau de mestre em informática sobre a orientação do professor António Duarte Costa e professor António Nestor Ribeiro.

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Setembro, 2011

Agradecimentos

Esta dissertação assinala a conclusão de mais um ciclo da minha vida académica. Este etapa importante da minha vida, não teria sido possível sem a ajuda e apoio de algumas pessoas.

Em primeiro lugar, quero agradecer aos meus pais, Manuela e Filipe, e à minha irmã, Joana, pelo apoio e compreensão que me proporcionaram e por permitirem que eu chegasse até aqui.

Gostaria de agradecer o apoio, disponibilidade e orientação académica do meu orientador, Professor Doutor António Duarte Costa e o meu co-orientador, Professor Doutor António Manuel Nestor Ribeiro. Gostava ainda de agradecer a compreensão e disponibilidade da Eng. Telma Mota e o Eng. Pedro Neves da PT Inovação.

Agradeço também aos meus amigos José Xavier, Tiago Ferreira, Paulo Ferreira, Rui Silva, Ricardo Silva, Alberto Gomes, Luís Claro e João Lopez, pelo companheirismo e apoio durante a realização deste trabalho.

Não podia deixar de agradecer também aos meus colegas e amigos do Instituto de Telecomunicações - Pólo de Aveiro e da PT Inovação, em especial ao Márcio e à Andreia, pelo acolhimento que me proporcionaram.

Por último, quero agradecer à minha namorada, Joana, por todo o apoio e compreensão que me proporcionou ao longo de todos os momentos de alegria, tristeza, frustração e ansiedade durante a realização deste trabalho.

Abstract

The concept that the personal computer is the "box" that is in the home desk is changing. More and more it is being tried to transfer all the functions that we have in this device, to more portable devices. Currently the mobile devices, such as Pocket PCs and Smartphones, are an example of this tendency, making the simple concept of cell phone into something more, which tends to supply the same functions that our typical computer supplies. With the strong investment of the manufacturers in the evolution of the mobile devices, the mobile operative systems have to guarantee the following of this evolution and to take advantage of the devices potentialities.

The applications of voice over IP (VoIP), are one of the fields in constant exploration by the companies that create applications for the mobile devices. The reduced cost of this service, comparing with the voice service supplied by the mobile service providers, is an incentive for the users. With the evolution of these communication services about IP, rises more and more, the interest in the evolution of this voice service to voice and video. Unfortunately, the new Android platform, even though it allows to do calls of voice over IP, it does not allow the implementation of video due to limitations of its API multimedia. In this way, this work focus in studying the internal structure of the Media Framework of the Android platform and integrate the SIP protocol in its structure, in order to allow the video reproduction during one SIP session.

As a result of the developed work, it was introduced and implemented one SIP component in the OpenCore framework. This component gives to the Media Framework of the Android the opportunity of controlling directly one SIP session, allowing not only the audio reproduction but also the video. Through the tests done to the developed SIP component, it was possible to conclude the correct functioning and identify some problems that can occur in future scenes with devices of different capacities, to which it was proposed one possible solution to surpass these problems.

Resumo

O conceito de que o computador pessoal é a "caixa" que está na secretária de casa está a mudar. Cada vez mais se tenta transferir todas as funcionalidade que se têm nesse dispositivo, para dispositivos mais portáteis. Actualmente os dispositivos móveis, como *Pocket PCs* e *Smartphones*, são um exemplo dessa tendência, tornando o simples conceito de telemóvel em algo mais, que tende em fornecer as mesmas funcionalidades que o nosso típico computador fornece. Com o forte investimento dos fabricantes na evolução dos dispositivos móveis, os sistemas operativos móveis têm de garantir o acompanhamento desta evolução e tirar proveito das potencialidades dos dispositivos.

As aplicações de voz sobre IP (VoIP), são uma das áreas em constante exploração por parte das empresas que criam aplicações para dispositivos móveis. O reduzido custo deste serviço, em comparação com o serviço de voz fornecido pelos operadores móveis, é uma motivação para os utilizadores. Com a evolução deste serviços de comunicação sobre IP, surge cada vez mais, o interesse em evoluir este serviço de voz para voz e vídeo. Infelizmente, a nova plataforma Android, embora permita efectuar chamadas de voz sobre IP, não permite a implementação de vídeo devido a limitações da sua API multimédia. Neste sentido, este trabalho foca-se em estudar a estrutura interna da Media Framework da plataforma Android e integrar o protocolo SIP na sua estrutura, de forma a possibilitar a reprodução de vídeo durante uma sessão SIP.

Como resultado do trabalho desenvolvido, foi apresentada e implementada uma componente SIP na framework OpenCore. Esta componente dá à Media Framework do Android a possibilidade de controlar directamente uma sessão SIP, possibilitando não só a reprodução de áudio como também do vídeo. Através dos testes feitos à componente SIP desenvolvida, foi possível concluir o seu correcto funcionamento e identificar alguns problemas que podem ocorrer em futuros cenários com dispositivos de diferentes capacidades, ao qual foi proposta uma possível solução para superar esses problemas.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	Application programming interface
BOS	Begin Of Stream
DRM	Digital Rights Management
DNS	Domain Name System
EOS	End Of Stream
GOP	Group Of Pictures
HTTP	Hypertext Transfer Protocol
IMS	IP Multimedia Subsystem
IDE	Integrated Development Environment
IP	Internet Protocol
JDK	Java Development Kit
MOD	Media on Demand
NDK	Native Development Kit
OSCL	Operating System Compatibility Library
PDA	Personal Digital Assistants
QoS	Quality of service
RTP	Real Time Protocol
RTSP	Real Time Streaming Protocol
SIP	Session Initiation Protocol

SDK	Software Development Kit
SDP	Session Description Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol

Capítulo 1

Introdução

Nos últimos anos, a área das telecomunicações tem sofrido um enorme desenvolvimento tecnológico. Constantemente, são lançados novos dispositivos móveis, de diversos fabricantes, com novas funcionalidades. Acompanhando este desenvolvimento, as redes de comunicação também têm sofrido melhoramentos, aumentado a sua velocidade e a sua potência de cobertura. Esta constante evolução potencializa a criação de novos cenários, os quais eram impossíveis de implementar devido às limitações tecnológicas. A transmissão de conteúdos multimédia entre dispositivos móveis, com qualidade aceitável, é um exemplo dessas limitações. No entanto, com as actuais potencialidades das redes de comunicação, essas limitações começam a ser ultrapassadas.

Actualmente, a plataforma Android disponibiliza como protocolo de streaming em tempo real o *Real Time Streaming Protocol* (RTSP)[8]. A sua capacidade de manipular uma *stream* de *media* (fazendo *pause*, *play*, *stop*, etc.), torna este protocolo uma excelente opção para a sua utilização quando pretendemos assistir a uma *stream*. No entanto, com o avanço tecnológico, começam a aparecer outras necessidades além de assistir à reprodução da *media* enviado por um servidor. Tem de existir flexibilidade nos serviços: a possibilidade de partilhar uma *stream* com um amigo ou um grupo de amigos não pode ser posta de parte, sendo necessário que exista interacção entre os dispositivos móveis, não ficando ligados à arquitectura cliente-servidor. O Session Initiation Protocol (SIP)[9], como protocolo de sessão, juntamente com o Session Description Protocol (SDP)[10], tem potencialidades para fornecer este tipo de serviço, disponibilizando ainda inúmeras novas funcionalidades para as quais o RTSP está limitado.

1.1 Motivação e Objectivos

O protocolo SIP, desde a sua definição na RFC 3261 [9], está ligado ao mundo da telefonia sobre redes IP. No entanto, com a evolução das redes de comunicação, dos dispositivos móveis e de todo o mundo multimédia, as suas funcionalidades têm potencializado o aparecimento de novas ideias e novos cenários no mundo da multimédia. A distribuição de conteúdos multimédia em tempo real por um conjunto de utilizadores, é uma das motivações à integração do protocolo SIP nos dispositivos móveis. As funcionalidades de localização para contacto e disponibilidade dos utilizadores, tornam o protocolo SIP uma mais-valia para as aplicações de distribuição de conteúdos multimédia. É importante disponibilizar o protocolo SIP na plataforma Android, não só para o enriquecimento multimédia da plataforma mas também para incentivar o aparecimento de novas aplicações capazes de cativar os utilizadores desta plataforma.

A necessidade de utilizar o protocolo SIP e o facto da plataforma Android não disponibilizar suporte para este protocolo, obriga a que as aplicações implementem a sua própria *SIP stack*. Normalmente, são aplicações multimédia onde existe a necessidade de manipular *media*: codificação, descodificação, reprodução, etc. Sendo o suporte para este tipo de operações, na plataforma Android, operações de baixo nível. Os desenvolvedores de software são obrigados a implementar os seus algoritmos de descodificação e codificação. No entanto, não é possível integrar estes algoritmos com as bibliotecas optimizadas da *framework* Opencore da plataforma Android, comprometendo as aplicações à utilização da API disponibilizada pela framework OpenCore. Infelizmente, a API do Android actualmente apenas dá suporte para a reprodução de áudio, ficando a reprodução do vídeo limitada à utilização de um protocolo de *stream*, como por exemplo o RTSP ou a reprodução a partir de um ficheiro.

O objectivo principal desta dissertação é integração do protocolo SIP na Media Framework do Android, habilitando a utilização das bibliotecas de descodificação e codificação da *framework* OpenCore para os elementos multimédia da aplicação (vídeo e áudio). A integração desta nova funcionalidade deverá ser transparente para a actual API da Media Framework do Android, adicionando a possibilidade da utilização desta nova funcionalidade nas actuais aplicações de *media*. Este trabalho está a ser desenvolvido em conjunto com a PT inovação, o qual se integra no projecto europeu C-Cast[1]. O domínio da tecnologia *IP Multimedia Subsystem* (IMS)[11], SIP e o *know-how* de desenvolvimento de aplicações Android, são um dos actuais objectivos estratégicos da PT Inovação, pelo que têm todo interesse em que a plataforma Android suporte não só áudio como também vídeo nas sessões SIP.

[1] http://www.ict-ccast.eu/

1.2 Síntese das principais Contribuições

A principal contribuição resultante do trabalho desenvolvido, traduz-se na integração do protocolo SIP na Media Framework do Android, a qual possibilita a reprodução de áudio e vídeo durante uma sessão SIP. Como fruto da definição da arquitectura para a integração do protocolo SIP na *framework* OpenCore da plataforma Android, foi publicado o artigo "Integration of SIP protocol in Android Media Framework", apresentado na conferência internacional EUROCON 2011 - "IEEE International Conference on Computer as a Tool".

1.3 Estrutura da dissertação

A estrutura da dissertação está organizada em seis capítulos: Introdução, Estado de arte, Requisitos e especificação, Implementação, Avaliação dos resultados e conclusões. Segue-se a descrição de cada capítulo:

- Introdução: estes capítulo faz o enquadramento e a contextualização do trabalho a desenvolver. É apresentada a motivação e os objectivos pretendidos, terminando com as principais contribuições e a descrição da estrutura da dissertação.

- Estado de Arte: apresenta as tecnologias base para o trabalho a desenvolver. É feita uma abordagem aos vários tipos de *stream* e aos protocolos utilizados durante a sessão (protocolos de sessão, transporte, descrição). Posteriormente, é feita uma descrição da plataforma Android, onde é dada uma descrição mais detalhada das suas componentes multimédia. Por fim, são apresentados alguns projectos relacionados com o trabalho a desenvolver, expondo as suas falhas e motivando o trabalho a desenvolver ao longo desta dissertação.

- Requisitos e especificação: apresenta os requisitos a ser cumpridos no trabalho a desenvolver e um estudo das principais componentes da *framework* Opencore, o qual servirá de base para a integração do protocolo SIP. Com base no estudo da *framework* Opencore, é apresentada a arquitectura proposta para a integração do protocolo SIP na *framework* OpenCore da plataforma Android.

- Implementação: o capítulo de implementação descreve o trabalho prático necessário para a implementação e integração da arquitectura definida no capítulo de Requisitos e Especificação. Inicialmente, é abordada a integração da *SIP stack* eXosip [12] na plataforma

Android, prosseguindo com a implementação das componentes definidas na arquitectura. Implementadas as componentes, é descrito o seu funcionamento interno e o funcionamento com as restantes componentes da *framework* Opencore.

- Avaliação dos resultados: apresenta os resultados obtidos durante a reprodução de uma *stream* de áudio e vídeo, utilizando a biblioteca SIP desenvolvida. Com base nos testes obtidos, é feita uma comparação com o protocolo RTSP, no qual foi utilizada a mesma *stream* nos seus testes. Visualizando os resultados obtidos e feita a comparação das bibliotecas, verificou-se alguns problemas que podem ocorrer entre dispositivos de diferentes capacidades computacionais. Perante a identificação desses problemas, é apresentada uma possível solução.

- Conclusões: No último capítulo, inicialmente é feita uma análise a todos os capítulos, são apresentadas as principais conclusões obtidas e as contribuições do trabalho realizado. Por fim, são apresentadas algumas propostas para continuação e melhoria do trabalho desenvolvido.

Capítulo 2

Estado da Arte

Ao longo deste capítulo, são abordadas as tecnologias envolvidas no trabalho a desenvolver nesta dissertação. Inicialmente é feita uma introdução ao *streaming* multimédia, onde é apresentado o seu conceito, o funcionamento das suas variantes e os protocolos utilizados em *streaming* com maior relevância para o trabalho a desenvolver. Posteriormente, é apresentada a nova plataforma Android, onde é feita uma introdução à Media Framework da plataforma Android e à *framework* OpenCore. Por fim, são apresentados alguns trabalhos (na área da multimédia) motivantes para o trabalho a desenvolver.

2.1 Streaming Multimédia

Com a actual popularidade das páginas web de partilha de vídeos e músicas, cada vez mais se fala em *streaming* multimédia. *Streaming* é uma técnica de distribuição de informação multimédia, fragmentada em pacotes por uma rede. Esta consiste na transmissão de dados, áudio e vídeo, de um servidor para um cliente.

Ao contrário de um *download* tradicional, numa sessão de *streaming*, os clientes não tem de esperar pelo final do *download* da *media* para a poder reproduzir. A sua reprodução é feita à *medida* que os fragmentos de *media* são recebidos. Durante a *streaming*, o cliente preserva a relação temporal da fonte, sendo necessário a criação de um *buffer* de *media*. O tamanho deste *buffer* depende da velocidade de transmissão dos dados, evitando pausas durante a sua reprodução.

Com o crescimento da Internet, a tecnologia de *streaming* é cada vez mais utilizada. A

baixa largura de banda de acesso à internet de alguns clientes e a necessidade de assistir à *media* com a maior brevidade possível após a sua requisição, tornam esta técnica uma mais valia à sua integração nos conteúdos multimédia na internet. Actualmente, são utilizados três tipos de *streaming*: *streaming* tradicional, *download* progressivo e *streaming* adaptativo [1].

2.1.1 *Streaming* tradicional

Uma das principais características que distingue a *streaming* tradicional dos restantes tipos de *streaming*, é o facto de possuir sessões. No momento em que é estabelecida a ligação para iniciar uma *streaming*, é iniciada uma sessão. A sessão termina quando é terminada a transmissão multimédia. A transmissão pode terminar por duas causas: o conteúdo a transmitir foi completamente transmitido ou por interrupção do cliente. A interrupção por parte do cliente é conseguida através da passagem de comandos ao servidor (ex: *pause* e *stop*). O facto de existir uma sessão durante a transmissão da *media*, favorece funcionalmente a capacidade de interacção do cliente com o servidor [13].

Uma outra característica da *streaming* tradicional é que, ao contrário dos restantes tipos de *streaming* em que existem vários segmentos a transmitir para cada conteúdo multimédia, no *streaming* tradicional existe apenas uma transmissão contínua do fluxo de dados [14]. A informação a ser transmitida é codificada em pacotes de pequeno comprimento, sendo posteriormente transmitido utilizando um protocolo de transporte. Normalmente, estes protocolos de transporte utilizam portas aleatórias para a transmissão do fluxo de dados. Esta situação em algumas redes é um problema, onde os *proxies* e as *firewalls* bloqueiam esse tipo de tráfego.

Na imagem 2.1 é apresentado um diagrama ilustrativo de uma transmissão de uma streaming tradicional.

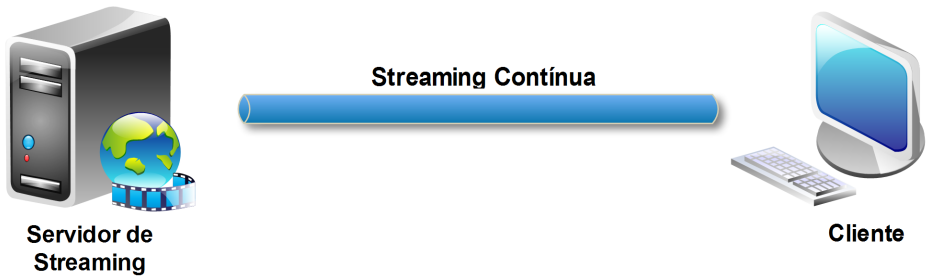

Figura 2.1: *Streaming* tradicional entre servidor de *streaming* e cliente [1]

2.1.2 *Download* Progressivo

O Download Progressivo consiste num método híbrido entre *download* de conteúdos e *streaming*. O *download* da *media* a reproduzir é feito a partir de um servidor HTTP, o qual envia para o cliente a *media* dividida em vários blocos através do protocolo HTTP. Por sua vez, o cliente recebe os dados, cria um *buffer* da *media* recebida e inicia a sua reprodução [15].

Ao contrário dos servidores de *streaming* tradicionais, os servidores Web HTTP não param de transmitir os dados até que o *download* esteja concluído. Este facto pode ser visto como um benefício ou como uma desvantagem. Num cenário onde o utilizador coloca a reprodução em pausa, como descrito anteriormente, o servidor Web HTTP vai continuar a enviar o conteúdo e o cliente vai continuar a fazer *buffer* da *media* recebida. Este cenário pode ser vantajoso para o utilizador que, posteriormente, vai continuar a ver a restante *media*. Em contrapartida, vai ser desvantajoso para o utilizador que não vai continuar a assistir à *media*, uma vez que gastou tráfego a fazer o *download* da restante *media*, da qual não vai tirar partido.

Uma outra diferença do Download Progressivo em relação à *streaming* tradicional, é o facto da *media* ser transportada pelo protocolo HTTP, o que resolve o problema das *firewalls* apresentado anteriormente. Na imagem 2.2 é apresentado um diagrama ilustrativo de uma transmissão de Download Progressivo.

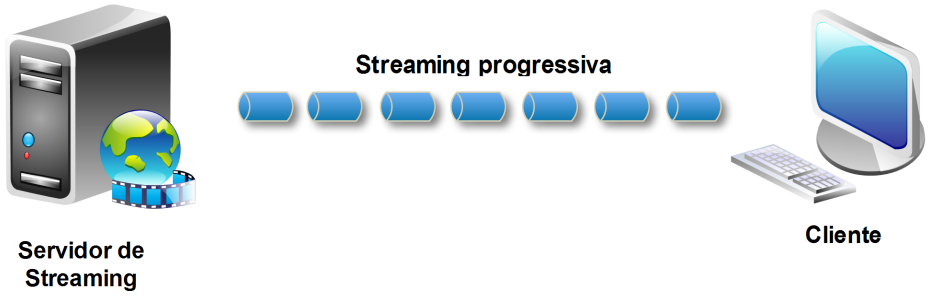

Figura 2.2: *Download* progressivo entre servidor de *streaming* e cliente [1]

2.1.3 *Streaming* Adaptativo

O Streaming Adaptativo consiste num método de funcionamento semelhante ao *streaming* tradicional, mas baseia-se no método de transmissão de *download* progressivo. Este tipo de *streaming* utiliza o protocolo HTTP para transporte da *media* e, em vez de transmitir apenas um único fluxo

de dados completo, transmite muitos fragmentos, os quais, quando reproduzidos ordenadamente, formam a *media* completa [16].

No Streaming Adaptativo analogamente aos restantes tipos de *streaming*, a *media* é fragmentada em blocos. No entanto, para *stream* de vídeo, o tamanho do fragmento está directamente relacionado ao tamanho do GOP (Group of Pictures), isto é, cada bloco do fragmento contém a informação de um *I-frame* até a próxima *I-frame* [17] [18]. Um *I-Frame*, é uma imagem completa, isto é, a sua representação não depende de outras imagens. Um GOP é composto por um *I-Frame* inicial e todo o conjunto de *frames* dependentes dela, as quais geram o movimento do vídeo ao longo da GOP. Desta forma, é possível reproduzir cada fragmento recebido sem a necessidade de criar um *buffer* de *media* para a sua descodificação, uma vez que não existem dependências entre os fragmentos recebidos.

Na imagem 2.3 é apresentado um diagrama ilustrativo de uma transmissão de Streaming Adaptativo.

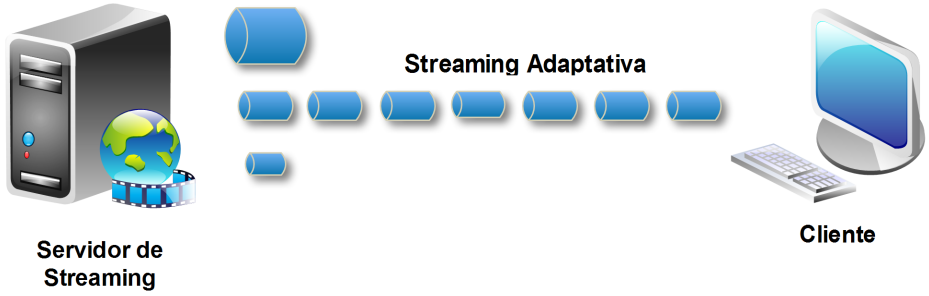

Figura 2.3: *Streaming* Adaptativo entre servidor de *streaming* e cliente [1]

2.2 Protocolos de Comunicação

Para implementar um sistema de *streaming* multimédia, é necessário utilizar protocolos para controlar as sessões. Além dos protocolos de controlo de sessão, são necessários protocolos de transmissão e de descrição da *media* a transmitir. Ao longo desta subsecção, são apresentados os protocolos RTSP[8] e SIP[9] para controlo de sessões, RTP[19] para transporte e SDP[10] para descrição.

2.2.1 Real Time Streaming Protocol (RTSP)

O *Real Time Streaming Protocol* (RTSP)[8] é um protocolo ao nível aplicacional, desenhado para estabelecer e controlar sessões multimédia entre dois pontos (Cliente – servidor) [8]. O protocolo RTSP não efectua a entrega da *stream*, sendo apenas responsável pelo controlo. Por outras palavras, o RTSP funciona como um controlo remoto, através do qual o cliente pode manipular um servidor de *streaming* multimédia.

Normalmente, a *stream* controlada pelo RTSP utiliza o protocolo RTP como protocolo de transporte [19]. No entanto, o controlo da sessão feito pelo RTSP é independente do protocolo utilizado no transporte, podendo ser utilizado outro protocolo para o transporte da *stream* de *media* [20] [21].

Além do protocolo de transporte da *stream*, o RTSP utiliza um outro protocolo para descrever a *media* a transmitir durante a sessão. O protocolo utilizado para descrever a *media* é o SDP, sendo este enviado no início da sessão juntamente com o comando DESCRIBE [10]. O envio, no início da sessão, do comando DESCRIBE e dos dados do SDP, tem como objectivo informar o cliente do tipo de *media* a receber, para que este seja capaz de efectuar a reprodução dos dados durante a sua recepção [22].

Na figura 2.4 está representada a troca de mensagens durante uma sessão RTSP.

2.2.2 Real-time Transport Protocolo (RTP)

O *Real-time Transport Protocol* (RTP) é o protocolo padrão para o transporte de áudio e vídeo em tempo real pela internet [19]. Actua ao nível aplicacional e permite a transmissão de dados ponto-a-ponto ou multi-ponto utilizando o conceito de *multicast*. Para transmissões multi-ponto, é necessário que a rede suporte *multicast*. A transmissão dos pacotes RTP pela rede é feita, normalmente, utilizando o protocolo UDP [23]. No entanto, pode ser utilizado outro protocolo, dependendo da rede onde será efectuado o transporte [24].

Em resumo, o protocolo RTP é utilizado simplesmente para a transmissão da *media*, não disponibilizando mecanismos para fornecer garantias de entrega dos pacotes ou para garantir a qualidade de serviço da transmissão. O RTP pode ser utilizado em servidores de *streams* (Media on Demand) ou em serviços de telefonia e vídeo-chamada (serviços interactivos).

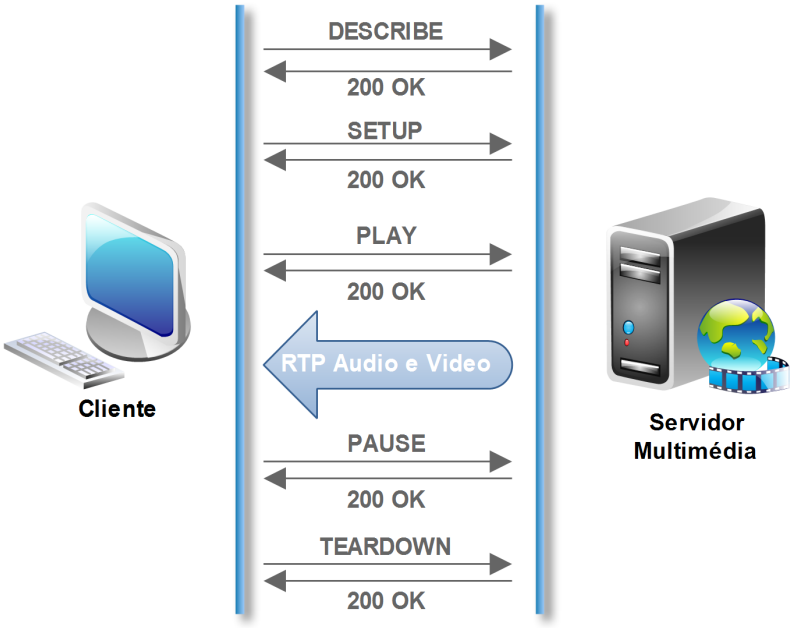

Figura 2.4: Troca de mensagens durante uma sessão RTSP

2.2.3 Session Description Protocol (SDP)

O *Session Description Protocol* (SDP)[10] é um formato utilizado para descrever sessões multimédia [10]. Este protocolo é utilizado em convites de sessões, anúncio de novas sessões e outras formas de inicialização de sessões multimédia.

Uma mensagem SDP é composta por nome da sessão, duração da sessão, informação sobre a *media* a transmitir durante a sessão e a informação necessário para a transmissão da *media*. Como se pode observar pela composição de uma mensagem SDP, por si só o protocolo SDP não é capaz de se transmitir pela rede, sendo necessário o seu encapsulamento em outro protocolo para o seu transporte. Normalmente, o SDP é encapsulado em protocolos de negociação de sessões multimédia, como por exemplo RTSP, SIP, etc. Na figura 2.5 é apresentado um exemplo de uma mensagem SDP.

Em resumo, o protocolo SDP foi projectado para descrever as informações sobre a *media* a transmitir numa sessão multimédia. O seu encapsulamento nos protocolos de controlo de sessões, tem como intuito informar os intervenientes da sessão das propriedades da *media* a ser transmitida [25].

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.atlanta.example.com
s=
c=IN IP4 host.atlanta.example.com
t=0 0
m=audio 49170 RTP/AVP 0 8 97
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 iLBC/8000
m=video 51372 RTP/AVP 31 32
a=rtpmap:31 H261/90000
a=rtpmap:32 MPV/90000
```

Figura 2.5: Exemplo de uma mensagem SDP [2]

2.2.4 Session Initiation Protocol (SIP)

O *Session Initiation Protocol* (SIP) é um protocolo ao nível aplicacional utilizado para a manipulação de sessões multimédia [9]. Através do protocolo SIP, é possível estabelecer, modificar e finalizar sessões multimédia. Além de estabelecer ligações ponto-a-ponto, o protocolo SIP fornece a possibilidade de convidar novos participantes para uma sessão já estabelecida, criando uma sessão multi-ponto [26]. A arquitectura do protocolo SIP suporta transparentemente o mapeamento de nomes e serviços de redireccionamento, o que permite a utilizadores móveis manter um identificador único e independente da sua identificação na rede [27]. Na figura 2.6 está representada a troca de mensagens durante uma sessão SIP.

Como podemos observar pela Figura 2.6, o SIP disponibiliza comandos para iniciar e manipular uma sessão multimédia. No *standard* SIP, estão definidos seis comandos básicos: INVITE, ACK, BYE, CANCEL, REGISTER e OPTIONS.

O comando INVITE é utilizado para convidar um utilizador para uma sessão. Se a sessão a estabelecer for uma sessão multimédia, é enviado juntamente ao pacote SIP do INVITE, o SDP com a informação da transmissão multimédia a transmitir.

O comando ACK é enviado após a negociação da sessão. Num cenário de uma transmissão de multimédia, a transmissão da *media* é iniciada logo após a recepção do ACK.

Os comandos BYE e CANCEL são utilizados para terminar a sessão. No entanto, o comando BYE é utilizado para terminar uma sessão já estabelecida, enquanto que o comando CANCEL é utilizado para terminar uma sessão ainda em negociação.

Finalmente, os comandos REGISTER e OPTIONS são utilizados para comunicar com o

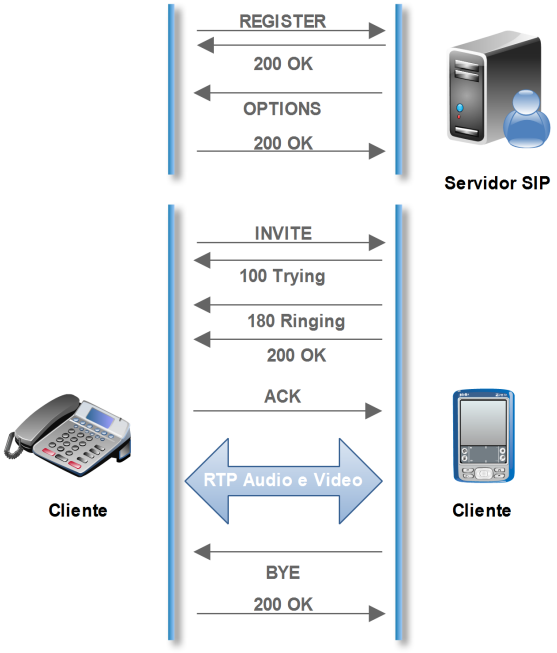

Figura 2.6: Troca de mensagens durante uma sessão SIP

servidor SIP. O comando REGISTER é utilizado para registar o cliente num servidor SIP, ficando esse utilizador associado a esse servidor. Por sua vez, o comando OPTIONS é utilizado para o cliente informar o servidor das suas capacidades multimédia (para o estabelecimento de futuras sessões) [28].

2.3 Plataforma Android

A plataforma Android[3], actualmente da Open Handset Alliance[29], é um conjunto completo de software código aberto (sistema operativo, *middleware* e aplicações base) projectado para correr em dispositivos de baixa capacidade (Telemóveis, PDAs, *Tablet PCs*). Além do conjunto base de software que torna o dispositivo funcional, a plataforma Android disponibiliza uma API em JAVA (Android SDK)[30] de apoio ao desenvolvimento de aplicações para a plataforma Android.

Software Development Kit (SDK)

O Software Development Kit (SDK[1]) do Android foi desenvolvido com o intuito de disponibilizar uma API em Java à qual os programadores podem utilizar para desenvolver as suas aplicações para Android. O SDK utiliza as bibliotecas do sistema, disponibilizando diversas funcionalidades aos programadores, entre as quais, a possibilidade de o programador utilizar algumas funcionalidades do hardware do dispositivo, sendo a API independente do tipo de hardware utilizado [31]. As aplicações desenvolvidas sobre esta API, são escritas em Java e correm na máquina virtual (Dalvik) do Android [32].

Em muitas situações é necessário a escrita de funções que exigem uma maior capacidade do processamento do dispositivo, onde o facto de essa função estar a correr sobre a máquina virtual pode ser uma desvantagem para o desempenho da aplicação. Para estas situações, a plataforma Android disponibiliza o Native Development Kit (NDK)[33].

Native Development Kit (NDK)

O Android NDK[2] é um conjunto de ferramentas que permite incorporar bibliotecas ou código nativo escrito em C/C++ nas aplicações desenvolvidas em Java, esta funcionalidade pode ser utilizada para reutilização de código ou para o aumento do desempenho da aplicação a desenvolver. Através do NDK, é possível escrever código em C/C++, compilar esse código para a plataforma Android com o compilador disponibilizado no NDK, e evocar essa função utilizando o Java Native Interface (JNI) na aplicação Java [34].

2.3.1 Arquitectura da plataforma Android

A Figura 2.7 apresenta a arquitectura da plataforma Android. Arquitecturalmente, a plataforma Android está dividida em quatro camadas, das quais fazem parte cinco componentes: Kernel, bibliotecas do sistema, Android Runtime, Framework de Desenvolvimento e aplicações base [35].

Na camada inferior da Arquitectura do Android encontramos a componente Kernel do sistema Android. O Kernel do sistema Android é baseado em Linux para o sistema de serviços (segurança, gestão de processos, etc). De uma forma genérica, esta camada funciona como uma

[1] http://developer.android.com/sdk/index.html
[2] http://developer.android.com/sdk/ndk/index.html

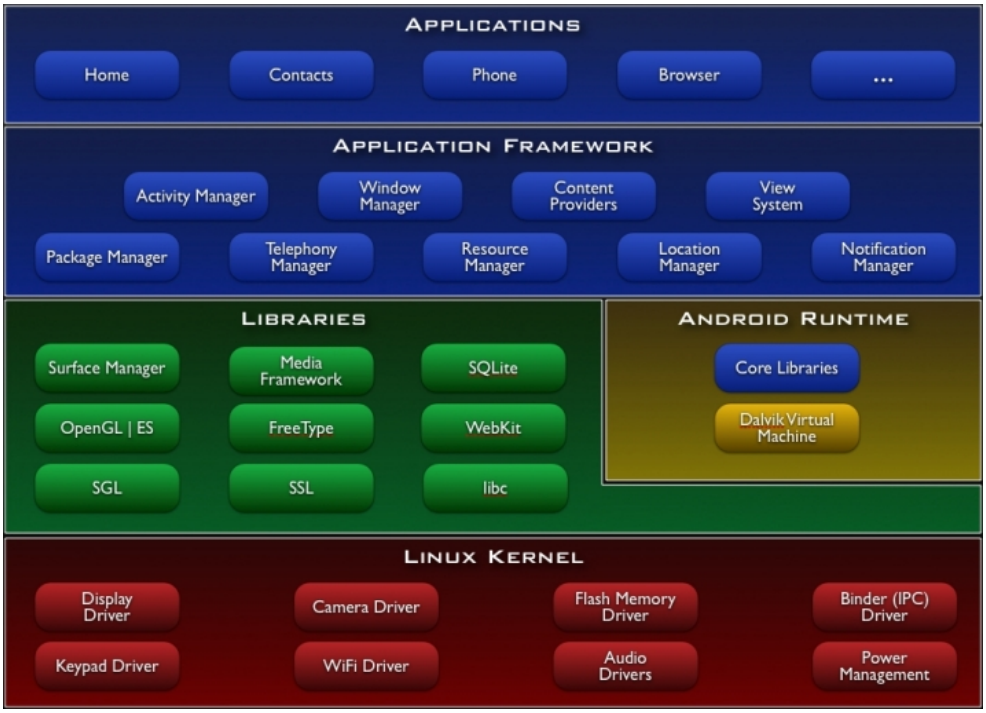

Figura 2.7: Arquitectura do Android[3]

abstracção do hardware para o resto do sistema.

Na camada superior ao Kernel encontramos duas componentes: as bibliotecas do sistema e o Android Runtime (serviços do sistema). A componente de bibliotecas do sistema dispõe de um conjunto de bibliotecas escritas em C/C++ que fornecem funcionalidades às aplicações da plataforma. O Surface Manager é um exemplo de uma biblioteca desta componente: é responsável por desenhar no ecrã a imagem correspondente às diferentes aplicações que estão a correr em processos diferentes e estão a obter actividade em momentos distintos, garantindo que é apresentada a respectiva imagem quando algo está visível e é alterado na aplicação ou aplicações visíveis. A componente Android Runtime disponibiliza à plataforma a máquina virtual Dalvik VM e as Core Libraries, que contêm um conjunto de bibliotecas e serviços do sistema. Os serviços desta componente são evocados através do Binder do sistema Android disponibilizado pelo Kernel.

Na camada superior às bibliotecas do sistema e ao Android Runtime, temos a componente de Framework de desenvolvimento. Esta componente disponibiliza uma API de desenvolvimento em JAVA independente do dispositivo. Através da API de desenvolvimento, o programador tem acesso a recursos do dispositivo sem que tenha conhecimento das bibliotecas do sistema ou do

hardware que a sua aplicação está a utilizar, facilitando, desta forma, o desenvolvimento das suas aplicações.

Finalmente, no topo da arquitectura da plataforma Android, encontramos a camada Aplicacional. Esta camada disponibiliza um conjunto de aplicações banais de um dispositivo móvel (contactos, SMS, calendário, *browser*, cliente *e-mail*, etc). As aplicações disponibilizadas nesta camada são escritas em JAVA e correm sobre a máquina virtual (Dalvik VM) do Android.

2.3.2 Android Media Framework

A Media Framework do Android é uma biblioteca que disponibiliza uma API para a manipulação de *media* (áudio e vídeo). A sua estrutura é composta por duas componentes: uma primeira componente que faz parte das bibliotecas do sistema (esta componente funciona como "*proxy*" para a evocação das funcionalidades disponíveis na API JAVA); uma segunda componente, situada na Android Runtime, que se trata do serviço onde é feita a manipulação da *media* [4]. Na Figura 2.8 está apresentado a arquitectura da Media Framework da plataforma Android.

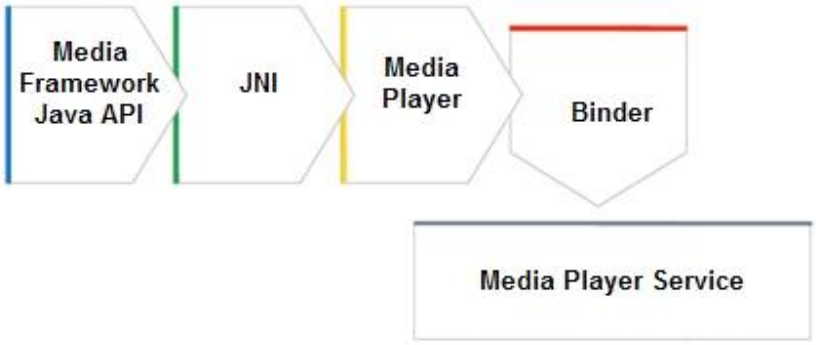

Figura 2.8: Invocação de um método da Media Framework API

Como podemos observar pela arquitectura da Figura 2.8, estas duas componentes comunicam entre si através do Binder do sistema Android fornecido pelo Kernel. Sempre que um método da API JAVA é evocado, a biblioteca Media Framework passa o respectivo comando ao Binder, e este, por sua vez, avisa o serviço (Media Player Service) que é necessário executar o comando.

O Media Player Service é um serviço do sistema Android. Por outras palavras, é um processo que arranca juntamente com o sistema Android e fica à espera de receber comandos para executar acções. Este serviço está dividido em quatro componentes: Media Recorder Service, onde é feita a captura e manipulação de imagem da câmera (fotográfica e vídeo); as bibliotecas

MIDI e Vorbis, onde é manipulado todo o tipo de *media* MIDI e Vorbis, respectivamente; e, finalmente, a biblioteca OpenCore, onde são tratados todos os restantes tipos de *media* suportados pela plataforma Android. A Figura 2.9 apresenta a arquitectura do Media Player Service.

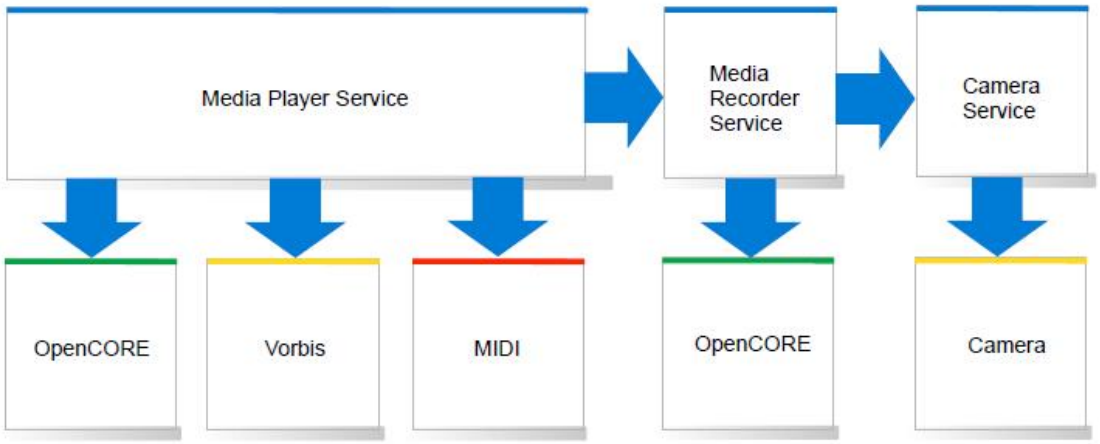

Figura 2.9: Arquitectura Media Player Service[4]

2.3.3 Framework OpenCore

A OpenCore é uma *framework* para o desenvolvimento de aplicações de *media*, disponibilizando funcionalidades de reprodução, *streaming* e gravação (imagem e vídeo). Além de disponibilizar alguns formatos na sua estrutura base, a sua estrutura é modular, o que a torna facilmente extensível, sendo possível adicionar novos *codecs*, formatos de arquivos e protocolos [36]. Na Figura 2.10 é apresentada a arquitectura de alto nível da *framework* OpenCore. Como podemos observar, a sua arquitectura esta dividida em cinco camadas, sendo estas compostas por seis componentes: Content Policy Manager, Multimedia Engines, Data Formats, Audio Codecs, Video Codecs e Android Interface.

A camada Content Policy Manager disponibiliza à Framework os métodos de acesso e controlo para a manipulação de conteúdo multimédia. Esta camada pode ainda ser utilizada para a integração de funcionalidades *Digital Rights Management* (DRM).

A camada Multimedia Engine faz o controlo da reprodução/gravação, recorrendo à invocação de funcionalidades das camadas inferiores. A reprodução/gravação de um elemento de media pode passar por diversos estados durante a sua execução (parado, inicializado, preparado, reprodução, gravação e erro). Para controlar os diferentes estados possíveis existe a necessi-

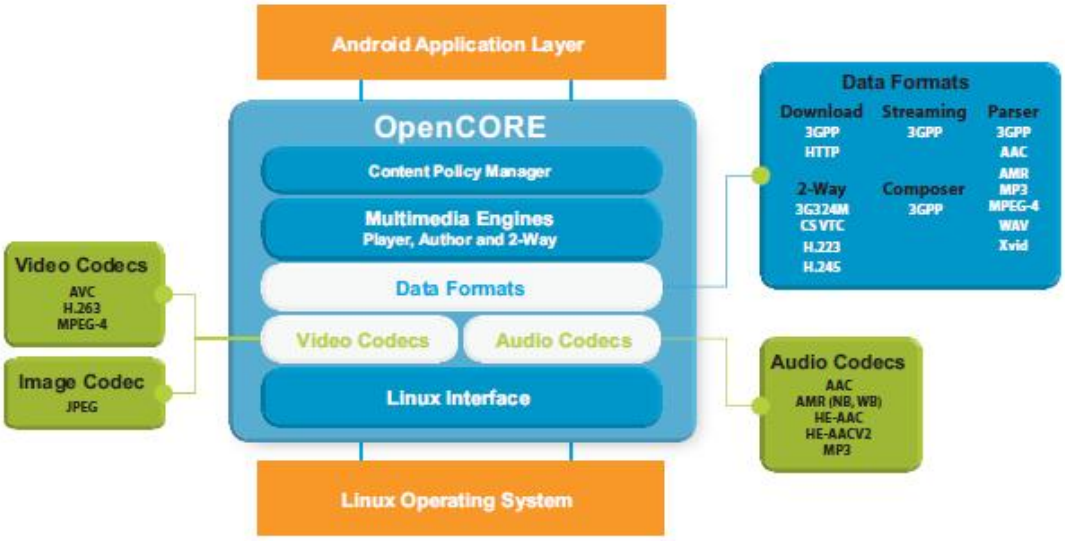

Figura 2.10: Arquitectura do OpenCORE[5]

dade de evocar diferentes funcionalidades em cada um dos estados. O Multimedia Engine cria um grafo apropriado para cada execução, facilitando o controlo da execução e a evocação de funcionalidades associadas a esse estado.

A camada Data Formats faz a leitura e escrita dos vários tipos de ficheiros de *media*, bem como a leitura nos vários protocolos de *streaming*. Nesta camada são suportados ficheiros do tipo .mp3, .mp4, .aac, .wav, .3gp e .amr. Quanto aos protocolos de *streaming*, são suportados RTSP e HTTP de Download Progressivo ou Playback Progressivo.

A camada de *codecs* é composta pelos componentes: Audio Codecs e Video Codecs. A componente de Audio Codecs inclui os *codecs*: GSM-AMR NR, WB, MP3, AAC, HE-AAC V1 e HE-AAC V2. Além dos *codecs* já incluídos, é disponibilizada uma interface para integrar outros formatos e suportar aceleração por hardware. A componente de Video Codecs inclui os *codecs*: H.264, H.263 e MPEG4. Analogamente ao Audio Codecs também disponibiliza uma interface para a integração de outros formatos e suportar aceleração por hardware.

A camada Android Interface disponibiliza a interface com as APIs do sistema operativo e a plataforma de serviços necessários (gestor de memoria, DNS lookup, rede, acesso a ficheiros, controlo de processos e *threads*, etc).

2.4 Projectos relevantes à integração do SIP na plataforma Android

O constante desenvolvimento da plataforma Android e o facto da plataforma ser código-aberto, propícia o crescente interesse em estudar e desenvolver novas funcionalidades para esta plataforma. Ao longo desta sessão serão abordados dois projectos com interesse no contexto do desenvolvimento desta dissertação: Project Context Casting (C-Cast) e Sipdroid.

2.4.1 Project Context Casting (C-Cast)

O Project Context Casting (C-Cast)[37] é um projecto Europeu que tem como objectivo a distribuição de conteúdo multimédia por dispositivos móveis. A distribuição do conteúdo multimédia é feito relativamente à localização, necessidades e interesses do utilizador.

O projecto C-Cast foca-se no desenvolvimento de duas áreas principais: a criação de contexto para os diferentes grupos que exigem a mesma informação ou serviço, e o desenvolvimento das tecnologias de *multicast* e transmissão de conteúdos em dispositivos móveis. No estudo destas duas áreas são abordadas três questões fundamentais: desenvolvimento de um gestor de grupos de contexto para os utilizadores; definição de uma framework para a recolha de informação do sensor, distribuição de informação de contexto e entrega da informação; desenvolvimento de mecanismos automáticos para a recolha, adaptação e fornecimento desses conteúdos.

No desenvolvimento e estudo da entrega da informação, existe a necessidade da utilização do protocolo SIP para a distribuição de conteúdo multimédia pelos utilizadores. Uma vez que os dispositivos móveis Android não dispõem de suporte para a transmissão de conteúdos de vídeo utilizando o protocolo SIP, é necessário desenvolver novas técnicas para ultrapassar este problema. A criação de um *proxy* que faz a tradução do protocolo SIP para o protocolo RTSP foi a solução utilizada neste projecto.

Como se pode observar pela figura 2.11, a solução apresentada resolve o problema. No entanto, minimiza as capacidades do cliente, aumenta a carga computacional e o grau de complexidade de desenvolvimento. Além de todas as desvantagens referidas, o cliente continua limitado à recepção de *stream*, sendo impossível enviar uma *stream* capturada no dispositivo a outro cliente em tempo real. Esta situação alerta para a necessidade da integração do protocolo SIP na plataforma Android.

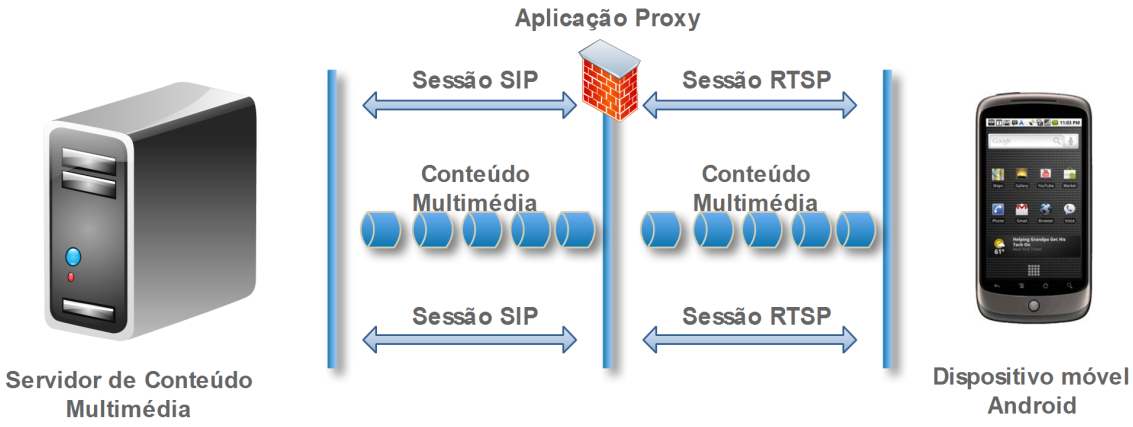

Figura 2.11: *Proxy* de distribuição de conteúdo multimédia

2.4.2 Sipdroid

O projecto Sipdroid[38] têm como objectivo o desenvolvimento de um cliente SIP para a plataforma Android. O projecto é desenvolvido na linguagem java, e utiliza a *SIP stack* MJSIP[39] para o estabelecimento das sessões SIP.

A versão inicial do Sipdroid oferece, como funcionalidades, a possibilidade de efectuar ou receber chamadas de voz utilizando o protocolo SIP para o estabelecimento da sessão. Para isso, basta que a aplicação se registe num servidor SIP (exemplo : Pbxes[3], Asterisk, etc). Na versão mais recente, actualmente versão 1.5, além de ser dada a possibilidade de utilizar novos *codecs* de áudio para o estabelecimento de chamadas de voz, é ainda possível o estabelecimento de chamadas de vídeo [40]. Nesta nova versão, foram adicionados novos *codecs* de áudio, que, ao contrário da versão inicial onde os *codecs* estavam escritos em java, estes são disponibilizados como bibliotecas nativas escritas em C++. Desta forma, a codificação torna-se mais eficiente, tanto ao nível computacional como ao nível energético.

Como anunciado, esta nova versão possibilita o estabelecimento de chamadas de Vídeo. No entanto, esta funcionalidade ainda é um pouco limitada, funcionado apenas com o servidor Pbxes. Esta limitação deve-se ao facto da plataforma Android estar limitada à utilização do protocolo RTSP para a reprodução de *streams* em tempo real.

Na página web do projecto Sipdroid não é disponibilizada qualquer informação sobre como

[3] www.pbxes.org/

são estabelecidas as chamadas de vídeo na aplicação. No entanto, se for feita uma captura do tráfego gerado durante a negociação da sessão RTSP (ao estabelecer a funcionalidade de vídeo), pode-se observar a captura dos pacotes de negociação da sessão RTSP para o vídeo a transmitir.

O facto de ser utilizadas transmissões RTSP de curta duração para simular uma transmissão em tempo real, faz com que a qualidade da experiência de utilizador seja consideravelmente fraca em relação ao que se obtém numa vulgar chamada de voz. Por outro lado, o facto de o cliente estar limitado à utilização de um determinado servidor que suporta essas funcionalidades extra, reduz a liberdade de escolha por parte do utilizador. Estas limitações motivam a integração nativa do protocolo SIP na plataforma Android, dando uma maior liberdade de escolha do servidor ao cliente e aumentando o desempenho do sistema de transmissão de vídeo actual, deixando de ser necessária a utilização de um segundo protocolo para o estabelecimento da sessão de vídeo.

2.5 Sumário

Como pudemos observar ao longo deste capítulo, o mundo multimédia nos dispositivos móveis é cada vez mais uma área em expansão. Perante as potencialidades dos novos dispositivos móveis, são constantemente idealizados novos cenários onde é necessário adaptar e evoluir os actuais sistemas.

No mundo da *stream* multimédia em tempo real, a plataforma Android ainda está pouco evoluída, disponibilizando apenas o protocolo RTSP para *stream* em tempo real e uma API ainda muito limitada para adicionar novas funcionalidades multimédia. Para combater estas deficiências da actual Media Framework do Android, são constantemente realizados trabalhos com o intuito de acrescentar novas funcionalidades. O trabalho a desenvolver ao longo desta dissertação tem como objectivo adicionar o protocolo SIP à actual Media Framework do Android (OpenCore), possibilitando o uso das bibliotecas de descodificação da Framework OpenCore do Android para o estabelecimento de sessões SIP. Esta integração possibilita às aplicações, utilizar os *codecs* de vídeo e áudio suportados pela plataforma Android durante sessões multimédia SIP, sendo que, o protocolo SIP passará a ser suportado, análogamente ao protocolo RTSP, pela Media Framework do Android.

Capítulo 3

Requisitos e Especificação

A descodificação/ codificação de grande parte dos formatos de vídeo existentes, requer um elevado nível de computação. A computação exigida nestas operações, pode ser em muitos casos demasiado elevada em comparação com a capacidade computacional do dispositivo móvel. Alguns dispositivos móveis utilizam aceleração por hardware para aumentar a sua capacidade de descodificação/ codificação do vídeo, no entanto, nem todos os dispositivos vêm equipados com este hardware.

A *framework* de *media* (OpenCore) utilizada na plataforma Android, disponibiliza uma API para lidar com estas duas situações. Mesmo que o dispositivo não disponha de hardware de apoio à descodificação, a *framework* OpenCore foi projectada para tirar o melhor desempenho possível na descodificação/codificação do vídeo, utilizando processadores de baixa capacidade (exemplo: ARM). Este processo deve-se à utilização de bibliotecas de alto desempenho de gestão de memória e algumas técnicas capazes de aumentar o desempenho desses processadores.

Em resumo, actualmente, a descodificação de vídeo com resultados aceitáveis nos actuais dispositivos móveis, deverá ser feita utilizando o hardware do dispositivo. Caso a descodificação por hardware não seja suportada pelo dispositivo, como alternativa, devem ser utilizadas bibliotecas optimizadas para tentar aumentar o desempenho do processador do dispositivo.

Ao longo deste capítulo, são expostos os requisitos do trabalho a desenvolver. É feito um estudo à Framework OpenCore e à componente que implementa o protocolo RTSP, o qual servirá como base na definição de uma arquitectura para a integração do protocolo SIP na Framework OpenCore. Por fim, é dada como conclusão uma abordagem rápida sobre a arquitectura proposta e os requisitos definidos.

3.1 Requisitos do trabalho a desenvolver

A integração do protocolo SIP na plataforma Android, poderá abrir um novo mundo de funcionalidades. A possibilidade de desenvolver novas funcionalidades, traz novas exigências, requerendo mais recursos e cada vez mais a evolução da plataforma. Actualmente, o protocolo SIP, já usado por aplicações na plataforma Android, é implementado directamente na aplicação. Este factor limita inúmeros cenários de partilha de sessões SIP entre diferentes aplicações SIP, deixando ainda as aplicações multimédia limitadas às funcionalidades de *media* disponibilizadas pela API da *framework* da plataforma Android.

Com a evolução das aplicações multimédia e com o aumento das exigências dos utilizadores, cada vez mais os desenvolvedores de aplicações aumentam a suas necessidades e exigências perante a plataforma. A utilização de vídeo nas aplicações multimédia SIP, é uma das exigências cada vez mais iminente tanto da parte das empresas que desenvolvem as aplicações como dos consumidores que anseiam por esta funcionalidade. Infelizmente, a actual plataforma Android apenas suporta reprodução de vídeo a partir de ficheiros gravados no dispositivo ou através de *stream* utilizando o protocolo RTSP ou HTTP progressivo.

É objectivo desta dissertação definir uma arquitectura para a integração do protocolo SIP na plataforma Android, possibilitando a utilização de vídeo nas sessões SIP. Além da possibilidade de utilizar vídeo, a arquitectura deve suportar a partilha de sessões SIP entre diferentes aplicações multimédia. Além destes requisitos aplicacionais, a arquitectura idealmente deverá fornecer a possibilidade de aplicações multimédia já desenvolvidas façam uso do protocolo SIP. Como ponto de partida à integração do protocolo SIP na plataforma Android, é feito um estudo da framework OpenCore para obter conhecimento de como proceder à integração da nova componente SIP na sua arquitectura.

3.2 Estudo da *framework* OpenCore

Como definido em 2.3.3, a *framework* OpenCore é um conjunto de componentes que possibilitam a reprodução de vídeo. Com base nessa descrição, ao longo desta secção será apresentada uma análise mais detalhada do seu funcionamento. Para isso, será utilizada alguma documentação existente sobre as APIs de algumas das suas componentes. No entanto, essa documentação aborda de forma genérica o seu funcionamento, não fornecendo informação técnica suficiente para se integrar o protocolo SIP do ponto de vista prático. Assim sendo, ao longo desta secção,

o estudo desses documentos é feito com acompanhamento e análise do respectivo código, o que possibilitará a obtenção de um conhecimento mais aprofundado sobre a *framework* e auxiliará a integração da nova componente SIP.

Como ponto de partida, é feito um estudo à interface da *framework* OpenCore. Essa interface tem o nome de PVPlayer e funciona como uma API para a manipulação das componentes da *framework*.

3.2.1 Introdução ao PVPlayer SDK

O PVPlayer SDK da *framework* OpenCore, é a API para um conjunto de componentes que permitem a reprodução de elementos multimédia. Estes elementos multimédia, são reproduzidos de forma sincronizada (exemplo: *frames* de vídeo), podendo ter diferentes origens (ficheiro, *stream*) e diferentes formatos de codificação (MP4, 3GPP, etc). Além de disponibilizar as funcionalidades básicas para a reprodução de elementos multimédia, fornece funcionalidades mais avançadas, como por exemplo, o controlo do volume do dispositivo durante a reprodução e a reprodução incremental de elementos multimédia (reprodução da *media* à medida que é feito o download). A sua arquitectura é modelar, e de fácil portabilidade entre diferentes plataformas, no entanto, a plataforma pode limitar algumas das suas funcionalidades.

Em resumo, o seu principal objectivo é abstrair a utilização dos vários componentes da *framework*, reduzindo assim, a complexidade da sua utilização.

3.2.2 Arquitectura do PVPlayer SDK

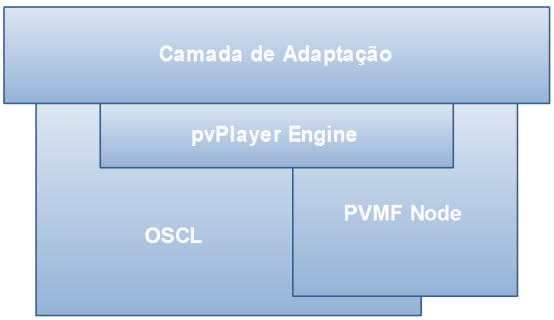

Figura 3.1: Arquitectura do PVPlayer SDK

A Figura 3.1 apresenta a arquitectura do PVPlayer SDK. Como se pode observar, a arquitectura do PVPlayer SDK agrupa os componentes da *framework* OpenCore, de uma forma mais simples. Esta é composta por quatro componentes: camada de adaptação, pvPlayer Engine, OSCL e PVMF Node.

No topo da arquitectura é apresentada a camada de adaptação. Esta componente é utilizada como um "proxy" para a componente pvPlayer Engine. É através dela que a aplicação, ou no caso da plataforma Android a Media Framework, invoca as funcionalidades da componente pvPlayer Engine. A utilização desta camada na arquitectura da *framework* OpenCore não é obrigatória, no entanto, na plataforma Android é utilizada para reduzir a complexidade da integração da *framework* OpenCore na Media Framework.

Na camada inferior à camada de adaptação, está situada a camada da componente pvPlayer Engine. Esta componente é o coração da *framework*, é nela que é feita a gestão das componentes utilizadas para a manipulação de *media*. A gestão das componentes e dos estados do pvPlayer Engine é feita com suporte de uma máquina de estados. A cada estado são associadas determinadas acções, sendo que o pvPlayer Engine só transita para o estado seguinte, assim que as acções estejam concluídas ou que se verifique alguma situação que obrigue a transição de estado (exemplo: erros). A máquina de estados do pvPlayer Engine e a sua descrição será apresentada posteriormente em 3.2.3.

Por fim, na camada inferior da arquitectura, são apresentadas duas componentes: a PVMF Node e a Operating System Compatibility Library (OSCL). A componente PVMF Node contém os módulos para a manipulação da *media*. A OSCL fornece uma API independente da plataforma para a manipulação de funcionalidades do sistema, como por exemplo: gestão eficiente de memória, bibliotecas optimizadas para obter um melhor desempenho do processador durante a descodificação e codificação de vídeo e áudio.

3.2.3 pvPlayer Engine

Como referido na subsecção anterior, o pvPlayer Engine consiste numa máquina de estados onde cada estado tem associadas acções. A reprodução de um elemento multimédia consiste na execução da máquina de estados, onde cada estado vai executar acções e carregar os componentes até que seja possível reproduzir o elemento multimédia. Após a execução do elemento multimédia ou após o evento de paragem de reprodução, toda a máquina de estado é reiniciada, voltando esta ao seu estado inicial. A Figura 3.2 representa a máquina de estados do pvPlayer Engine.

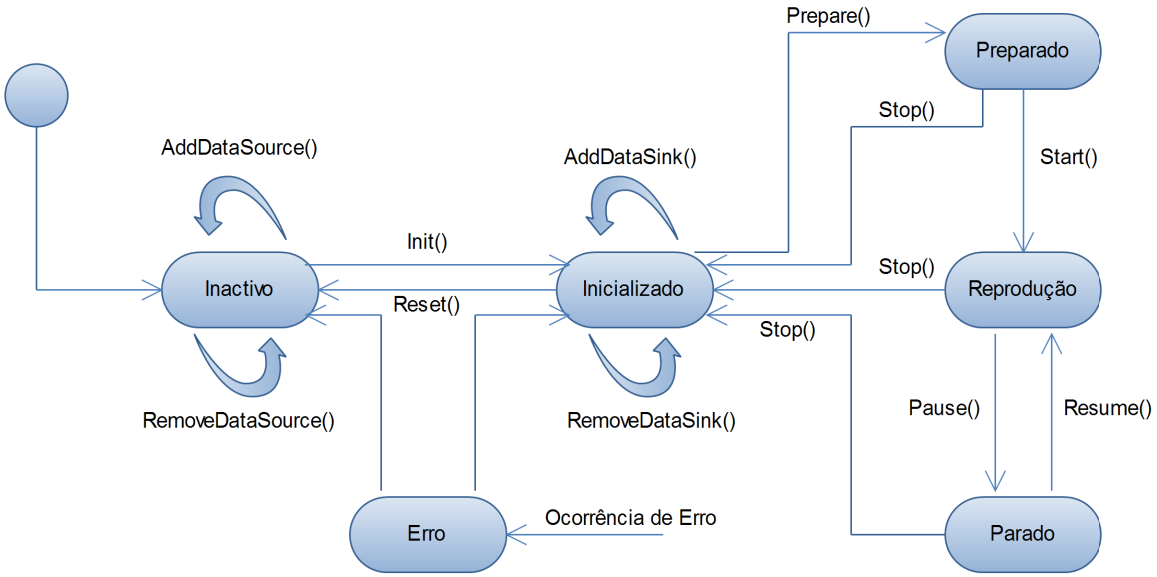

Figura 3.2: Máquina de estados do pvPlayer Engine da framework OpenCore

Como podemos observar pela Figura 3.2, a máquina de estados do pvPlayer Engine é composta por seis estados possíveis: inactivo, inicializado, preparado, reprodução, parado e erro. O pvPlayer Engine encontra-se num estado inactivo, logo após a sua instanciação, isto é, assim que é invocado o serviço. Neste estado é carregado e inicializado a componente que vai fornecer os dados a reproduzir. O método AddDataSource(), é o método responsável pela inicialização dessa componente. Definida a fonte dos dados, seja ela *streaming* ou ficheiro, o pvPlayer Engine está pronto para avançar para o próximo estado.

A inicialização da componente que vai fornecer os dados, é feita dinamicamente dependendo do tipo de *media* a reproduzir. Inicialmente é analisado o tipo de *media* a reproduzir, sendo posteriormente feita uma pesquisa aos registos dos vários módulos disponíveis na *framework* para tratar esse tipo de dados. O ficheiro *pvplayer.cfg* é o ficheiro que contém a lista dos módulos disponíveis a serem utilizados pelo pvPlayer Engine. Feita a pesquisa ao ficheiro *pvplayer.cfg* e encontrado o respectivo módulo a ser utilizado, este é carregado e inicializado. A inicialização dos diferentes módulos suportados pelo pvPlayer Engine é feita de forma análoga recorrendo à interface SMNodeFSPSharedLibraryInterface, sendo que todas as componentes que funcionam como fonte de dados para a reprodução da *media*, têm, obrigatoriamente, de implementar esta interface para sua inicialização. A interface SMNodeFSPSharedLibraryInterface define os método de inicialização das bibliotecas a ser importadas em tempo de execução pelo pvPlayer Engine.

Concluídas as acções do estado inactivo, através da evocação do método *Init()* do pvPlayer Engine, o seu estado passa para inicializado. No estado inicializado, o pvPlayer Engine questiona a componente que vai fornecer os dados sobre a informação (*metadados*) relativa à *media* a ser reproduzida. Obtida esta informação, o pvPlayer Engine consulta o registo de *codecs* suportados, e se for encontrado no seu registo um *codec* para o tipo de *media* a reproduzir, então esse módulo será carregado e inicializado juntamente com os respectivo componente de reprodução (*Render*). Caso não exista nenhum *codec* registado para o tipo de *media* a reproduzir, será devolvido um erro ao pvPlayer Engine, o qual informará a camada superior do problema sucedido e transitará para o estado de erro. A consulta à fonte para obter informações sobre a *media* é feita recorrendo a interface PVInterfaceMetadaExt. As componentes de reprodução, são vistas pelo pvPlayer Engine como *Sink Nodes*, podendo estes não necessitar de inicializar o descodificador de *media*, caso o reprodutor tenha suporte para a reprodução directa da *media*. Concluída a inicialização dos Sink Nodes o pvPlayer Engine fica disponível para avançar para o próximo estado. Caso ocorra algum erro durante as acções referidas a máquina de estados transita para o estado de erro.

Após a execução de todas as acções do estado inicializado, é possível evocar o método prepare() para transitar para o estado de preparado. A evocação do método prepare(), implica preparar os seus Sink Nodes para execução e a invocação do comando de preparação no componente a fornecer os dados. Devolvida a resposta de sucesso dos Sink Nodes e da componente a fornecer os dados, então a máquina de estados transita para o estado de preparado. Em caso de falha dos Sink Nodes ou da componente de dados, a máquina de estados transita para o estado de erro.

O envio do comando de preparação para a componente que fornece os dados, para o reprodutor e para as componentes de descodificação/codificação se necessárias, implica a interligação entre estas três componentes. Recebido o comando de preparação no componente que fornece os dados, este têm de preparar um segmento de memória contínuo (*buffer*) para guardar a *media*. Após a alocação do *buffer*, é estabelecida uma ligação com o componente de descodificação/codificação. Esta ligação é feita através de um objecto que utiliza Sockets Unix para transmitir informação entre diferentes componentes. Por sua vez, a componente de descodificação/codificação ao receber o comando de preparação prepara-se para iniciar a descodificação, ficando à espera de receber a *media* através da porta de comunicação com a componente fonte. Por fim, a componente de reprodução, analogamente à componente de descodifcação, ao receber o comando de preparação também se preparar para iniciar a reprodução, ficando à espera de receber a *media* descodificada através da porta de comunicação ligada ao componente de de-

scodificação. A Figura 3.3 apresenta a interligação das componentes de leitura, descodificação e reprodução.

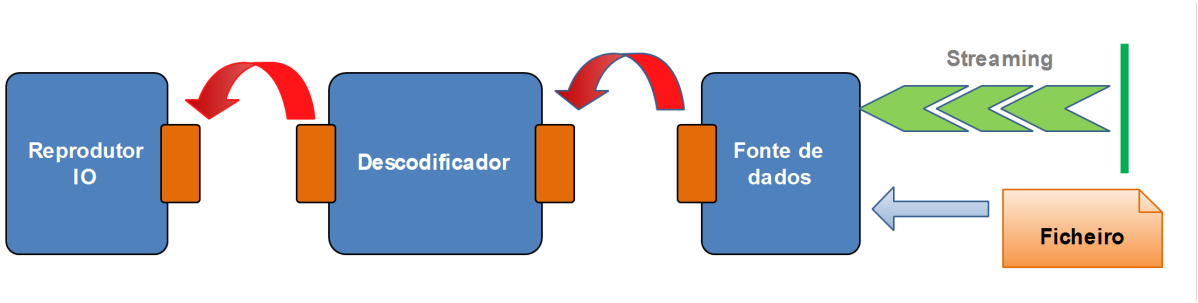

Figura 3.3: Transmissão da *media* entre as componentes de recepção, descodificação e reprodução

Concluidas todas as acções do estado preparado, através da evocação do método start() é enviado o comando start aos *Sink Nodes* e à componente que vai fornecer os dados. Estes, por sua vez, iniciam as suas funcionalidades e retornam o seu estado. No caso dos descodificadores e dos reprodutores, iniciam as suas funções assim que recebem a *media* a descodificar ou a reproduzir, respectivamente. No caso da componente a fornecer a *media*, inicia a leitura ou a recepção da *media*, enviando essa informação para a componente de descodificação ou de reprodução, caso não seja necessário a descodificação da *media*. Recebida a resposta das componentes, a máquina de estados transita para o estado de reprodução ou para o estado de erro, dependendo da resposta recebida das componentes.

Durante a reprodução da *media*, a máquina de estados pode transitar para o estado de parado no caso da evocação do método de *pause()*, ou transitar para o estado inactivo no caso da evocação do método *stop()*. Nesta última situação, a evocação do método *stop()* implica que a componente que fornece os dados e as componentes de descodificação e reprodução sejam reiniciadas, sendo é feita uma limpeza aos seus *buffers* e reiniciado o seu estado interno.

3.2.4 Sumário

Como se pode observar ao longo da descrição do PVPlayer SDK e em particular da componente pvPlayer Engine, a integração da componente que implementa o protocolo SIP terá de ser controlada pelo pvPlayer Engine. Para isso, será necessário que a nova componente a desenvolver siga o funcionamento interno do pvPlayer Engine e implemente as interfaces necessárias para a sua manipulação. Um outro aspecto importante, é a comunicação com as componentes de

descodificação/codificação.

O desenvolvimento da componente SIP, necessita, analogamente ao RTSP, de implementar a comunicação com os descodificadores ou reprodutores para o envio da *media* recebida. O estudo do funcionamento da componente RTSP é não só importante para verifcar o seu funcionamento com a máquina de estados do pvPlayer Engine, com também para ter conhecimento mais detalhado sobre a comunicação com os descodificadores e os reprodutores.

3.3 Protocolo RTSP na framework OpenCore

Ao longo desta secção será abordada a componente que implementa o código RTSP na *framework* OpenCore. No entanto, o facto de não existir documentação sobre o funcionamento desta componente, implica que a descrição dada sobre o seu funcionamento seja baseada no estudo do seu código.

Sem documentação e não sabendo à partida quais as componentes utilizadas durante a execução do protocolo, a solução adotada para ultrapassar este problema consiste em compilar o código fonte do Android com a *flag* de *debug* activa. Esta *flag* força a que todos os *logs* escritos no código fonte, sejam impressos no ficheiro de *log* do Android (*logcat*). Através da visualização dos *logs* em modo *debbug* durante a reprodução de uma *streaming* RTSP, é possível identificar quais as componentes utilizadas numa sessão RTSP e estudar o seu funcionamento.

3.3.1 Arquitectura

A *framework* OpenCore é composta por várias componentes, entre as quais, a componente de formatos de dados. Como referido anteriormente em 2.3.3, a componente de formatos de dados é responsável pela manipulação da *media* a reproduzir, sendo esta constituída por subcomponentes específicas para a manipulação da *media*. A Streaming Node é uma das subcomponentes da Formatos de dados. A sua funcionalidade é habilitar a plataforma Android da capacidade de manipular diferentes tipos de *streaming* de *media*. O protocolo RTSP é um dos protocolos de *stream* suportados actualmente pela Streaming Node na plataforma Android. A figura 3.4 apresenta a arquitectura do protocolo RTSP na *framework* OpenCore.

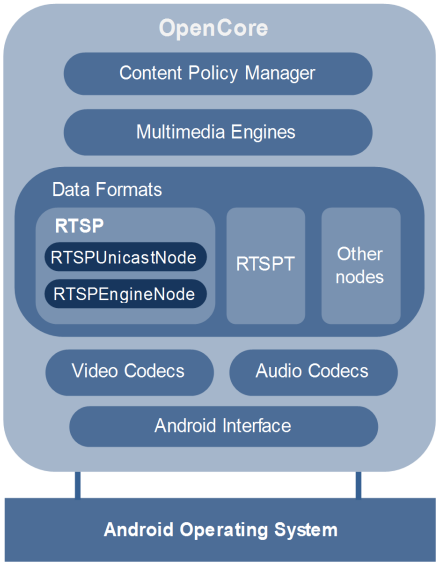

Figura 3.4: Arquitectura do protocolo RTSP na framework OpenCore

3.3.2 Funcionamento da biblioteca de Stream RTSP

Como se pode observar pela Figura 3.4, existem diferentes tipos de nodos na camada de dados. Os diferentes nodos disponíveis nesta camada, entre os quais o Streaming Node, estão compilados sobre a forma de bibliotecas no sistema, sendo essas bibliotecas registadas no serviço durante o arranque para posterior importação.

Durante a inicialização de uma *stream* multimédia na plataforma Android, a camada de dados da *framework* OpenCore é responsável pela manipulação da *media* a reproduzir. Esta efectua um teste ao tipo de *media* (*Streaming*, ficheiro, etc), e caso o tipo de *media* ser uma *stream*, através da pesquisa ao registo dos formatos de dados suportados é importado ao sistema o respectivo nodo de *stream* (Streaming Node).

O Streaming Node, como referido anteriormente, é uma biblioteca para manipulação de Streaming. A sua funcionalidade na *framework* OpenCore é habilitar a capacidade de manipular *streams* de *media*, disponibilizando uma interface genérica para os diferentes tipos de *streams* suportados. Através desta interface genérica o pvPlayer Engine é capaz de manipular da mesma forma uma *stream* RTSP ou HTTP progressivo, abstraindo o pvPlayer Engine do protocolo de sessão utilizado.

Durante a inicialização do nodo de *stream*, é feito um teste à URL da *stream* a reproduzir,

obtendo o tipo de protocolo a utilizar. Reconhecido o protocolo a utilizar durante a sessão, é feita uma pesquisa ao registo de protocolos suportados pelo Streaming Node, e, caso o protocolo seja suportado, é criado o respectivo nodo. A biblioteca Streaming Node, é compilada juntamente com o código fonte da plataforma Android e tem o nome de *libopencore_streaming*. Esta é registada estaticamente no registo (*pvplayer.cfg*) sendo carregada dinamicamente em tempo de execução.

Analogamente ao Streaming Node, o protocolo RTSP é também uma biblioteca compilada juntamente com o código fonte da plataforma Android e é importada dinamicamente em tempo de execução, sempre que for necessário à *framework* OpenCore estabelecer uma sessão multimédia RTSP. A manipulação do protocolo RTSP na plataforma Android é feita utilizando duas componentes principais: a RTSPUnicastNode e a RTSPEngineNode.

A componente RTSPUnicastNode, tem como objectivo, receber comandos para executar acções sobre o protocolo RTSP. Basicamente, este nodo funciona como uma interface para o pvPlayer Engine manipular o protocolo RTSP. Por sua vez, a componente RTSPEngineNode, faz o processamento da sessão RTSP. É na componente RTSPEngineNode que é criado o estado da sessão e são enviadas as mensagens RTSP, consoante o estado e as acções que se pretendem executar. A Figura 3.5, apresenta um diagrama ilustrativo das componentes utilizadas durante o funcionamento da biblioteca.

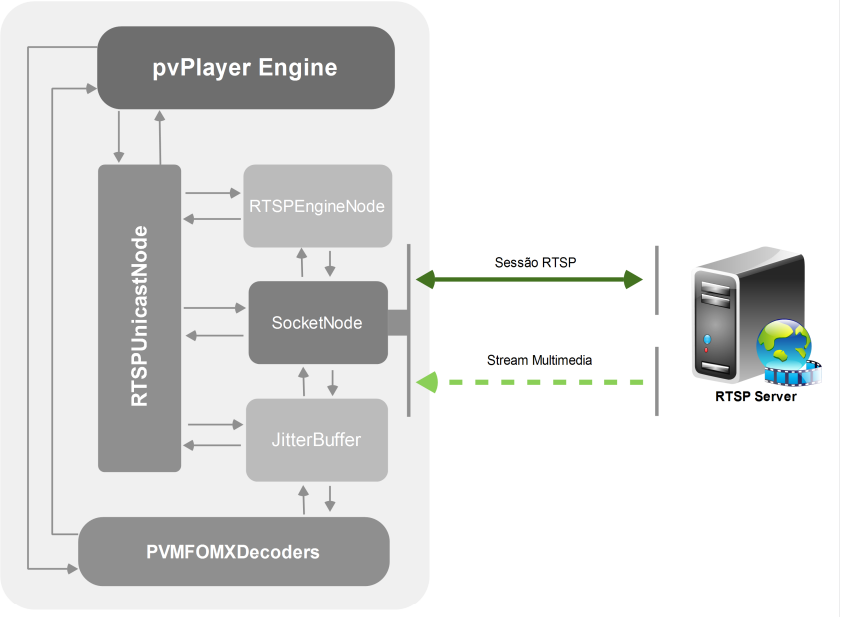

Figura 3.5: Componentes utilizados pela biblioteca RTSP

A invocação das funcionalidades RTSP, invocadas pelo RTSPUnicastNode ao RTSPEngineNode, utiliza um sistema de *queue* de comandos. Quando o RTSPUnicastNode pretende executar uma determinada acção, envia o comando respectivo de essa acção ao RTSPEngineNode. Por sua vez, o comando é guardado numa fila de comandos e executado assim que houver disponibilidade. A cada comando enviado pelo RTSPUnicastNode ao RTSPEngineNode, é associado um código que identifica esse comando durante a sessão. Esse código é utilizado para no final da execução do comando, ser devolvido o resultado da sua execução ao RTSPUnicastNode. A devolução dos resultados do comando após a execução, é feita através de um *callback*, sendo utilizado para o RTSPUnicastNode informar o pvPlayer Engine do estado da sessão.

A comunicação com o servidor RTSP para o estabelecimento da sessão e a recepção da *stream* de multimédia, é efectuada pelo RTSPEngineNode com recurso a três componentes: a bilioteca *libpv_rtsp_parcom*, o SocketNode e o JitterBuffer.

A biblioteca *libpv_rtsp_parcom* é utilizada pelo RTSPEngineNode para a construção e leitura das mensagens RTSP. Esta biblioteca disponibiliza um conjunto de funcionalidades para a manipulação e troca de mensagens RTSP entre o dispositivo e o servidor, efectuando o respectivo *parser* e construção das mensagens com base na definição padrão do protocolo. O envio e recepção das mensagens é efectuado através da componente SocketNode.

A SocketNode é a componente que implementa os *sockets* para a comunicação com o servidor RTSP e para a recepção da *stream* multimédia. Esta componente disponibiliza métodos para a criação e manipulação dos *sockets*, reservando e gerindo de forma eficiente a alocação de memória para armazenar a informação recebida. Após a recepção da informação no *socket*, através do método HandleSocketEvent(), a informação é devolvida à componente que requereu o *socket*. Essa informação é devolvida retornando o respectivo apontador para a estrutura que contém a informação.

Por fim, o JitterBuffer é a componente que controla a *media* recebida. Esta componente está dividida em quatro funcionalidades principais: processamento dos pacotes RTP recebidos pelo *socket*, armazenamento temporário da *media* recebida, controlo do fluxo de pacotes recebidos durante a sessão, e, por fim, pré-processamento da *media*.

O processamento dos pacotes RTP recebidos pelo *socket* é a primeira tarefa a ser executada pelo JitterBuffer após a recepção da informação do *socket*. Este processamento efectua a leitura da estrutura padrão do protocolo RTP a partir da informação recebida, que consiste em retirar a *media* a reproduzir do pacote RTP e identificar a informação de transmissão da *media*, como por exemplo: tamanho do pacote, *timeStamp* do RTP, número de sequência do pacote, etc. Após

a leitura da media e de toda a informação do pacote RTP, essa informação é colocada num formato específico (*OsclSharedPtr<PVMFMediaDataImpl>*), o qual será utilizado para manipular a *media* e a sua respectiva informação pelas restantes componentes.

A alocação da estrutura do novo formato e o espaço de memória onde será guardada a *media*, utiliza um conjunto de blocos de memória previamente alocada. Esta alocação é feita pela classe *PVMFMediaFragGroupCombineAlloc<OsclMemAllocator>*, a qual gere o espaço alocado de forma eficiente. Basicamente, na instanciação do objecto de alocação, é alocado um conjunto de blocos de memória pré-definida, que servirá para guardar a *media*. O espaço de memória é reservado inicialmente ao sistema, sendo libertado apenas na destruição do objecto. Esta componente de alocação de memória guarda a informação respectiva aos seus blocos, os que estão alocados e os que estão livres. Sempre que um bloco é requerido para guardar um novo fragmento de *media*, é devolvido um bloco marcado como livre e a informação é reescrita. Sempre que um bloco é libertado, este apenas é marcado como livre.

Processados os pacotes RTP recebidos, essa informação será guardada num *buffer* temporário, o qual é utilizado para evitar que eventuais atrasos na entrega do pacotes por parte do servidor, possam causar atrasos na reprodução da *media*. Para isso, é armazenada uma determinada quantidade de fragmentos no *buffer*, sendo essa quantidade controlada recorrendo ao protocolo RTCP. Assim sendo, a componente de *buffer* comunica com o servidor RTSP através do protocolo RTCP para garantir a dimensão do *buffer*, equilibrando a recepção da media em relação ao seu envio para descodificação e reprodução.

Preenchido o *buffer*, o JitterBuffer está pronto para iniciar o envio dos fragmentos recebidos para os descodificadores. Após a recepção do comando de início de envio da *media* para os descodificadores, o JitterBuffer começa a consumir os fragmentos do seu *buffer*. Antes do envio dos fragmentos para os descodificadores, é feito um pré-processamento da *media*. Este pré-processamento consiste em fazer um *parse* ao *payload* da *media*, verificando se este está correcto. Este pré-processamento evita que *media* de pacotes corrompidos seja enviada para os descodificadores. Desta forma, os pacotes que derem erro neste pré-processamento, serão automaticamente descartados. Antes do envio do primeiro fragmento de *media* para o descodificador ou reprodutor, é enviada pelo JitterBuffer uma mensagem BOS. Esta mensagem está contida na mesma estrutura que os pacotes de *media*, mas contém um identificador especial a informar o descodificador do início de uma transmissão de *stream*. Após o envio de toda a *stream* recebida, analogamente à mensagem BOS, é enviada uma mensagem EOS, que sinaliza ao descodificador o fim da transmissão.

O envio da informação do JitterBuffer para os descodificadores é feita recorrendo ao método *Receive()* da interface *PVMFPortInterface*, estando limitado à recepção de dez fragmentos. Atingido esse limite, o método *isIncomingFull()* da interface *PVMFPortInterface* retorna o valor de verdadeiro, devendo ser parado o envio dos fragmentos de *media* para o descodificador. Caso este limite não seja respeitado, a porta devolverá um erro durante a utilização do método *Receive()*. Após o consumo dos fragmentos por parte do descodificador ou reprodutor, podem ser enviados novamente mais fragmentos até ser atingido novamente o limite, criando assim um ciclo de envio e consumo. Este ciclo é parado quando receber uma mensagem EOS, ou quando as componentes receberem o comando para parar a operação.

3.3.3 Funcionamento do Media Player com o RTSP

Como referido anteriormente em 3.2.3, o pvPlayer Engine da *framework* OpenCore, tem como base de funcionamento, uma máquina de estados. Essa máquina de estados é composta por seis estados possíveis: inactivo, inicializado, preparado, reprodução, parado e erro.

O pvPlayer Engine encontra-se num estado inactivo, logo após a sua instanciação, isto é, assim que é evocado o serviço. Neste estado é possível definir a fonte de dados a reproduzir. No caso da fonte ser uma *stream* RTSP, o pvPlayer Engine importa as bibliotecas de *streaming* RTSP e evoca a funcionalidade de definição da fonte da *stream*, passando a *URL*. Por sua vez, o Streaming Node recebe o pedido do pvPlayer Engine, faz o pedido ao nodo RTSPUnicastNode, o qual envia o comando ao RTSPEngineNode para definir a *URL* fonte. Após o pvPlayer Engine receber a confirmação por parte do Streaming Node, que a fonte foi definida, o pvPlayer Engine está pronto para avançar para o próximo estado (inicializado).

Definida a fonte da *media* a reproduzir, evocando o método *init()* do pvPlayer Engine, o seu estado passa para inicializado. No estado de inicializado, o pvPlayer Engine vai inicializar os Sink Nodes e a componente de fonte de dados, neste caso o RTSPUnicastNode. A evocação do comando *init()* no StreamingNode evoca o comando *init()* na componente RTSPUnicastNode, onde, por sua vez, irá criar a componente para recepção da *media* (SocketNode), o JitterBuffer, que será utilizador para armazenar temporáriamente a *media* recebida, e o RTSPEngineNode que iniciará a sessão RTSP.

Uma sessão RTSP é iniciada com o envio do comando DESCRIBE pelo RTSPEngineNode. O servidor RTSP devolve de seguida uma resposta que contém um SDP com a informação da *media* a receber ou uma resposta de erro. Se a resposta recebida, retornar erro, então é devolvido

o erro por *callback* ao pvPlayer Engine. Caso contrário, são carregadas todas as definições da *media* a receber e é devolvido por *callback* o estado de sucesso. Após o pvPlayer Engine receber a confirmação, que o comando foi efectuado com sucesso, então o pvPlayer Engine prepara todos os seus Sink Nodes com os *metadados* da *media*, ficando pronto para avançar para o próximo estado. Caso tenha ocorrido um erro durante a execução do comando no RTSPEngineNode, o PVPlayer Engine transita para o estado de erro.

Inicializados os Sink Nodes com os respectivos *metadados* da *media*, o pvPlayer Engine pode transitar novamente para o estado de inactivo cancelando a operação, ou pode transitar para o estado de preparado evocando o método *prepare()*. A evocação do método *prepare()*, implica preparar todos os seus Sink Nodes para execução e a evocação do comando de preparação no RTSPUnicastNode. Com a evocação do método de preparação, o RTSPEngineNode inicia os JitterBuffers para guardar temporariamente a *media* recebida e os *sockets* para receber a *media* a ser transmitida pelo servidor. Após iniciar as componentes necessárias para a recepção da *media*, envia o comando SETUP ao servidor com a descrição das portas as quais os seus *sockets* estão a escuta. O servidor de seguida responde com as portas nas quais vai utilizar na sessão multimédia, ou com uma mensagem de erro. Recebida a resposta do servidor, se a operação for bem sucedida é então guardada a informação e é devolvido um *callback* ao pvPlayer Engine a informar que está pronto para iniciar a recepção da *media*. Caso contrário, se foi devolvido algum erro pelo servidor, ou ocorreu algum erro durante a preparação dos *nodos*, é devolvida uma *callback* com erro e o pvPlayer Engine transita para o estado de erro.

Preparado o pvPlayer Engine e todas as componentes para receber a *media*, a sua máquina de estados pode transitar para o estado de reprodução através da evocação do método *start()*. A evocação do método *start()* implica que o RTSPEngineNode envie o comando PLAY ao servidor RTSP. Após a resposta de 200 OK do servidor, inicia-se a transmissão por parte do servidor e a recepção por parte do cliente. A *media* recebida pelos *nodos* de recepção (*sockets*) é passada ao JitterBuffer, onde, posteriormente, é passada ao nodo de descodificação para ser descodificada e posteriormente reproduzida.

Durante a recepção e reprodução da *media*, o pvPlayer Engine pode transitar para o estado de *pause* através do comando *pause()*. A transição para o estado *pause*, implica o envio do comando de PAUSE ao servidor RTSP. Esse comando faz com que o servidor pare a transmissão. No entanto, todas as componentes ficam à espera de receber novamente a *media*. É possível, posteriormente, continuar a reprodução da *media* sem haver a necessidade de reiniciar todo o processo de transmissão. Quando evocado o comando de STOP no pvPlayer Engine, é enviado

o comando TEARDOWN ao servidor RTSP. Neste caso, todas as componentes são paradas e eliminadas, voltando ao estado de inicializado.

3.3.4 Sumário

Como se pode observar ao longo desta secção, a componente RTSP é uma extensão à componente de *streaming*. A manipulação do protocolo é feita com base na sua componente que integra o protocolo (RTSPEngineNode), a qual faz uso de segundas componentes para o envio e recepção das mensagens pela rede e o armazenamento temporário da informação a reproduzir. O seu controlo é feito pelo pvPlayer Engine por intermédio da componente SIPUnicastNode, a qual funciona como *proxy* para a biblioteca RTSP.

Com base no estudo feito ao longo das secções 3.2.1 e 3.2.4, é possível definir uma arquitectura para a integração do protocolo SIP na *framework* Opencore. Embora o funcionamento do protocolo SIP seja diferente do protocolo RTSP, este terá de implementar a mesma estrutura de recepção de comandos do pvPlayer Engine e respeitar a comunicação com os descodificadores para efectuar o envio dos fragmentos de *media* a reproduzir.

3.4 Arquitectura proposta para a biblioteca SIP

Com base na apresentação dada pela Google sobre a Media Framework do Android[4], o estabelecimento das sessões multimédia actualmente suportadas pela Media Framework do Android (RTSP e HTTP Streaming), é directamente tratado na *framework* OpenCore. Isto deve-se ao facto da *framework* OpenCore ser um projecto independente do Android, o que obriga a que a *framework* OpenCore não dependa de terceiros para a execução das suas funcionalidades. Este factor implica a integração do protocolo SIP directamente na *framework* OpenCore, o que por um lado limita a partilha de sessões SIP apenas por clientes multimédia, mas, em contrapartida, cumpre a transparência da sua utilização nas actuais e novas aplicações multimédia.

Ao longo desta secção é apresentada a arquitectura proposta para a integração do protocolo SIP na *framework* OpenCore. São descritas as funcionalidades das suas componentes e a razão da necessidade da sua implementação.

3.4.1 Arquitectura

Olhando novamente para a arquitectura da *framework* OpenCore apresentada no estado de arte, Figura 2.10, a componente SIP deverá ser integrada na camada de formato de dados. Esta nova componente necessita de implementar a mesma interface que os protocolos de sessão actualmente suportados pela *framework*, adicionando apenas a essa interface as funcionalidades extra do SIP, como por exemplo o registo no *SIP proxy*. Além da implementação da interface para que a componente SIP possa ser utilizada pelas componentes da camada superior, esta nova componente, necessita de manipular as restantes componentes das camadas inferiores, como por exemplo os *codecs* de vídeo e áudio.

A Figura 3.6 apresenta a arquitectura proposta para a integração do protocolo SIP na *framework* OpenCore.

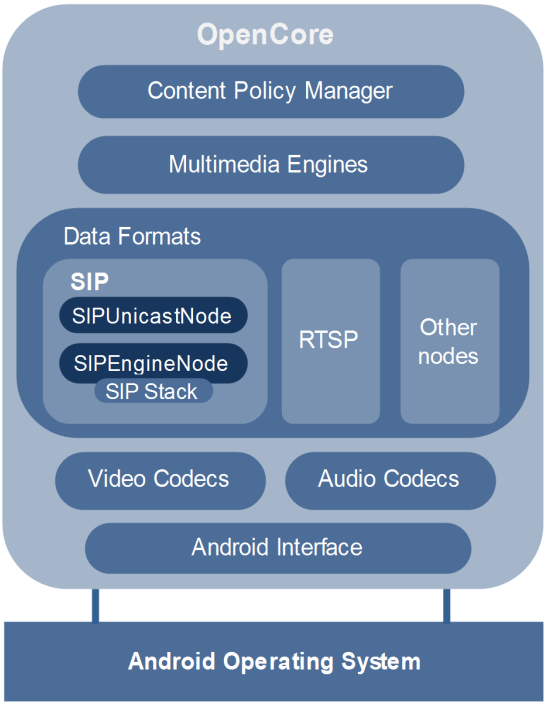

Figura 3.6: Arquitectura proposta para a integração do protocolo SIP

Como podemos observar na Figura 3.6, analogamente ao protocolo RTSP, o protocolo SIP está situado na camada de dados da *framework* OpenCore, sendo este uma subcomponente do Streaming Node. Esta integração respeita a divisão estrutural das componentes da *framework*,

seguindo a definição da estrutura inicial para a integração de futuras componentes ou "*plugins*", como são definidas as componentes dos protocolos de sessões na *framework* OpenCore.

A nova arquitectura definida, além de vir possibilitar a utilização do vídeo em sessões multimédia SIP, responde aos requisitos definidos em 3.1. A sua integração ao nível da *framework* OpenCore, vem oferecer transparência na utilização do protocolo SIP em relação aos restantes protocolos de *stream*. Não existindo alterações à actual Media Framework da plataforma Android, a utilização de um endereço SIP para a aplicação será análoga à utilização de um endereço RTSP, sendo posteriormente os endereços e a reprodução tratados de forma diferente na *framework* OpenCore. Além da transparência da sua utilização pelas actuais aplicações, a arquitectura proposta possibilita a partilha de sessões SIP entre aplicações multimédia, guardando os *Users Agent* (informação relativa ao registo do utilizador) dos possíveis registos em servidores SIP, podendo estes *Users Agents* ser usados por diferentes aplicações.

Espera-se ainda que a integração proposta melhore as actuais aplicações de voz que utilizam sessões SIP. O facto de utilizar a descodificação da *media* com bibliotecas optimizadas da *framework* OpenCore, diminui o processamento durante a descodificação, o que garante, consequentemente, uma diminuição do consumo energético do dispositivo. Além fornecer uma maior eficiência energética, possibilita que as actuais aplicações utilizem *codecs* com maior compactação nas suas transmissões, o que reduz o custo e exigências da rede durante a transmissão.

A arquitectura apresentada na Figura 3.6 foi publicada no artigo "Integration of SIP protocol in Android Media Framework"apresentado na conferência internacional EUROCON 2011 - "IEEE International Conference on Computer as a Tool".

3.4.2 Componentes da biblioteca SIP

Na arquitectura proposta, analogamente a arquitectura do protocolo RTSP, é necessário ter um nodo SIPUnicastNode que funcionará como interface para o Streaming Node. Este nodo é crucial para garantir a integração deste novo módulo na actual arquitectura, garantindo a possibilidade de este poder ser controlado pelo pvPlayer Engine. É ainda necessário o desenvolvimento do componente SIPEngineNode que irá controlar a negociação da sessão SIP. No entanto, ao contrário do RTSPEngineNode que implementa a lógica do protocolo RTSP e o envio das mensagens do protocolo, a componente SIPUnicastNode apenas funcionará como máquina de estado para controlar a sessão SIP.

A manipulação e transmissão do protocolo SIP, é feita por uma componente (*Sip stack*) in-

dependente da SIPEngineNode. Esta separação facilita o desenvolvimento e a manutenção do código, ficando a integração do protocolo na *framework* OpenCore independente da *Sip stack* utilizada para a manipulação do protocolo SIP. A comunicação entre o SIPEngineNode e a *SIP stack*, utiliza uma classe intermédia chamada Adapter. Esta classe segue o padrão *facade*[41], e tem como objectivo criar uma maior independência do SIPEngineNode à *stack* SIP utilizada, garantindo que em futuras alterações a *SIP stack* apenas seja necessário efectuar alterações à componente Adapter.

O SIPUnicastNode além funcionar como uma interface para a manipulação do protocolo SIP, analogamente ao RTSPUnicastNode, também é responsável por criar e gerir as componentes de recepção da *stream* e as componentes de envio dos fragmentos de *media* para os descodificadores. A componente de recepção de *media* tal como a SocketNode do RTSP, receberá a informação dos *sockets* e passará essa informação à componente que a tratará e a passará aos descodificadores. Por sua vez, a componente de envio dos fragmentos de *media* para os descodificadores, assim como o JitterBuffer do RTSP, faz o parser do pacote RTP e coloca a *media* no formato reconhecido pelos descodificadores (*OsclSharedPtr<PVMFMediaDataImpl>*). Colocada a *media* no formato específico para os fragmentos de *media*, esta passa pelo *parser* de *payload* para fazer um pré-processamento de possíveis erros. Feito o pré-processamento da *media*, se ocorrer algum erro durante o *parser* do *payload*, o pacote é automaticamente descartado. Caso contrário, o fragmento é enviado para o descodificador, onde será descodificado e posteriormente enviado para o reprodutor.

3.4.3 Resultados Esperados

A nova arquitectura proposta, responde à necessidade de integração do protocolo SIP na plataforma Android, dando não só possibilidade de utilizar áudio nas sessões SIP, como também o suporte para vídeo, o que era impossível até ao momento. Com a integração do protocolo SIP na plataforma Android através da arquitectura proposta, espera-se, além das funcionalidades referidas em 3.1, um desempenho muito idêntico à reprodução de *streams* multimédia utilizando o protocolo RTSP.

Baseado no estudo feito ao pvPlaver Engine em 3.2.3 e ao funcionamento do protocolo RTSP em 3.3.3, após a inicialização da sessão a descodificação e reprodução da *media* apenas difere na componente que envia os fragmentos para os descodificadores. Uma vez que esta componente, no caso do protocolo SIP, não irá fazer gestão de um *buffer* de fragmentos de *media*, reduz-

se a utilização de um segundo protocolo para gestão do *buffer*. Este factor diminui a carga de processamento desse protocolo, a qual deverá ser visível nos resultados obtidos, mesmo que essa diferença seja pouco significativa.

3.5 Sumário

Como apresentado ao longo deste capítulo, embora as componentes da *framework* OpenCore tenham um funcionamento complexo, a sua estrutura foi projectada para ser extensível. Este factor facilitou a definição da arquitectura para a integração da biblioteca SIP, dando a possibilidade de esta se integrar de forma dinâmica na *framework* OpenCore.

Com base na arquitectura definida neste capítulo para a integração do protocolo SIP, o próximo passo será proceder à implementação da arquitectura.

Capítulo 4

Implementação

Ao longo deste capítulo é apresentada a implementação da arquitectura SIP definida em 3.4.1. Como ponto de partida, é feita um estudo à SIP *stack* a integrar na plataforma Android, sendo descrita a sua arquitectura e as alterações necessárias para a sua integração na plataforma Android.

Descrita a integração da SIP *stack* na plataforma Android, segue-se a descrição da integração da biblioteca SIP. Inicialmente, explica-se como proceder ao registo da nova biblioteca SIP, no registo do sistema da *framework* OpenCore. Após o registo, é apresentado de forma detalhada o funcionamento de cada componente da biblioteca SIP e o funcionamento da máquina de estados que controla o protocolo SIP. Por fim, é apresentado o funcionamento da nova biblioteca SIP com a componente pvPlayer Engine, componente que controla a reprodução na *framework* OpenCore.

4.1 Integração da SIP Stack eXosip na plataforma Android

Existem diversas *SIP stacks* (código-aberto) que podem ser utilizadas para a integração na plataforma Android. Grande parte destas SIP *stack*, dão suporte para várias plataformas (Windows, Mac, Linux, Solaris, etc). No seu código fonte, disponibilizam um ficheiro de configuração, que cria uma *Makefile* específica para a plataforma em questão. Infelizmente, embora a plataforma Android seja baseada em Linux, o compilador utilizado para compilar a plataforma não é *100%* idêntico e nem dispõe de todas as bibliotecas utilizadas pelos compiladores usados pelo Linux (exemplo: *gcc*). Assim sendo, pode ser necessário efectuar alterações ao código da *SIP stack*, para que esta compile correctamente com o compilador utilizado na plataforma Android.

Uma vez que podem existir incompatibilidades de bibliotecas, a escolha da *SIP stack* deve convergir para uma *SIP stack* o mais simples possível, baixando desta forma a probabilidade de obter problemas com as bibliotecas utilizadas pela *SIP stack*. O número de *SIP stacks* escritas em *C/C++* código-aberto é grande, no entanto, existem duas que se sobressaem em relação às outras: PJSIP[39] e eXosip[12]. Ambas são bastante utilizadas e existe muita documentação sobre elas.

A PJSIP é GNU General Public License (GPL) e tem uma documentação bastante organizada e de agradável leitura. A eXosip é GNU General Public License (GPL) v2 e também apresenta uma boa documentação, no entanto, é mais simples que a PJSIP. O facto de ser mais simples, não tendo tantas componentes, torna-a uma escolha favorável para a sua integração na plataforma Android. As suas referências são boas, existem diversos projectos a utilizar a OSIP*stack*, entre os quais, o software de VoIP Linphone. Assim sendo, a *SIP Stack* a ser utilizada na biblioteca SIP a integrar na plataforma Android, será a eXosip.

4.1.1 Arquitectura da SIP stack eXosip

A Figura 4.1, apresenta a arquitectura da *SIP stack* OSIP com a sua extensão eXosip. A arquitectura apresentada é constituída por quatro camadas: camada aplicacional de alto nível eXosip; camada aplicacional; camada de configuração; e, por fim, camada funcional.

A camada funcional contém todos os serviços básicos para a comunicação e estabelecimento de uma sessão SIP. Nesta camada, encontramos as componentes que fazem uso dos protocolos de transporte suportados pelo SIP (UDP, TCP), *Network handler*, *SIP factory*, *SIP transaction* e *SIP server location*.

As componentes que fazem uso dos protocolos de transporte (UDP, TCP), são utilizadas para receber as mensagens SIP e passar as respectivas mensagens ao *Network handler*. Por sua vez, o *Network handler* tem como função o encapsulamento ou desencapsulamento das mensagens SIP. Isto é, recebe as mensagens das componentes de transporte e passa-as à componente de *parser* (*SIP parser*). A componente *Network handler* também é utilizada para enviar as mensagens SIP recebidas pela componente *SIP transaction*. A *SIP transactions*, por sua vez, valida os cabeçalhos dos pedidos e das respostas SIP. A localização do servidor SIP, necessário para o estabelecimento da sessão e envio das mensagens, é feito através da componente SIP *server location*. Esta componente é responsável por fazer pesquisas ao registo DNS dos clientes SIP, para que seja possível estabelecer a ligação com o cliente pretendido. Por fim, a componente *SIP*

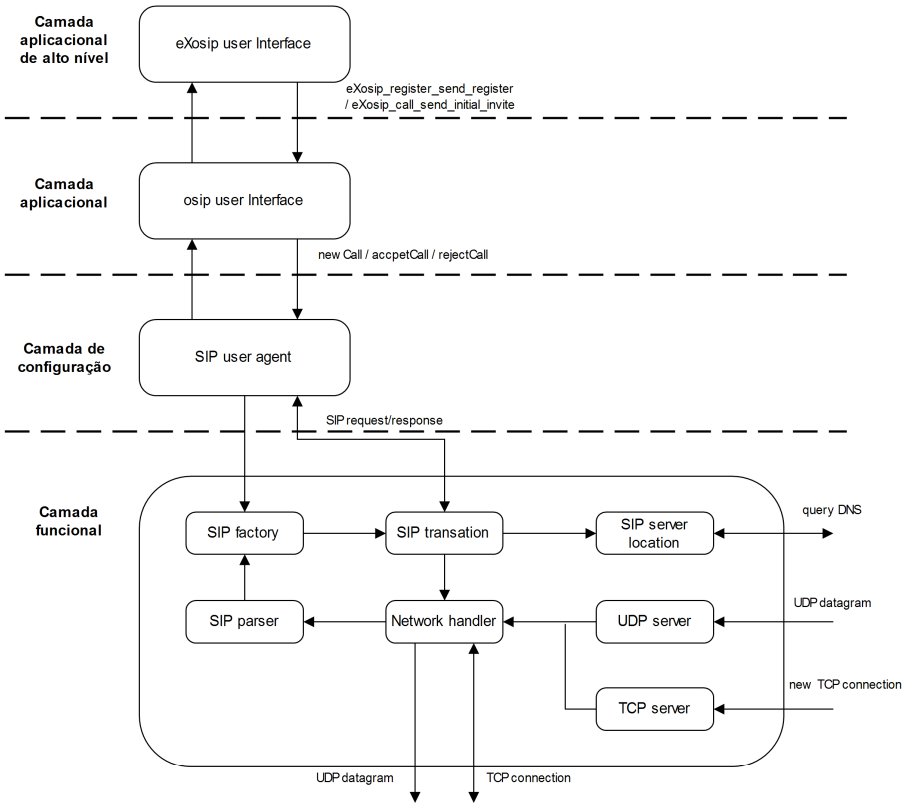

Figura 4.1: Arquitectura da SIP stack eXosip

factory disponibiliza os métodos para a criação e inicialização dos objectos desta camada.

A camada de configuração contém a componente *User agent*, que é responsável por efectuar a comunicação entre a aplicação do utilizador (através da interface disponibilizada na camada superior) e as componentes da *stack* de comunicação. A componente *User agent*, é invocada sempre que é necessário enviar um pedido SIP, ou sempre que é recebida uma mensagem SIP na *stack*. A sua estrutura interna é composta por uma máquina de estados, que controla a recepção e o envio de mensagens SIP. Com base no seu estado, através da sua API, é possível à aplicação o envio de mensagens SIP, ou a definição de *callbacks* para a recepção de pedidos SIP externos (exemplo: convite para inicializar uma sessão, confirmação de inicio de sessão, etc).

Por fim, no topo da arquitectura, são apresentadas duas camadas aplicacionais. Uma camada aplicacional de mais "baixo nível", que fornece a interface para a manipulação da *SIP stack* OSIP, agrupando todos os métodos necessários para a manipulação do protocolo SIP. E uma camada de mais alto nível, denominada eXosip, que fornece a interface de alto nível para a *SIP stack* OSIP.

Esta Interface de alto nível, tem como objectivo diminuir a complexidade de utilização da *SIP stack* OSIP. A sua estrutura tem como base a interface da camada inferior, disponibilizando uma API com métodos para a criação, inicialização e transmissão de mensagens SIP. A sua API pode ser consultada a partir da referência [12].

4.1.2 Integração da eXosip na plataforma Android

Existem dois métodos possíveis para a integração da eXosip *stack* na plataforma Android: compilando o código fonte da eXosip *stack* para a arquitectura ARM usando o compilador *gcc* do Linux, ou integração do código fonte da eXosip *stack* directamente no código fonte da plataforma Android.

A primeira opção, implica a geração de uma biblioteca da eXosip *stack*, que posteriormente é invocada pela aplicação. A segunda opção consiste em adicionar o código fonte da eXosip *stack* ao código fonte da plataforma Android, criar um *Android.mk* e adicionar esse *Android.mk* ao *Makefile* principal, para que seja compilado.

Neste trabalho será utilizada a segunda opção. Embora a primeira opção seja mais rápida, a segunda opção é mais "correcta". Ao usar a primeira opção, apenas se está a adicionar a biblioteca à plataforma. Se posteriormente for necessário efectuar alguma alteração à eXosip *stack*, é necessário voltar a descarregar o código fonte da eXosip, efectuar as alterações, voltar a compilar o código, e por fim, colocar novamente a biblioteca na plataforma. Se o código fonte da nova componente OSIP(*stack*), for integrado lado a lado com o código fonte das restantes componentes da plataforma, se posteriormente for sendo necessário efectuar alterações à *SIP stack* ou a componentes da *SIP stack*, basta fazer *make* da respectiva componente. Por sua vez, o *Makefile* da plataforma irá compilar todas as componentes necessárias para que essa alteração funcione correctamente.

Copiado o código fonte da OSIP *stack* para a directoria *"/external / opencore / protocolos / rtsp_client_engine/"*, é necessário alterar o *Android.mk* da *rtsp_client_engine* para que este compile o código da OSIP *stack*. Adicionado os ficheiros *".c"* da *stack* ao parâmetro *LOCAL_SRC_FILES* e as directorias que contém os ficheiros *".h"* ao parâmetro *LOCAL_C_INCLUDES*, ao correr o comando *make* para compilação da plataforma, o *Android.mk* da biblioteca *libpvrtsp_cli_eng_node* irá compilar o código da OSIP *stack*.

Para que a OSIP *stack* compilasse correctamente, foi necessário efectuar várias alterações ao seu código fonte. Na OSIP *stack* foi necessário remover a *package sys/unistd.h*, pois não existe na

plataforma Android, e as suas funcionalidades não são necessárias para a plataforma em questão. No caso da camada eXosip, foi necessário adicionar um novo método de comparação para testar se uma *string* estava inicializada ou não. Esta *string* é inicializada com todos os seus caracteres com o caractere *'\0'*. Na sua versão original, para fazer a comparação é feita a operação *tcp_firewall_ip!='\0'*. O *tcp_firewall_ip* é um apontador para o *array* de carateres e ao efectuar a operação é feito o teste se o primeiro elemente é diferente do caratere *'\0'*. No entanto, o compilador do Android dá um erro nesta expressão, dizendo que o apontador *tcp_firewall_ip* nunca seria *null*. Para resolver o problema é necessário fazer uma comparação ao *array* para confirmar se foi alterado. Esta solução é bastante menos eficiente do que a expressão *tcp_firewall_ip!='\0'*, a qual funcionava correctamente no compilador *gcc v4* do Linux. No entanto, esta comparação apenas é feita uma vez na inicialização da *stack*, o que torna a sua complexidade computacionalmente insignificante no funcionamento da pilha protocolar SIP.

O código fonte da OSIP *stack* é disponibilizado para diferentes plataformas. Este factor implica a existência de um *script* de configuração antes da compilação do código fonte. Este *script* testa quais as bibliotecas disponíveis no sistema e qual o tipo de sistema ao qual se vai compilar o código, gerando o respectivo *Makefile* com as configurações necessárias para a compilação do código fonte no sistema operativo em questão. Corrigidos todos os problemas de compilação, é necessário activar as *flags* necessárias para que o código funcione correctamente na plataforma Android. Uma vez que, o ficheiro de configuração, apenas tem suporte para plataformas Linux, Windows e Mac OS, é necessário ver quais as *flags* necessárias para a sua compilação num sistema operativo Linux. Assim sendo, activando essas *flags* estaticamente no *Android.mk*, foram adicionadas as *flags* aos paramentos *LOCAL_CFLAGS* e *LOCAL_LDFLAGS*.

4.1.3 Utilização do padrão Facade e Adapter

Como descrito em 3.4.2, para a integração da eXosip na arquitectura a desenvolver, será criada uma classe Adapter para implementar o padrão Adapter. O padrão Adapter, é utilizado para adaptar a interface de uma classe. O seu objectivo prático consiste em possibilitar a manipulação de classes com interfaces diferentes a partir de uma única interface [42]. A Figura 4.2 apresenta uma figura ilustrativa do padrão Adapter.

O padrão Adapter é crucial para garantir a possibilidade de alteração da *SIP stack*, garantindo que a componente que evoca o Adapter não necessita de sofrer alterações. Com base neste padrão, para efectuar uma alteração à *SIP stack* utilizada, apenas é necessário efectuar alterações

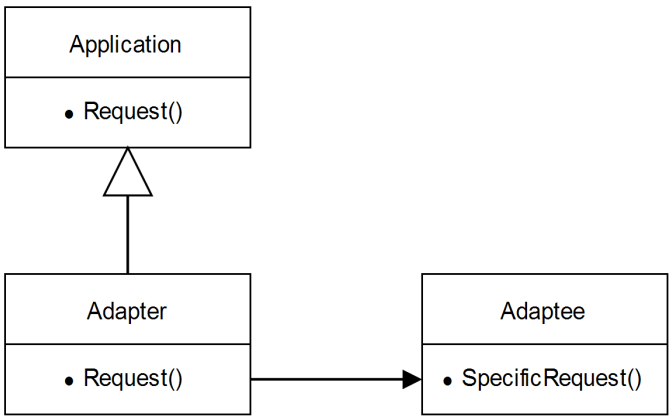

Figura 4.2: Figura ilustrativa do padrão Adapter [6]

à classe Adapter. Desta forma, a *SIP stack* é transparente à restante arquitectura a implementar.

Por sua vez, a camada superior da *SIP stack* eXosip utiliza o padrão Facade. O padrão Facade tem como objectivo agrupar um conjunto de classes com diversas funcionalidades numa única classe, disponibilizando métodos simples para manipulação desse conjunto de métodos das várias classes [41]. A aplicação deste padrão no topo da arquitectura OSIP *stack*, tem como objectivo facilitar a sua utilização, disponibilizando uma interface simples para a utilização das funcionalidades da *SIP stack*. A Figura 4.3 apresenta uma figura ilustrativa do padrão Facade.

4.1.4 Verificação funcional

Compilado o código fonte da eXosip *stack* e as componentes de auxílio à sua integração na arquitectura, é necessário testar se tudo está a funcionar correctamente. Para testar a eXosip portada para a plataforma Android, foi feito inicialmente um teste simples. Este teste consiste em invocar as funcionalidades da *SIP stack* juntamente com a invocação das funcionalidades do protocolo RTSP. Assim sendo, o objectivo é, ao iniciar a negociação do protocolo RTSP, vai ser também registado e enviado um convite para uma sessão SIP através da evocação da eXosip *stack*. Para efectuar este teste será utilizado o servidor SIP Trixbox e o dispositivo HTC Magic com o Sipdroid instalado.

Como definido na secção de descrição da arquitectura do protocolo RTSP, o protocolo RTSP é manipulado na componente RTSPEngineNode. Para evocar as funcionalidades SIP em simultâneo das funcionalidades RTSP, é necessário evocar a *SIP stack* na mesma função em que

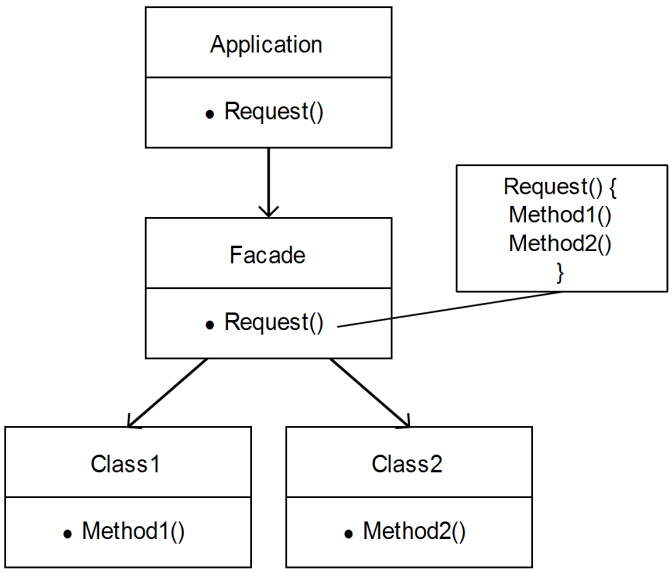

Figura 4.3: Figura ilustrativa do padrão Facade

é evocada a funcionalidade de enviar o pedido RTSP ao servidor. Assim sendo, foi criado o objecto Adapter, que implementa o padrão Adapter para a eXosip, no construtor da componente RTSPEngineNode, onde, posteriormente, é adicionada a evocação dos seus métodos de inicialização e convite para sessão no método *DoInitNode(PVRTSPEngineCommand &aCmd)* da componente RTSPEngineNode.

Efectuadas as alterações, evocando o *Media Player* do Android passado uma *URL* RTSP podemos observar que ao inicializar a stream RTSP a eXosip *stack* se está a registar correctamente no servidor e está a ser recebido o convite para início de sessão no Sipdroid do HTC G1. Para confirmar que tudo está a funcionar correctamente, através de uma captura de pacotes utilizando o Wireshark[43], é possível observar-se a troca de mensagens SIP e a inicialização da sessão RTSP em simultâneo.

Com os testes efectuados, pode-se concluir que a eXosip foi portada correctamente para a plataforma Android e que está a funcionar correctamente. O próximo passo, passa pelo desenvolvimento das componentes de manipulação da sessão SIP. Estas componentes têm que ser integradas na actual arquitectura da *framework* Opencore, e têm que disponibilizar as funcionalidades de recepção da *stream* para a sua reprodução nas componentes internas da *framework* Opencore.

4.2 Integração do protocolo SIP na Framework OpenCore

Ao longo desta secção é apresentada, de forma prática, a integração do protocolo SIP e suas componentes. Inicialmente, é descrito como efectuar o registo da nova biblioteca SIP no registo das bibliotecas de *streaming* da *framework*. Após descrever o registo, é apresentada a arquitectura e a descrição detalhada das componentes da biblioteca SIP. Dando continuidade à descrição das componentes, é apresentado o funcionamento da máquina de estados que controla a execução do protcolo SIP na componente SIPEngineNode, e, por fim, é descrito o funcionamento final da nova biblioteca SIP com o actual pvPlayer Engine da *framework* OpenCore.

4.2.1 Registo da biblioteca SIP

Para que a nova biblioteca SIP funcione na actual *framework* OpenCore, é necessário adicionar esta nova componente ao seu registo. Este registo será mais tarde consultado e a biblioteca carregada dinâmicamente pela *framework*, dando a esta possibilidade de utilizar o protocolo SIP na reprodução de elementos multimédia que façam uso deste protocolo. A verificação do tipo de *media* a reproduzir, é feito na classe playerdrive e tem como base a informação (endereço SIP) passada ao reprodutor multimédia do Android.

Para que o formato SIP seja reconhecido na verificação da classe *playerdrive*, é necessário definir um nome que identifique as streams SIP na lista de formatos disponíveis na *framework*. De seguida é necessário alterar o *parser* da classe playerdrive, para que este consiga associar um determinado endereço SIP ao identificador SIP definido nos formatos suportados.

Definido o identificador SIP, é necessário registar a nova biblioteca SIP no registo de bibliotecas de *stream* suportadas. Este registo é utilizado para carregar dinâmicamente as diferentes bibliotecas, consoante o tipo de *media* a reproduzir. Para que seja possível carregar as várias bibliotecas em tempo de execução, existe um ficheiro (pvplayer.cfg) de mapeamento, onde a cada biblioteca física no sistema está associado um determinado código. Este código, por sua vez, é registado e associado ao identificador de formato SIP. Através desta associação, sempre que é necessário correr uma *stream* SIP, com base no identificador SIP, é consultado o registo de bibliotecas suportas e é identificada e carregada dinâmicamente em tempo de execução, a respectiva biblioteca para manipulação do protocolo SIP.

O carregamento da biblioteca SIP dinâmicamente, implica que se implemente uma interface de carregamento para que o StreamingNode a consiga instanciar de forma genérica. As-

sim sendo, respeitando os nomes padrão das presentes bibliotecas na *framework* OpenCore, é necessário criar uma classe *pvmf_smnode_sipunicast_plugin_registry_populator_interface.cpp*. Esta estende a classe PVMFSMPluginLibManager e reescreve os métodos de carregar e destruir a componente SIP, LoadLibAndCreatPlugin e DeletePluginAndUnloadLib, respectivamente. Esta interface de carregamento e instanciação, evoca o *factory* (PVMFSMSIPUnicastNodeFactory) responsável pela instanciação da componente SIP. A evocação do *factory* é feita através da interface SIPUnicastManagerPluginInterface, que, por sua vez, estende a interface genérica (SMNodeFSPSharedLibraryInterface) para evocação dos vários *factories*.

A Figura 4.4, apresenta o diagrama de carregamento e instanciação das bibliotecas de stream.

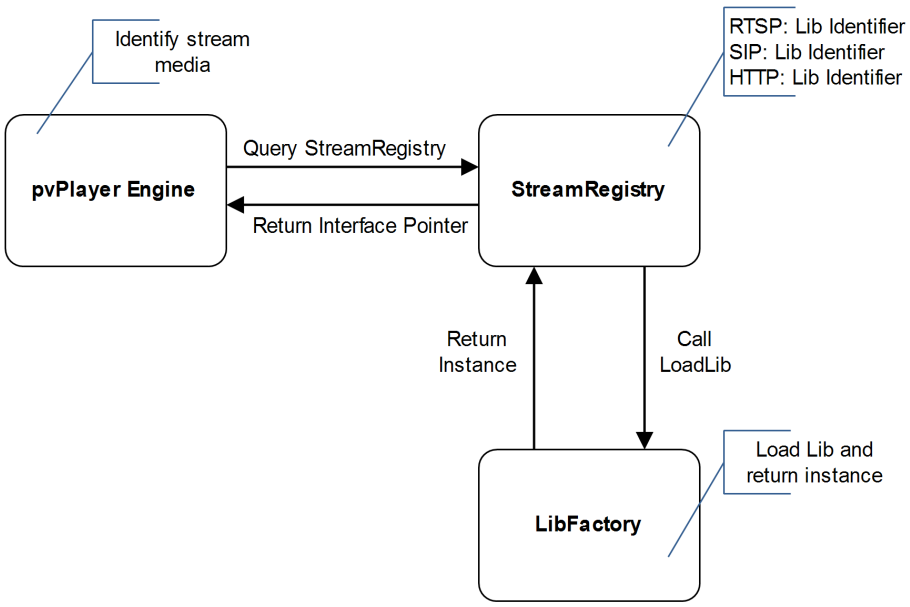

Figura 4.4: Diagrama de carregamento e instanciação das bibliotecas de stream.

4.2.2 Arquitectura e descrição das componentes da biblioteca SIP

No capítulo 3.4.1, foi apresentada a arquitectura do ponto de vista da sua integração com a *framework*, descrevendo de forma abstracta os seus principais componentes. Ao longo desta secção, será apresentada a arquitectura de forma mais detalhada, sendo dada uma descrição da implementação, de um ponto de vista mais técnico, de cada componente em particular.

A Figura 4.5, apresenta a arquitectura das componentes utilizadas pela biblioteca SIP no

estabelecimento e recepção da *media* durante uma sessão SIP.

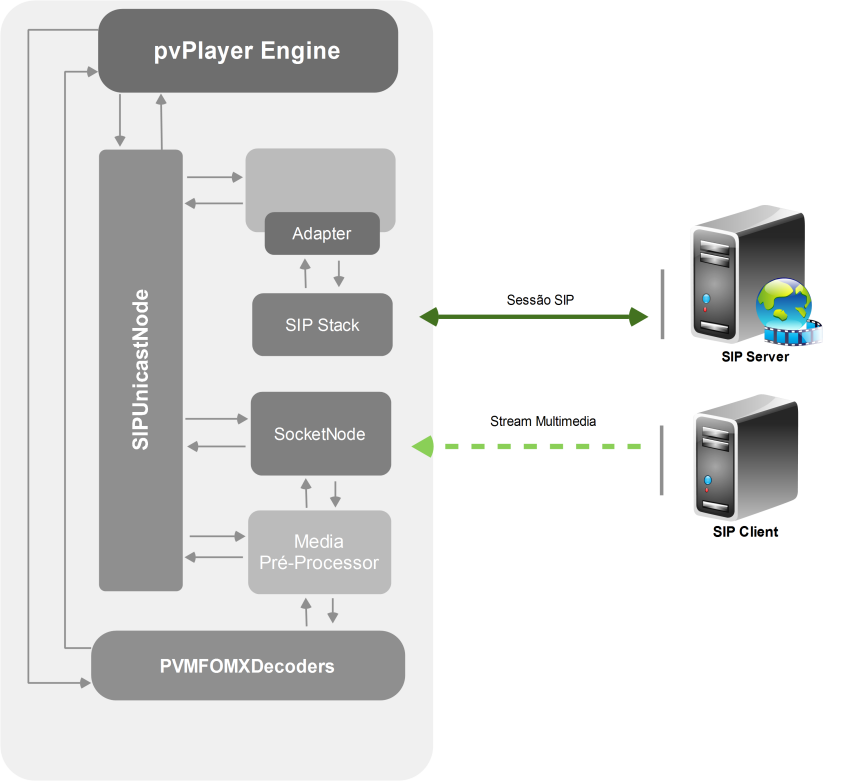

Figura 4.5: Componentes utilizados pela biblioteca SIP

Como referido em 3.4.2, a componente SIPUnicastNode funciona como uma interface para o pvPlayer Engine manipular o protocolo SIP. Para que o pvPlayer Engine seja capaz de manipular a componente SIPUnicastNode, é necessário que este implemente as interfaces utilizadas pelo pvPlayer Engine para manipulação das componentes que funcionam como fonte de dados. Assim sendo, é necessário que a nova componente implemente um conjunto de interfaces. Um exemplo dessas interfaces é a *PVMFNodeInterface*, que obriga a implementação de métodos de controlo (*Start()*, *Stop()*, *Pause()*, *Reset()*, *CancelAllCommands()*). Devido à necessidade de cada componente de fonte de dados implementar um conjunto de interfaces, essas interfaces estão agrupadas numa única interface base chamada *PVMFSMFSPBaseNode*. Esta interface implementa de forma genérica alguns dos seus métodos base, e força a implementação dos restantes métodos que são considerados necessários à componente de fonte de dados, para que o pvPlayer Engine a possa manipular.

A implementação dos métodos impostos pelas interfaces do PVMFSMFSPBaseNode, con-

sistem muito abstractamente na criação das componentes da biblioteca SIP e no mapeamento das funcionalidades evocadas pelo pvPlayer Engine. Isto é, a cada funcionalidade ou acção que o pvPlayer Engine pretenda executar sobre a fonte de dados, corresponde uma ou mais acções a executar nas componentes SIP. Este mapeamento é feito na implementação dos métodos das respectivas interfaces no SIPUnicastNode a serem evocados pelo pvPlayer Engine.

Por sua vez, a componente SIPEngineNode estabelece e controla a sessão SIP através da evocação de métodos básicos impostos pela interface PVMFNodeInterface, a partir da qual a componente SIPUnicastNode manipula a SIPEngineNode. Além da interface PVMFNodeInterface, a componente SIPENgineNode implementa a interface PVSIPEngineNodeExtensionInterface e a SIPCallBackEngine.

A interface PVSIPEngineNodeExtensionInterface impõem a implementação dos métodos para a configuração dos dados relativos ao registo SIP e à informação SDP transmitida e recebida durante a negociação SIP. A interface SIPCallBackEngine impõem a implementação dos métodos de *callback* da SIP *stack*, permitindo assim, a devolução de mensagens assíncronas recebidas pela SIP *stack*.

A comunicação com a SIP *stack*, é feita por intermédio de um adaptador. Esta componente adaptador, mapeia métodos básicos SIP (Register, Invite, Bye) na evocação dos respectivos métodos que evocam essas funcionalidades na SIP *stack*. Com a utilização desta componente Adapter é possível mudar a SIP *stack* utilizada, sem a necessidade de efectuar alterações no SIPEngineNode, sendo apenas necessário alterar os métodos básicos do adaptador para evocar as funcionalidades do respectivo método na SIP *stack*.

Além da manipulação da SIP *stack* para o envio das mensagens SIP, a componente SIPEngineNode implementa uma máquina de estados que controla a integridade da sequência das mensagens SIP. Isto é, controla a tentativa de envio de uma mensagem de BYE antes do envio ou recepção de uma mensagem INVITE. Posteriormente em 4.2.3 será descrita mais detalhadamente a máquina de estados do SIPEngineNode e quais as acções ou mensagens que podem ser executadas em cada estado.

Após a negociação da sessão através da componente SIPEngineNode, é devolvido à componente SIPUnicastNode a porta na qual será recebida a *media*. Recebida a informação da porta a receber a *media*, a componente SIPUnicastNode configura a componente SocketNode para abrir a respectiva porta.

A componente SocketNode, tem um funcionamento muito simples. A sua função consiste em

receber a *media* do *socket*, alocar memória para guardar a informação recebida e enviar a *media* para a componente de pré-processamento. Para fornecer as suas funcionalidades, disponibiliza métodos à componente SIPUnicastNode para a sua criação e controlo dos *sockets*. A alocação de memória para guardar a informação recebida, utiliza a estrutura OsclRefCounterMemFrag, a qual fornece uma gestão optimizada dessa memória. Recebida a informação do *socket* e colocada na estrutura OsclRefCounterMemFrag, esta é passada a componente de pré-processamento da *media*.

A componente de pré-processamento da *media*, tem como intuito retirar o fragmento de *media* e a respectiva informação (*metadata*) do pacote RTP recebido da componente SocketNode. Feito o *parser* do pacote RTP, é alocada a estrutura PVMFSharedMediaDataPtr para guardar o fragmento de *media* e a respectiva *metadata*. A alocação da estrutura PVMFSharedMediaDataPtr é feita a partir de um conjunto de memória pré-alocada, a qual é obtida a partir da componente PVMFMediaFragGroupCombinedAlloc. Esta componente fornecida pela *framework* OpenCore, disponibiliza métodos de alocação e gestão de memória eficiente. Uma maior descrição do funcionamento da componente PVMFMediaFragGroupCombinedAlloc, foi previamente descrita em 3.3.2.

Colocada a *media* na estrutura PVMFSharedMediaDataPtr, esta é passada à componente de pré-processamento para efectuar o pré-processamento do *payload* da *media*. Este pré-processamento tem como objectivo fazer uma pre-verificação de possíveis fragmentos de *media* com erros. No caso, do fragmento conter erros, então o fragmento é automaticamente descartado, caso contrário o fragmento é enviado para a componente de descodificação.

O envio dos fragmentos de *media* para os descodificadores é feito através da interface PVMFPortInterface, a qual é implementada pela componente de pré-processamento. A implementação da interface PVMFPortInterface, implica a implementação dos métodos de configuração e envio dos fragmentos para os descodificadores. A configuração da porta de comunicação com os descodificadores, é feita a partir da pvPlayer Engine por intermédio da SIPUnicastNode.

A interface PVMFPortInterface, além de disponibilizar métodos de configuração e envio dos fragmentos de *media*, disponibiliza também métodos de controlo do envio da *media*, informando a componente do estado da porta para o envio dos fragmentos de *media*. A componente de pré-processamento utiliza esses métodos de controlo para saber se os descodificadores podem receber fragmentos de *media* ou se já tem o *buffer* cheio.

4.2.3 Funcionamento da máquina de estados de controlo do protocolo SIP

Como referido anteriormente, a componente SIPEngineNode utiliza uma máquina de estados para controlo do protocolo SIP. A cada estado é associado um conjunto de acções, podendo estas ser de configuração ou de recepção/envio de mensagens SIP. As acções de cada estado implementam uma determinada lógica aplicacional, impedindo a execução de acções de um determinado estado sem primeiro executar as acções dos seus estados antecedentes.

A Figura 4.6, apresenta a máquina de estados utilizada pela componente SIPUnicastNode para controlo do protocolo SIP.

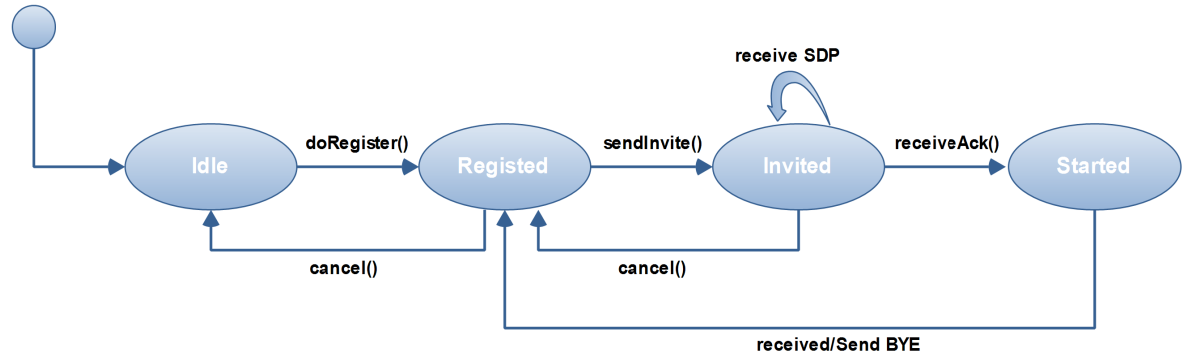

Figura 4.6: Máquina de Estados da componente SIPEngineNode para controlo do protocolo SIP

Como se pode observar pela Figura 4.6, a máquina de estados é composta por quarto estados: *Idle*, *Registed*, *Invited* e *Started*.

O estado Idle é o estado inicial, sendo obtido quando a componente SIPEngineNode é criada. Para iniciar a execução e estabelecimento das sessões SIP, é necessário efectuar o registo no servidor SIP. O registo é feito através do método doRegister(), o qual é evocado à componente Adaptar, que, por sua vez, evoca os métodos de registo da SIP *stack*. Devolvido o resultado de sucesso do registo no servidor SIP, a máquina de estados transita para o estado Registed, caso contrário, mantém o estado Idle.

No estado Registed, a componente SIPUnicastNode tem duas acções possíveis: cancelar a sessão SIP ou enviar um convite a um cliente SIP para iniciar uma sessão. Caso a acção escolhida seja cancelar a sessão SIP, a máquina de estados transita para o estado de Idle novamente, sendo o registo libertado posteriormente por *timeout* no servidor SIP. Caso a acção escolhida seja enviar um convite a outro cliente SIP, é evocado o método sendInvite() ao componente Adapter, que, por sua vez, utiliza a SIP *stack* para o envio da mensagem INVITE. Enviada a mensagem, a

máquina de estados transita para o estado Invited, ficando a aguardar a resposta do outro cliente com o SDP da *media* a receber. Recebida a informação do SDP é devolvida ao SIPUnicastNode.

No estado Invited, após a recepção do SDP é possível executar duas acções: cancelar o convite ou devolver o *Ack* de confirmação de início de sessão. A acção de cancelamento do convite, pode ser tomada por iniciativa da componente SIPUnicastNode ou por *timeout* da acção de recepção do SDP do cliente SIP. A acção de cancelamento do estado Invited, consiste na evocação do método BYE à componente Adapter que, por sua vez, evoca o envio da mensagem BYE na SIP *stack* e na transição da máquina de estados novamente para o estado Registed. A acção de confirmação de início de sessão, consiste na evocação do método Ack da componente Adapter que através do SIP *stack* envia a mensagem Ack ao cliente SIP, transitando a máquina de estados para o estado Started.

O estado Started, é o estado em que se encontra o SIPEngineNode durante a transmissão multimédia da sessão SIP. Durante o estado Started, a componente SIPEngineNode aguarda pelo evento do componente Adapter a informar que a sessão SIP foi cancelada pelo outro Cliente SIP, ou pela acção de termino da sessão por parte da componente SIPUnicastNode. No caso da recepção do evento a informar fim de sessão, a componente SIPEngineNode envia através do componente Adapter a mensagem BYE a confirmar fim de sessão e transita para o estado Registed informando o SIPUnicastNode. No caso do pedido de execução da acção de término da sessão SIP por parte do SIPunicastNode, o SIPEngineNode envia através da componente Adapter a mensagem BYE para informar o outro cliente SIP da intenção de terminar a sessão. Após receber a confirmação de fim de sessão do cliente SIP, a máquina de estados transita para o estado Registed.

De volta ao estado Registed, a componente pode executar novamente as acções de Invite, transitando novamente para os estados de sessão (Invite, Started), podendo voltar ao estado Idle ou simplesmente ficar no estado Registed a aguardar instruções da componente SIPUnicasNode e eventos da SIP *stack*.

4.2.4 Funcionamento da biblioteca SIP com o pyPlayer Engine

Como definido em 3.2.3 o pvPlayer Engine funciona com base numa máquina de estados, onde a cada estado estão definidas acções que são realizadas por diversas componentes. Com base na definição da máquina de estados da componente SIPEngineNode, apresentada anteriormente em 4.2.3, ao longo desta subsecção é apresentada uma associação entre a máquina de estados

do pvPlayer Engine e a máquina de estados do SIPEngineNode, serão descritas quais as acção a serem executadas e o comportamento das várias componentes da biblioteca SIP para cada estado da pvPlayer Engine. A Figura 4.7, apresenta de forma genérica o funcionamento do pvPlayer Engine e a interacção com as componentes descritas ao longo desta subsecção.

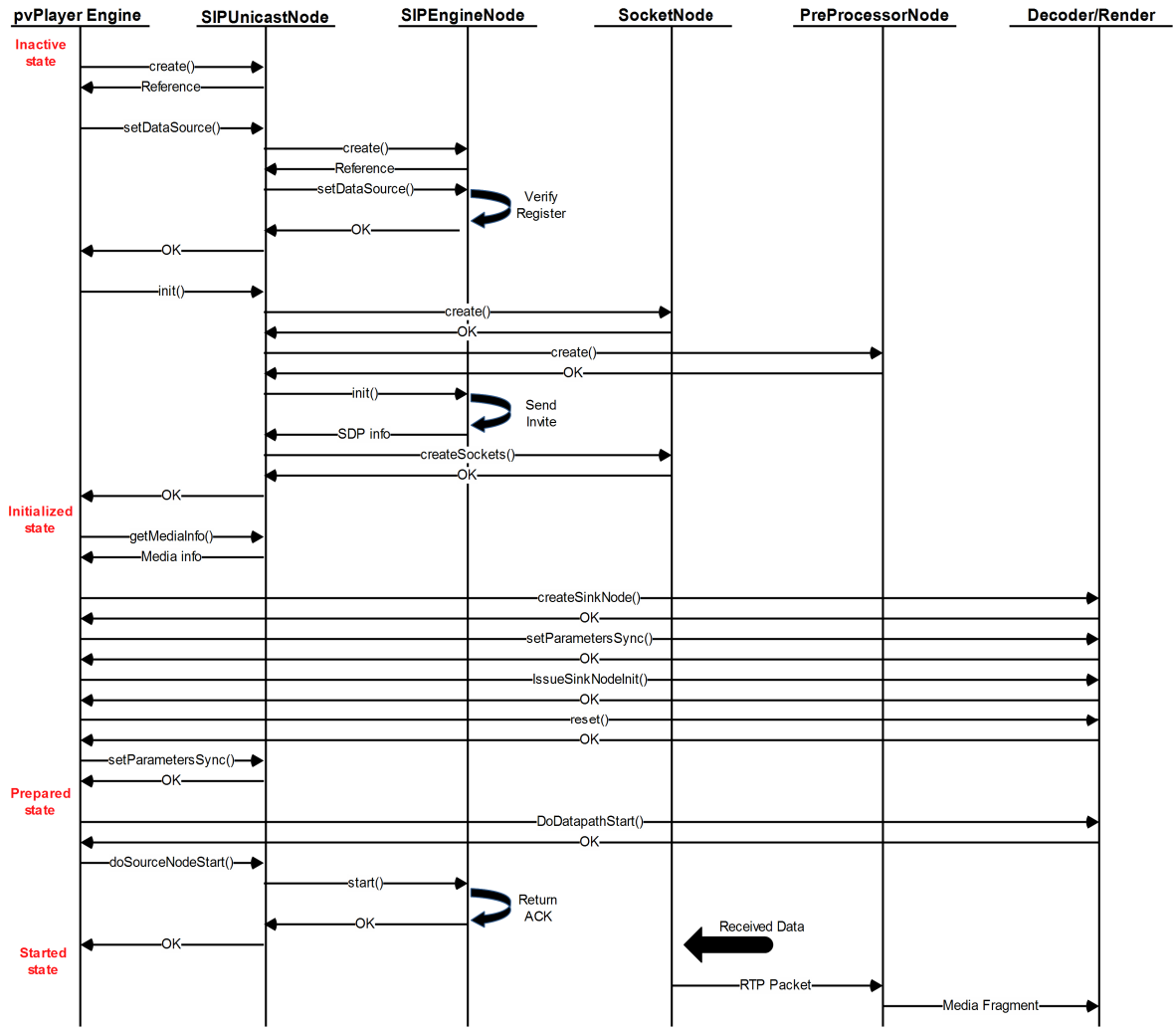

Figura 4.7: Diagrama de sequência do pvPlayer Engine

Seguindo a descrição dada em 3.2.3, durante a inicialização do serviço do PVPlayer SDK, a máquina de estados do pvPlayer Engine é inicializada no estado Inactivo. No estado inactivo, o pvPlayer Engine identifica o tipo de *media* a reproduzir e inicializa o respectivo nodo para fornecer os dados. Na situação em particular, em que se pretende estabelecer um sessão SIP (sendo passado ao PVPlayer um endereço SIP), o pvPlayer Engine com base no registo e

no StreamingNode inicializa a biblioteca SIP, a qual devolve uma referência da instântcia da componente SIPUnicastNode.

Instanciada e obtida a referência para a componente SIPUnicastNode, o pvPlayer Engine evoca o método setDataSource(), o qual defini no SIPUnicastNode o endereço SIP do cliente a estabelecer a sessão e a informação relativa ao registo no servidor SIP. A evocação do método setDataSource() na componente SIPUnicastNode, implica a instanciação da componente SIPEngineNode. Durante a instanciação da componente SIPEngineNode, como referido em 4.2.3, é criada a sua máquina de estados que irá controlar a execução do protocolo SIP, sendo esta inicializada no estado Idle. Instanciada e retornada a referência para a componente SIPEngineNode, a SIPUnicastNode evoca o método setDataSource() da componente SIPEngineNode(), passando o enderenço do cliente SIP e a informação relativa ao registo no servidor SIP.

Evocado o método setDataSource() na componente SIPEngineNode, esta guarda o endereço do cliente SIP e inicia o registo no servidor SIP. O registo no servidor SIP é feito pela SIP *stack*, por intermédio do componente Adapter. A componente Adapter é instanciada, e, através da sua referência, é evocado o método register(), o qual recebe como parámetro a informação relativa ao registo (nome de utilizador, senha de acesso, endereço do servidor SIP). A componente Adapter, por sua vez, instancia a SIP stack através do seu *factory* e evoca os métodos de registo. A SIP *stack* utiliza um registo de *User Agents*, sendo que, caso ainda não exista nenhum *User Agent* criado para o registo no servidor SIP pretendido, cria um novo *User Agent* para se registar no servidor, caso contrário, utiliza o *User Agent* já registado.

Efectuado o registo com sucesso, é devolvido o estado de sucesso ao SIPEngineNode, onde, por sua vez, devolve sucesso da execução do método setDataSource() à componente SIPUnicastNode. A componente SIPUniscastNode, analogamente ao SIPEngineNode, também devolve sucesso na execução do método setDataSource() à componente pvPlayer Engine. Recebido sucesso na definição da fonte de dados, o pvPlayer Engine evoca o método Init() para a sua máquina de estado transitar para o estado de inicializado.

A evocação do método Init(), antes de transitar a máquina de estados do pvPlayer Engine para o estado inicializado, necessita inicializa a componente de fonte de dados. Essa inicialização é feita através da evocação do método Init() na componente SIPUnicastNode. A evocação do método Init() na componente SIPUnicastNode adiciona a lista de comandos o comando de inicialização, o qual será executado assim que houver disponibilidade para processamento desse comando por parte do dispositivo. Este sistema de comandos é utilizado pelas componentes da *framework* Opencore para repartir o processamento do dispositivo pelos diversos módulos em

execução.

Processado o comando de inicialização na componente SIPUnicastNode, este instanciará as componentes de recepção da *media* e de pré-processamento. Instanciadas as componentes, a componente SIPUnicastNode, evoca o seu método init() da componente SIPEngineNode, que, por sua vez, evoca o método sendInvite() da componente Adapter. A evocação do método sendInvite() da componente Adapter, evoca os métodos de envio do invite ao cliente SIP, definido anteriormente pelo método setDataSource(). Enviada mensagem INVITE ao clente SIP, a biblioteca SIP fica a aguardar uma resposta do cliente.

Recebida a resposta do cliente com o SDP a descrever a *media* a transmitir durante a sessão, a componente SIPEngineNode, devolve o SDP à componente SIPUnicastNode. Por sua vez, a componente SIPUnicastNode processa o SDP recebido, guarda a informação processada, cria os *sockets* na componente SocketNode para receber a *media*, e, por fim, devolve a confirmação de sucesso à componente pvPlayer Engine. Recebida a confirmação que a fonte de dados foi inicializada correctamente, o método Init() da componente pvPlayer Engine é finalmente concluído com sucesso e a máquina de estados transita para o estado de inicializado.

No estado inicializado, a componente pvPlayer Engine consulta a fonte de dados (SIPUnicastNode) para obter a informação relativa à *media* a receber, onde é devolvida a informação obtida a partir do processamento do SDP recebido. Recebida a informação da *media* a ser reproduzida, a componente pvPlayer carrega os respectivos componentes de reprodução da *media* através do método addDataSink(). Carregados os componentes, o pvPlayer Engine procede à configuração das componentes com base nas características da media a ser reproduzida através dos métodos SetParametersSync() e DoCapConfigSetParameters(). Configuradas as componentes de reprodução com sucesso, a componente pvPlayer Engine pode evocar o método prepare() para que a sua máquina de estado transite para o estado preparado.

A evocação do método prepare() na componente pvPlayer Engine, implica a inicialização e preparação dos *Sink Nodes* e da fonte de dados. Para a inicialização dos *Sink Nodes* é evocado o método IssueSinkNodeInit(), o qual tenta inicializar o *Sink Node*. Após inicialização do *Sink Node*, é feito um teste ao suporte do tipo de *media* a reproduzir, verificando se esta pode ser reproduzida directamente pelo dispositivo ou se necessita de um descodificador para descodificar a *media* antes da sua reprodução. Caso se verifique que a *media* pode ser reproduzida directamente, o pvPlayer Engine configura a porta de comunicação com a fonte de dados para recepção da *media* e faz reset() ao *Sink Node* limpando todo o possível conteúdo nos seus *buffers*. Caso se verifique que é necessário utilizar um descodificador para descodificar a *media* antes do seu envio

para reprodução, é carregado o descodificador recorrendo ao registo de *codecs* e configurando o *codec* com base nas características da *media* a ser reproduzida.

Inicializado e configurado o descodificador, o pvPlayer Engine configura a porta de recepção da *media* no descodificador para receber a *media* da fonte de dados e a porta de saída para enviar a *media* para as componentes de reprodução (*Sink Nodes*). Configuradas as portas do descodificador, são configuradas as portas do reprodutor e feito um *reset* a ambas as componentes para que sejam limpos os seus *buffer* internos. Configuradas as componentes de descodificação e reprodução, o pvPlayer Engine procede a configuração da porta de comunicação com o descodificado ou com o reprodutor através da evocação do método setParametersSync() da componente SIPUnicastNode, se o dispositivo tiver suporte para descodificação directa da *media*. Configurada a componente de fonte de dados, a máquina de estados do pvPlayer Engine transita para o estado de preparado.

No estado de preparado o pvPlayer Engine pode proceder à sua reprodução evocando o método start(). O método start(), por sua vez, evoca o método DoDatapathStart() e DoSourceNodeStart(). O método DoDatapathStart() informa as componentes de descodificação e reprodução, para iniciar a sua execução assim que receberem fragmentos de *media* nas suas portas de comunicação. O método DoSourceNodeStart(), adiciona o comando de Play à componente SIPUnicastNode, o qual, assim que seja processado, evocará o método start() na componente SIPEngineNode. Evocado o método start() no SIPEngineNode, através da componente Adapter, é devolvida a confirmação de início de sessão (*Ack*) ao cliente SIP. Devolvido o comando Ack, a máquina de estados da componente SIPEngineNode passa para o estado Started, sendo devolvida à componente SIPUnicastNode a confirmação de início de sessão. Por sua vez, a componente SIPUnicastNode, informa a componente pvPlayer Engine que a sessão foi iniciada. Recebida a confirmação da componente fonte e das componentes de descodificação e reprodução que iniciaram a sua execução, a máquina de estados da componente pvPlayer Engine transita para o estado de Reprodução.

Iniciada a transmissão dos fragmentos de *media* pelo cliente SIP, o componente SocketNode, recebe a informação no seu *socket*, guarda a informação na estrutura específica e coloca a informação na componente de pré-processamento da *media*. A componente de pré-processamento da *media*, recebe a informação do SocketNode e retira o fragmento de *media* e toda a *metadata* relevante do pacote RTP. Retirado o fragmento e a repectiva *metadata*, é feito um pré-processamento do *payload* do fragmento com o intuito de verificar a existência de erros. Caso o fragmento não contenha nenhum erro detectado no pré-processamento, este é colocado na porta de comunicação

com a componente de descodificação ou de reprodução para ser consumido.

Durante a recepção e reprodução da sessão multimédia, o pvPlayer Engine pode parar a reprodução evocando os métodos pause() ou stop(). No caso do protocolo RTSP, os métodos pause() e stop() têm funcionalidades distintas, como referido em 3.3.3. No caso do protocolo SIP, os dois métodos significam parar a transmissão, voltando a máquina de estados da componente SIPEngineNode para o estado Registed, a máquina de estados da componente pvPlayer Engine para o estado de Inicializado, sendo os descodificadores e reprodutores parados e limpados os seus *buffers* internos. Pode-se considerar, que para o protocolo SIP só faz sentido a existência do método stop(), não fazendo sentido parar temporalmente uma *stream* que está a ser transmitida em tempo real.

4.3 Sumário

Como foi possível constatar no decorrer deste capítulo, o compilador C/C++ utilizado para compilar o código fonte da plataforma Android tem algumas particularidades, obrigando a algumas alterações no código fonte da SIP *stack* eXosip. A biblioteca SIP foi integrada segundo a arquitectura definida no capítulo anterior. Embora já tendo conhecimento do funcionamento do pvPlayer Engine e da bibliotexa RTSP, a utilização das componentes de gestão de memória da *framework* OpenCore revelaram-se um problema na sua utilização. Este facto deveu-se à falta de documentação destas componentes.

Integrada a SIP *stack* e implementada a biblioteca SIP na *framework* OpenCore, o próximo passo será testar a biblioteca desenvolvida e avaliar os resultados obtidos.

Capítulo 5

Avaliação dos resultados

Ao longo deste capítulo é feita uma avaliação dos resultados da transmissão multimédia numa sessão SIP. Inicialmente é descrito o ambiente de teste, descrevendo os dispositivos e as tecnologias usadas. Após a descrição do ambiente de teste, são expostos os resultados de desempenho obtidos pela nova biblioteca SIP em relação à biblioteca RTSP. Por fim, embora não seja objectivo desta dissertação avaliar a qualidade de transmissão do vídeo, é feita uma análise da experiência do utilizador durante a transmissão e são expostas algumas situações a ter em conta durante a transmissão multimédia.

5.1 Ambiente de testes

Testar a nova biblioteca SIP desenvolvida no âmbito desta dissertação, envolve a construção de um ambiente de teste específico, requerendo algumas componentes externas à biblioteca SIP. Para testar o estabelecimento e a transmissão multimédia durante a sessão SIP, serão utilizadas as seguintes componentes: dispositivo móvel HTC G1[7], servidor SIP Trixbox e um servidor de *stream* SIP desenvolvivo em particular para o teste da biblioteca SIP. A Figura 5.1 apresenta o cenário de teste e a comunicação entre as suas componentes.

Como podemos observar na Figura 5.1, as componentes utilizadas no ambiente de testes comunicam entre si com o intuito de estabelecer uma sessão SIP de forma a negociar e iniciar a transmissão de uma *stream* multimédia. O dispositivo móvel HTC G1, primeiro dispositivo móvel com suporte da Google para o desenvolvimento de aplicações na plataforma Android, foi alterado para funcionar com a biblioteca SIP desenvolvida. A sua versão original do Android

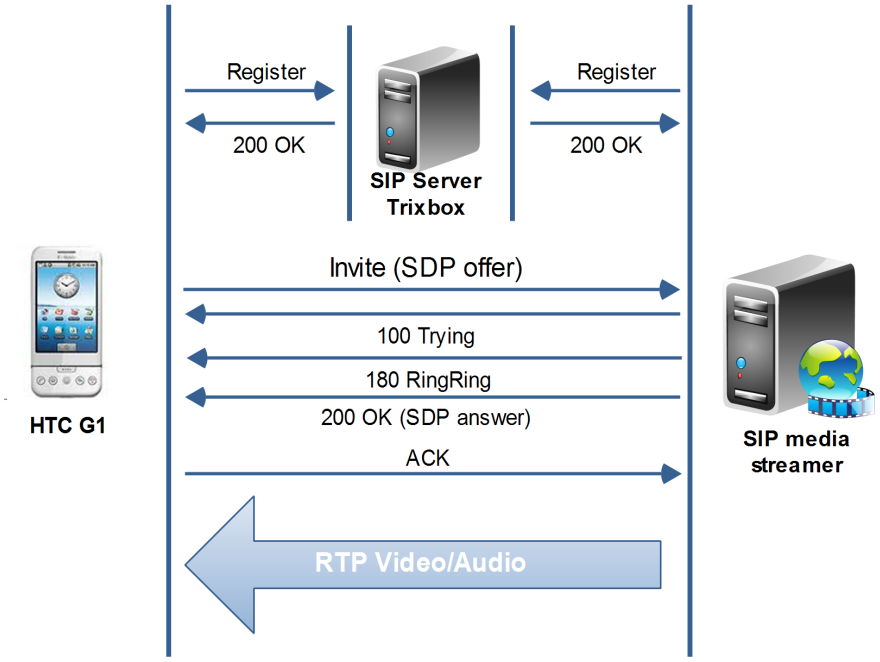

Figura 5.1: Ambiente de testes

1.5 foi alterada para a versão 2.0.1, a qual era a versão mais recente da plataforma Android no início do trabalho desenvolvido. Esta alteração implicou um processo um pouco trabalhoso: compilação da versão do Kernel Linux do dispositivo HTC G1 para a plataforma 2.0.1, associação desse Kernel ao código fonte da plataforma Android e finalmente compilação do respectivo código fonte utilizando a estrutura de compilação disponibilizada pela Google (Makefile e os seus respectivos .sh para sua criação e configuração).

A associação do Kernel ao código fonte é feita na directoria "vendor", sendo que esta directoria disponibiliza informação relativa aos dispositivos suportados pela plataforma, informação essa que é necessária para que a compilação seja efectuada correctamente para o dispositivo em questão. Compilado o código fonte, recorrendo ao comando "fastboot", foram enviadas as respectivas imagens da plataforma Android 2.0.1 para o dispositivo (boot.img, system.img e userdata.img). Após os envio das das imagens das várias partições geradas durante a compilação, o HTC G1 passará a funcionar com a nova versão 2.0.1 do Android. A tabela 5.1, apresenta as características técnicas do dispositivos HTC G1.

Por sua vez, o servidor SIP (Trixbox) utilizando no teste, é uma máquina virtual Linux com um servidor de registo SIP previamente configurado [44]. Uma vez que, a nível de servidor

Hardware	***Description***
Processor	Qualcomm® MSM7201A™, 528 MHz
Memory	ROM: 256 MB, RAM: 192 MB
Display	3.2-inch TFT-LCD flat touch-sensitive screen with 320 x 480 (HVGA) resolution
Network	HSPA/WCDMA: Up to 7.2 Mbps down-link (HSDPA) and 2 Mbps up-link (HSUPA) speeds
WiFi	Wi-Fi®: IEEE 802.11b/g - Texas Instruments WL 1251B network Chipset
Camera	3.2 megapixel color camera with auto focus

Tabela 5.1: Características técnicas do dispositivos HTC G1 [7]

de registo SIP se pretendia algo que simplesmente fornecesse um registos dos clientes em teste (HTC G1 e o streamer SIP), a escolha do Trixbox deveu-se ao facto de fornecer um sistema Linux já praticamente todo configurado, que disponibiliza uma interface de fácil utilização para registo dos clientes SIP. Feito do *download* da máquina virtual do Trixbox a partir da sua página da web, esta foi posta a correr numa computador utilizando o VirtualBox[1]. Ligada a máquina virtual, através do *browser* é possível aceder à interface web do servidor de registo SIP, ao qual foram adicionados dois registos.

Finalmente, para o servidor de *stream* SIP, foi desenvolvido um pequeno *streamer* para ser possível testar a biblioteca SIP desenvolvida. O *streamer* foi desenvolvido na linguagem C e utiliza a SIP *stack* eXosip para a comunicação com o servidor de registo SIP e para o estabelecimento da sessão SIP com o cliente. De forma geral, este pequeno streamer SIP, após se registar num servidor SIP e receber um pedido de Invite, carrega uma estrutura que contém a *media* previamente gravada, envia a resposta ao Invite com o SDP que contém a informação relativa à *media* a transmitir e após a recepção do ACK que confirma o inicio da sessão, inícia a transmissão da *stream* de *media*.

Para a *media* a transmitir, foram capturados e guardados numa estrutura os pacotes RTP transmitidos durante uma sessão RTSP. Esta captura foi feita durante uma sessão RTSP no dispositivo, onde foram guardados todos os pacotes RTP recebidos durante a *stream*. Para confirmar que nessa amostra não faltam pacotes, foi verificado se o número de pacotes da *media* no *streamer* RTSP coincidia com o número de pacotes gravados na estrutura, onde se pode confirmar que se tinha na estrutura todos os pacotes relativos a *media*. Com a *media* a transmitir gravada numa estrutura em ficheiro e com o *streamer* pronto a efectuar a transmissão dos pacotes para o cliente SIP, é necessário definir a frenquência de transmissão dos pacotes. Para garantir que os

[1] http://www.virtualbox.org/

pacotes são enviados correctamente, simulando um cenário em tempo real, o envio dos pacotes é feito com base no tempo referência do pacote RTP (*timestamp*). A descrição da *media* utilizada durante a transmissão é apresentada na tabela 5.2.

Vídeo	
Dimensões	320x176
Codec	H.264/AVC
Rácio de frames	13 frames por segundo
Áudio	
Codec	MPEG-4 AAC audio
Canais	Stereo
Rácio de amostra	16000Hz
Geral	
Duração	38 segundos

Tabela 5.2: Características da media utilizada durante a transmissão de teste.

Montado o ambiente de teste com todas as componentes necessárias para testar a biblioteca SIP desenvolvida, é possível registar a biblioteca SIP desenvolvida no servidor SIP Trixbox e estabelecer a sessão SIP entre o dispositivo e o *streamer*. Com todas as componentes do ambiente de teste a funcionar correctamente, o próximo objectivo é a verificação do funcionamento da nova biblioteca SIP. A biblioteca RTSP será utilizada como referência para verificar se as componentes que funcionam em conjunto com a biblioteca SIP estão a funcionar correctamente.

5.2 Avaliação de desempenho da biblioteca SIP

Existem diversos testes que podem ser feitos à biblioteca SIP desenvolvida: testes de qualidade de imagem, testes à *media* recebida e testes de desempenho. No entanto, sendo o propósito desta dissertação a integração do protocolo SIP na Media Framework do Android, com suporte para a reprodução de vídeo, o principal objectivo é garantir que a biblioteca desenvolvida cumpre todos os requisitos definidos e funciona correctamente. Portanto, para o trabalho em questão a qualidade da *media* ou a qualidade de recepção da *media* não serão os aspectos mais importantes, uma vez que as metodologias de alguns dos sistemas de avaliação destes aspectos é impraticável nos actuais dispositivos móveis devido à elevada necessidade de computação. Assim sendo, neste trabalho será essencialmente focado o teste funcional e o teste de desempenho da biblioteca SIP.

Como referido anteriormente na secção 3.4.3, se a biblioteca SIP estiver desenvolvida correctamente, o seu desempenho terá de ser muito idêntico ao desempenho da biblioteca RTSP. O problema é como testar as duas bibliotecas de forma idêntica e fiável. Uma forma de tentar ultrapassar este problema, é garantir que a *media* a reproduzir nas duas bibliotecas é exactamente a mesma. Obtidos os resultados de utilização do processador do dispositivo, para o processo de reprodução da *media* em cada instante, é possível comparar os valores entre as duas bibliotecas. Para obter estes resultados, foi desenvolvido um *script* que inicia com o envio do primeiro fragmento de *media* para o descodificador e termina após o *buffer* do reprodutor ser totalmente consumido.

O *script* consiste na execução do comando "top" de um em um segundo, o qual devolve a informação de utilização de processador de cada processo a correr. Com estes resultados, é possível construir um gráfico da utilização de processador em cada instante de reprodução da *media*, o qual pode ser utilizado para comprar o desempenho obtido nas duas bibliotecas.

5.2.1 Avaliação da biblioteca RTSP

O primeiro teste a ser feito será à biblioteca RTSP, ao qual foram efectuados dez testes à reprodução da *media* referida em 5.2. Os resultados obtidos nos testes efectuados são apresentados nos anexos A.1 e A.2, sendo que a tabela A.3 apresenta a média dos resultados obtidos para cada instante de reprodução. Uma vez que os resultados obtidos variam, esta tabela é utilizada na construção do gráfico que será dado como resultado da utilização do processador da biblioteca RTSP durante a reprodução da *media*. O cálculo desta média deve-se ao facto de não ser possível obter um valor exacto da utilização do processador para cada instante. No entanto, para aumentar a precisão do resultado, foi calculado o desvio padrão dessa média em relação aos resultados obtidos e com base nesse desvio padrão foi calculado um intervalo de confiança, que garante que em 95% dos dez testes efectuados os resultados estão entre esse intervalo. Estes resultados, bem como o resultado da média de utilização de processador durante a reprodução da *media* utilizando a biblioteca RTSP, são apresentados na Figura 5.2 e nos anexos A.3 e A.4.

Como se pode observar pelo gráfico apresentado na Figura 5.2, a utilização do processador durante a reprodução da *media* varia entre os 55% e os 80%. No entanto, excepcionalmente nos dois primeiros segundos a utilização do processador é máxima. Este facto deve-se ao arranque da execução das componentes de descodificação, reprodução e do *buffer* da biblioteca RTSP. Após o arranque das componentes, o processamento gasto na reprodução da *media* é essencialmente feito

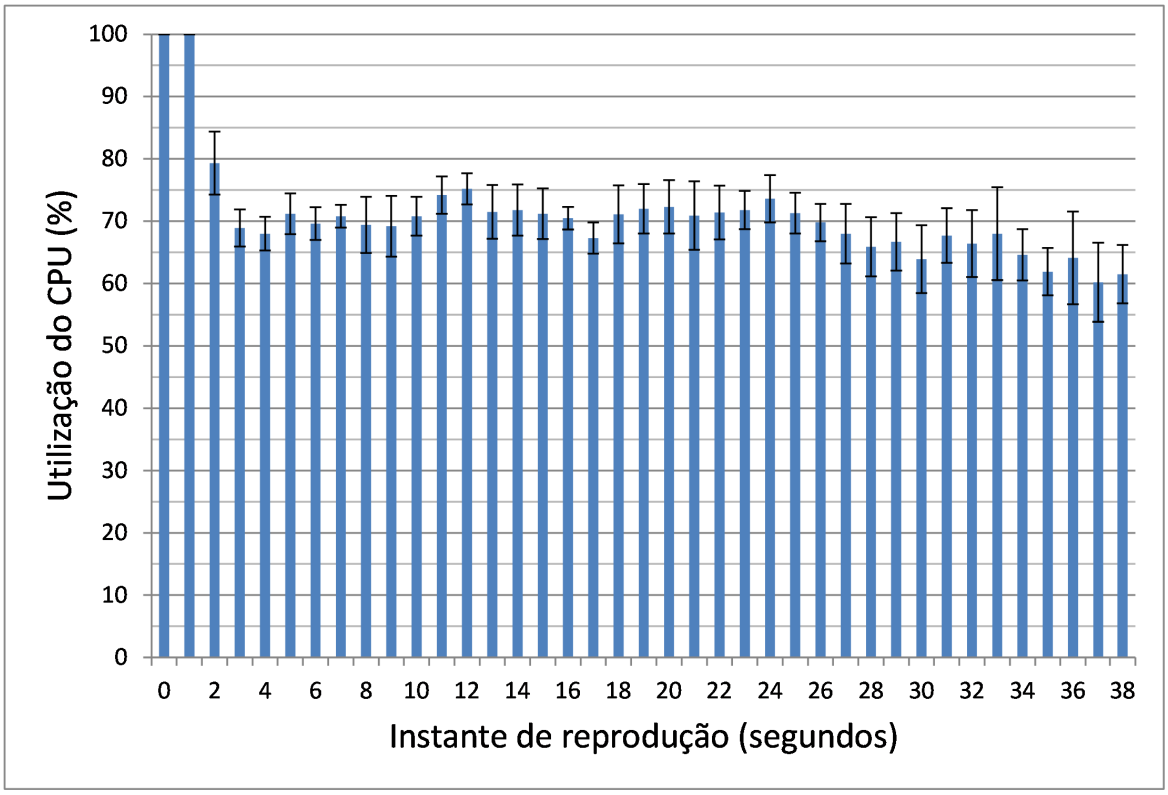

Figura 5.2: Média do processamento gasto durante a reprodução da *stream* multimédia referida em 5.2 utilizando a biblioteca RTSP.

pela descodificação/reprodução da *media* e pela biblioteca RTSP, a qual necessita de processar os pacotes RTP recebidos e guardá-los no seu *buffer* interno.

5.2.2 Avaliação da biblioteca SIP

Efectuado o teste à biblioteca RTSP, o próximo passo é testar a biblioteca SIP para que se possa comparar os resultados. Usando a mesma metodologia de teste e recorrendo ao *streamer* referido anteriormente em 5.1, o qual garante uma transmissão análoga à transmissão recebida no teste à biblioteca RTSP, foram efectuados também dez testes à biblioteca SIP. Os resultados obtidos nos dez testes efectuados são apresentados nos anexos B.1 e B.2, sendo que a tabela do anexo B.3 apresenta a média dos resultados obtidos para cada instante de reprodução. Uma vez que analogamente ao resultados obtidos na biblioteca RTSP, existe variação nos resultados, para que seja possível apresentar um resultado que se aproxime do valor real é necessário calcular a média.

Calculada a média dos resultados obtidos para cada instante de reprodução, foi calculado o desvio padrão da *media* em relação aos resultados dos testes efectuados e os respectivos intervalos de confiança. Estes resultados estão representados no gráfico da Figura 5.3 e nos anexos B.3 e B.4.

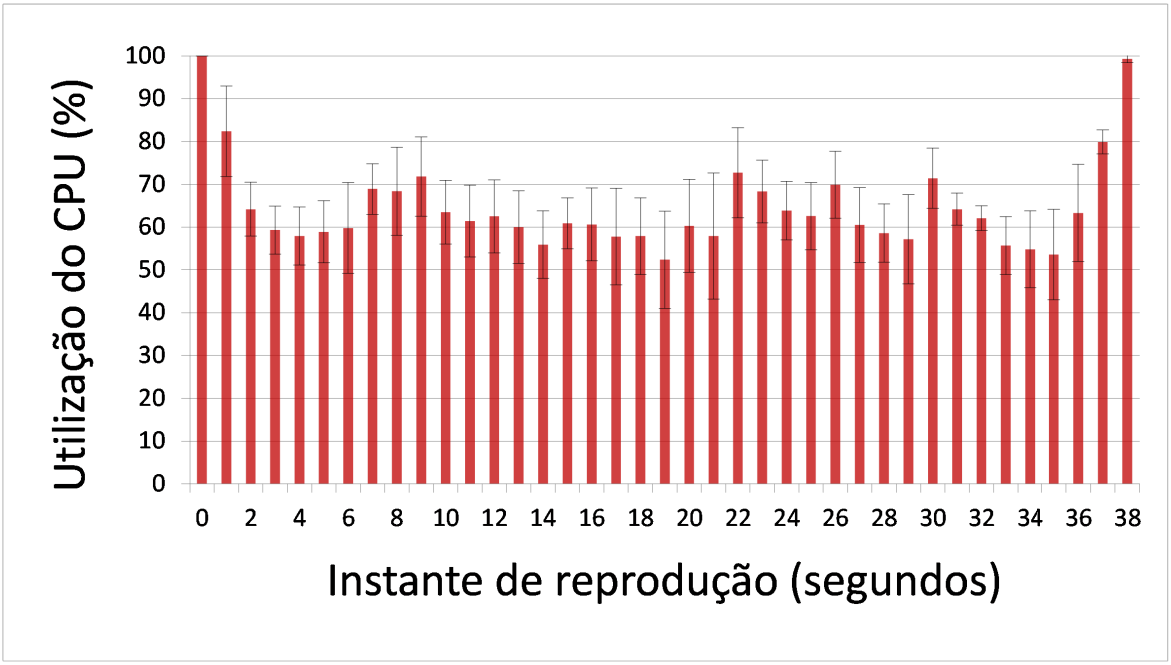

Figura 5.3: Média do processamento gasto durante a reprodução da stream multimédia referida em 5.2 utilizando a biblioteca SIP.

Como se pode observar pela Figura 5.3, os resultado obtidos na biblioteca SIP são muito idênticos aos resultados obtidos na biblioteca RTSP apresentados na figura 5.2. Inicialmente, existe analogamente à biblioteca RTSP uma utilização máxima do processador, factor esse que se deve ao já referido arranque de execução das componentes de descodificação/reprodução. No entanto, podemos observar que ao contrário da biblioteca RTSP, essa utilização máxima é reduzida mais rapidamente na biblioteca SIP do que na biblioteca RTSP. Esse factor deve-se ao facto da biblioteca SIP fazer o pré-processamento ao *payload* do fragmento de *media* à medida que os pacotes são recebidos. No caso da biblioteca RTSP como já tem pacotes em *buffer*, após a ordem de execução, faz logo o pré-processamento ao *payload* do fragmento a um conjunto de fragmentos. Este conjunto de fragmentos é igual ao limite máximo do *buffer* do descodificador (dez fragmentos).

Além da utilização máxima inicial, é possível observar-se uma utilização também máxima no final da reprodução da *media* na biblioteca SIP. Tendo apenas como base o conhecimento técnico

e estrutural do funcionamento da biblioteca SIP em relação ao resultado da biblioteca RTSP, não é possível explicar este facto. No entanto, com uma comparação mais pormenorizada, tendo em atenção os picos extremos e a constante variação, realizando alguns testes concretos é possível identificar o problema. Uma análise e explicação pormenorizada deste problema é efectuada em 5.2.3. De forma a facilitar a comparação das duas bibliotecas, a Figura 5.4 apresenta um gráfico com os resultados e os respectivos intervalos de confiança.

5.2.3 Comparação e avaliação do desempenho da biblioteca SIP

Como se pode observar pela Figura 5.4, é visível uma ligeira diminuição de processamento da biblioteca SIP em relação à biblioteca RTSP, representada respectivamente a cor vermelha e a cor azul. Esta ligeira diminuição de processamento, como já referida anteriormente em 3.4.3, já era esperada e deve-se ao facto da biblioteca SIP não implementar as componentes de armazenamento e controlo do *buffer* temporário implementado pela biblioteca RTSP. Em média a biblioteca RTSP utiliza 79% do processador do dispositivo ao longo da reprodução da *media*, com um intervalo de confiança médio de 3.9%. No caso da biblioteca SIP, é utilizado em média 65% do processador do dispositivo ao longo da reprodução da *media*, mas com um intervalo de confiança médio de 8%. Esta diferença significativa dos intervalos de confiança, está directamente relacionada com a maior variação da utilização do processador ao longo da reprodução da *media* na biblioteca SIP.

Analisando os resultados individuais de cada teste feito à biblioteca SIP (apresentados nas tabelas B.1 e B.2), pode verificar-se que em algumas situações existem alterações significativas na utilização do processador do dispositivo. Com base na visualização da reprodução da *media* durante a transmissão, é possível observar algumas falhas/pausas na reprodução de vídeo, sendo pouco regulares ou mesmo nenhumas as ocorrências do mesmo problema no áudio. Esta situação que acontece na utilização da biblioteca SIP e não acontece na biblioteca RTSP, está relacionada com o *buffer* do descodificador.

Para verificar esta situação, foi adicionado um *log* de utilização do *buffer* do descodificador durante o teste de carga. Através do log adicionado, é possível confirmar que sempre que a utilização do processador tem um pico de utilização muito baixo, o buffer de descodificação está cheio. Após um pico elevado de utilização, o *buffer* de vídeo continua a ser consumido normalmente. Entre estes dois picos observam-se falhas na visualização do vídeo, havendo um reposicionamento do vídeo e em algumas situações também do áudio após essas falhas.

O reposicionamento da *media* é relativo à falha de fragmentos descartados e é feita pelo reprodutor com base na informação temporal da *media* (*timestamp*). Esta situação justifica a constante utilização máxima do processador no final da reprodução da *media*. O fragmento a informar fim de envio de fragmentos de áudio (mensagem EOS) é enviado primeiro para o descodificador e reprodutor do que a mensagem de fim de fragmento de vídeo. Este comportamento leva a que a sincronização da componente de reprodução force o término da reprodução do vídeo, limpando todo o *buffer* e parando a actividade das componentes de descodificação e reprodução.

No caso da biblioteca RTSP este problema não acontece, pois a componente que implementa o *buffer* temporário faz a sincronização temporal dos fragmentos. Estes resultados obtidos, vêm alertar o problema de transmissão entre dispositivos com capacidades diferentes. No caso em que os dispositivos têm a mesma capacidade computacional, a codificação e descodificação é sincronizada pelo tempo de envio do transmissor que coincide com o tempo de captura. No entanto, no caso do transmissor ter uma maior capacidade computacional e estar a transmitir num formato que o receptor não tenha capacidade de acompanhar a descodificação em relação à recepção da *media*, existirá perda de pacotes e consequentemente má experiência para o utilizador. Este problema vem responder à interrogação do elevado consumo computacional por parte da biblioteca SIP no final da reprodução da *media*.

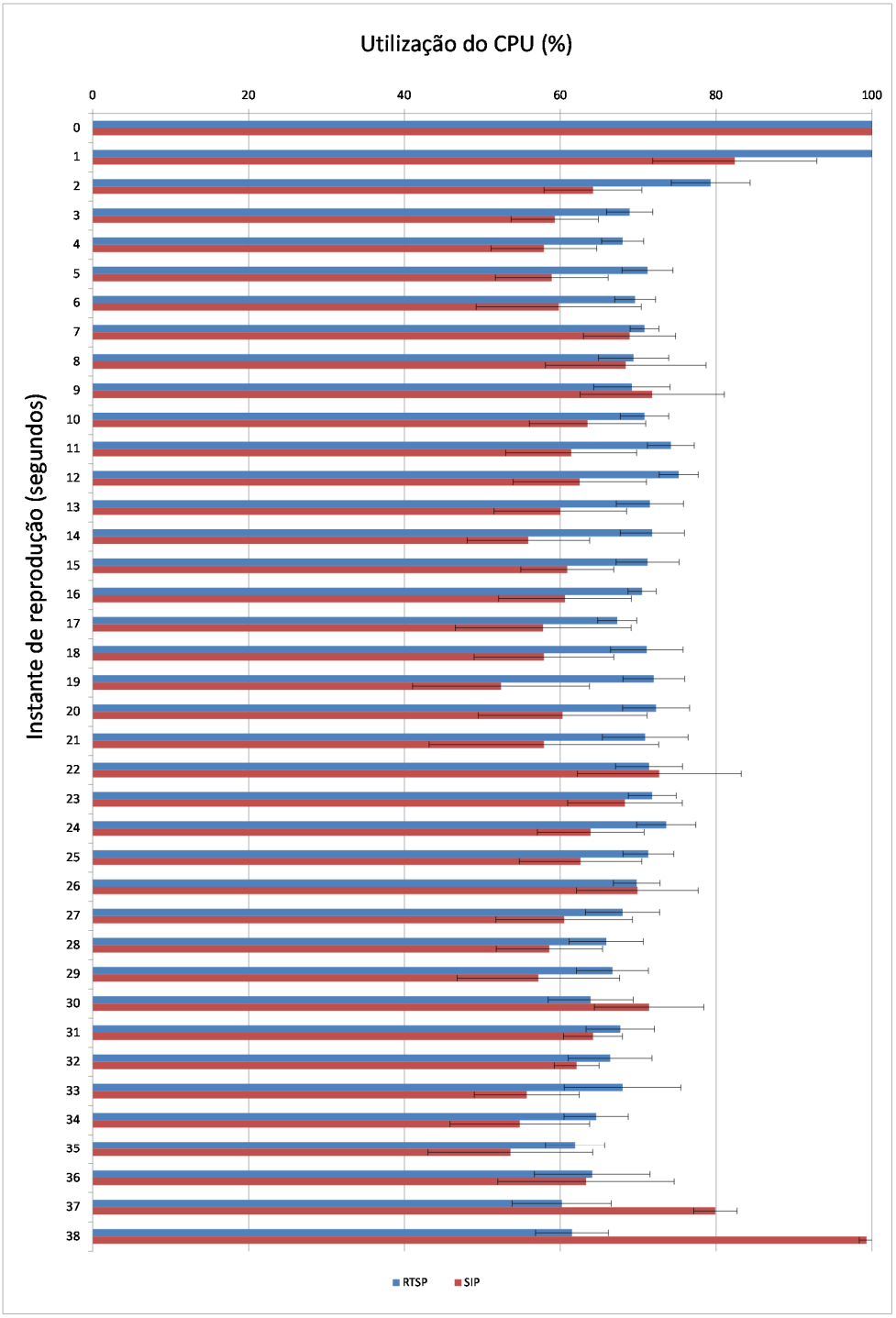

Figura 5.4: Resultados obtidos nos testes efectuados ás bibliotecas SIP e RTSP.

5.3 Problema de desempenho entre dispositivos

Como observado através dos resultados obtidos ao longo dos testes efectuados, existe um problema de desempenho entre diferentes dispositivos que pode influênciar a qualidade de experiência do utilizador durante a reprodução da *media*. Este problema pode ser contornado com a criação de um *buffer* como o da biblioteca RTSP. No entanto, a existência de um *buffer* implica um atraso na reprodução da *media*. Num cenário em que se pretende obter uma transmissão em tempo real, é necessário diminuir o atraso o mais possível, sendo que a criação do *buffer* não é solução propriamente aceitável.

Como solução para este problema, pode ser proposto um sistema de negociação do formato da *media* a ser transmitida sempre que o dispositivo verificar que é impossível efectuar a correcta descodificação e reprodução. Desta forma, o dispositivo garante que consegue fazer a descodificação da *media* durante a sua reprodução, diminuindo ou anulando as perdas de fragmentos devido à lotação do *buffer* do descodificador.

Actualmente o protocolo SIP, definido na RFC 2976 [45], disponibiliza a mensagem re-Invite, a qual pode ser utilizada para alterar as características da sessão SIP. Com esta mensagem, é possível que o dispositivo que recebe a *media* seja capaz de informar o dispositivo que a transmite da necessidade de alterar o formato da *media* a transmitir. A figura 5.5 apresenta um diagrama ilustrativo de um possível cenário com o envio da mensagem de re-invite para a reconfiguração da sessão SIP.

A Figura 5.5 pode levantar a questão de porquê não fazer a negociação do formato correcto logo na primeira mensagem de Invite. Como resposta a esta interrogação, esse facto deve-se a não se conhecer as características de processamento nem a disponibilidade de processamento do dispositivo a transmitir a *media* na altura de estabelecer a sessão. Conhecendo à partida as potencialidades do outro dispositivo, conseguia-se estabelecer inicialmente o formato correcto. No entanto, mesmo sabendo que o dispositivo tem determinadas características, não se sabe se as suas potencialidades vão estar totalmente disponíveis para a reprodução da *media*. Isto é, mesmo tendo um dispositivo muito potente, uma vez que o sistema operativo Android é multi-tarefa, nada garante que ao longo da reprodução da *media* todos os recursos do dispositivo vão estar disponíveis para efectuar essa operação.

A solução proposta vem resolver o problema da utilização da biblioteca SIP entre terminais com diferentes capacidades computacionais. Além do teste de qualidade por pacotes descartados, podem ser associadas outras métricas de qualidade de experiência do utilizador. Embora não

referidas neste trabalho, devido a se desviarem um bocado do tema e do objectivo pretendido, existem aspectos importantes que devem ser tidos em conta durante um transmissão multimédia em tempo real. Aspectos esses que podem ser considerados nesta componente, contribuindo para fornecer o serviço com a maior qualidade possível, não dependendo das características do dispositivo mas sim das suas potencialidade disponíveis no momento de execução.

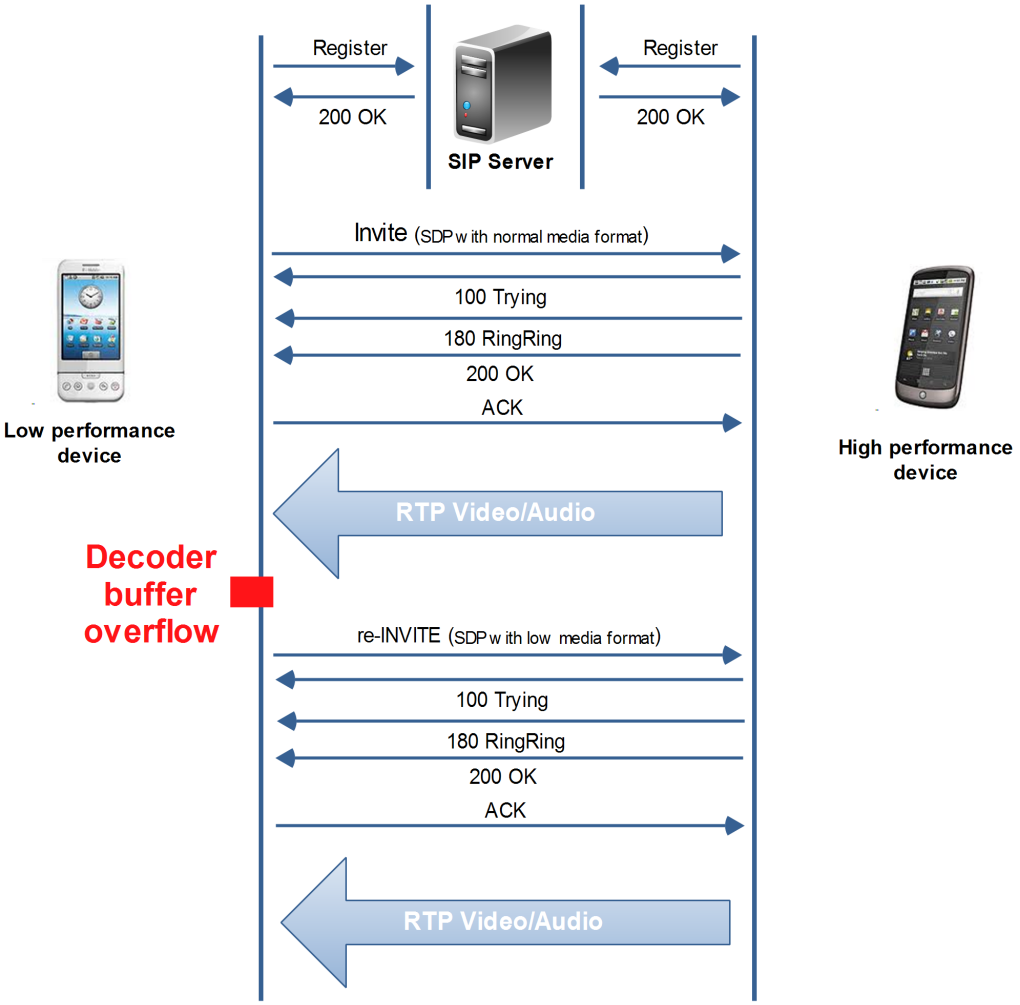

Figura 5.5: Negociação de formato de *media* entre dispositivos de diferentes capacidades de processamento.

5.4 Sumário

Ao longo deste capítulo foram abordados os testes efectuados à biblioteca SIP desenvolvida. Perante os resultados obtidos, pode-se observar que a biblioteca desenvolvida funciona correctamente, permitindo a reprodução de vídeo durante uma sessão SIP com era pretendido. Através dos resultados obtidos foi ainda possível detectar alguns problemas que podem ocorrer durante uma sessão SIP, entre dispositivos de diferentes capacidades de processamento.

Identificado e exposto o problema, foi apresentada uma possível solução. Esta solução além de resolver o problema em questão, pode no futuro ser um ponto de partida para melhorar a experiência do utilizador. Com base em futuras análises da qualidade do áudio e do vídeo reproduzido durante a sessão, é possível efectuar alterações à sessão, de modo a oferecer uma melhor qualidade de experiência ao utilizador.

Capítulo 6

Conclusões e Trabalho Futuro

No decorrer da dissertação foi apresentado e explicado o trabalho desenvolvido. Neste capítulo, é apresentada uma síntese de cada etapa do trabalho desenvolvido, as conclusões e as contribuições. Para finalizar, é proposto trabalho a ser desenvolvido no futuro, o qual dá continuação e tende a melhorar o trabalho iniciado nesta dissertação.

6.1 Resumo do trabalho desenvolvido

Esta dissertação teve como principal objectivo a integração do protocolo SIP na Media Framework da plataforma Android, o que possibilita a recepção e reprodução de vídeo durante uma sessão SIP. De forma a obter uma base de conhecimento para integrar o protocolo SIP na plataforma Android com suporte para vídeo, foi realizado um estudo aos vários tipos de *stream* existentes e os protocolos de sessão, transporte e descrição utilizados durante uma transmissão multimédia. Em continuação, foi analisado o sistema Android, dando uma descrição mais pormenorizada da componente multimédia. Por fim, foram apresentados alguns dos projectos SIP mais populares na comunidade Android, os quais evidenciam a necessidade da integração do protocolo SIP, com suporte para vídeo, na plataforma Android.

Introduzidas as tecnologias e exposta a necessidade do trabalho a desenvolver, foram definidos os requisitos aos quais a integração do SIP deverá responder. Com base nos requisitos e no estudo feito à Media Framework da plataforma Android, onde é possível concluir a impossibilidade da integração do protocolo SIP com suporte para vídeo na sua estrutura, uma vez que não existe suporte para manipular as componentes de vídeo directamente, é introduzida a *framework* Open-

Core. Recorrendo à leitura do seu código fonte da *framework* OpenCore e alguma documentação do funcionamento geral da sua API, é descrita a sua arquitectura e o seu funcionamento com a Media Framework do Android. É ainda descrito pormenorizadamente o funcionamento das suas componentes durante a execução da uma *stream* multimédia. Conhecida a arquitectura e o funcionamento interno da *framework* OpenCore, foi definida uma arquitectura para a implementação da biblioteca SIP, a qual será capaz de responder aos requisitos definidos. A arquitectura definida para a biblioteca SIP, segue a estrutura da biblioteca RTSP já disponível na *framework* OpenCore, que permite o seu registo e importação dinâmica perante a necessidade de execução de uma sessão SIP.

A implementação e integração da biblioteca SIP na Media Framework OpenCore da plataforma Android, envolveu compilação da SIP *stack* eXosip na estrutura do código fonte da plataforma Android, possibilitando a importação e utilização da SIP *stack* nas suas bibliotecas. Integrada a SIP *stack* na plataforma Android, foram desenvolvidas as componentes de biblioteca SIP, a quais são usadas para lidar com as restantes componentes da *framework* Opencore (componente de execução, descodificadores, reprodutores e a própria SIP *stack* integrada anteriormente).

Implementadas as componentes da biblioteca SIP, utilizando a estrutura de compilação do código fonte da plataforma Android (Makefile), ao qual foi adicionada à sua estrutura a nova biblioteca SIP e a SIP *stack*, é compilado o sistema e gerada a respectiva imagem. A imagem do sistema Android gerada, disponibiliza ao *Media Player* da Media Framework do Android a capacidade de estabelecer sessões SIP com suporte à recepção e reprodução de vídeo. Desta forma, definido o servidor SIP de registo, a actual API do *Media Player* do sistema Android é capaz de receber como parâmetro um endereço SIP e iniciar uma sessão SIP, reproduzindo a *media* recebida durante a sessão.

Com o objectivo de avaliar o funcionamento e o desempenho da biblioteca SIP desenvolvida, foram efectuados vários testes para obter resultados e comparações de desempenho, na descodificação e reprodução da *media*, de forma a validar a solução. Com base nos resultados obtidos, foi possível concluir que a biblioteca SIP desenvolvida responde aos requisitos definidos, possibilitando não só a reprodução de áudio durante uma sessão SIP, mas também de vídeo, como era pretendido. Ao longo dos testes efectuados, foram ainda encontrados alguns problemas que podem vir a ocorrer no estabelecimento de sessões SIP entre dispositivos de diferentes capacidades computacionais. Esta situações foram avaliadas e foi exposta uma possível solução para a resolução do problema em causa.

6.2 Principais Contribuições

O trabalho desenvolvido ao longo desta dissertação contribuiu com a integração do protocolo SIP directamente na Media Framework da plataforma Android, a qual dá a possibilidade da reprodução, não só de áudio como também de vídeo, durante uma sessão SIP. Esta integração, além de ser a única solução que até ao momento possibilita a reprodução de vídeo durante uma sessão SIP no sistema Android, veio dar uma visão e uma descrição mais detalhada da *framework* OpenCore. Esta contribuição ao nível de documentação, é bastante importante para o apoio ao desenvolvimento da *framework*, adicionando documentação importante, à escassa informação disponível sobre o funcionamento interno da *framework*. Por fim, em função dos resultados obtidos, foram apresentados alguns problemas importantes a ter em consideração durante uma sessão SIP entre diferentes dispositivos. Com base na identificação desses problemas, foi exposta uma solução, a qual poderá contribuir no melhoramento da qualidade de experiência do utilizador durante uma sessão SIP.

Como fruto da definição da arquitectura para a integração do protocolo SIP na *framework* OpenCore da plataforma Android, foi publicado o artigo "Integration of SIP protocol in Android Media Framework", apresentado na conferência internacional EUROCON 2011 - "IEEE International Conference on Computer as a Tool". Um novo artigo, com os resultados e a identificação dos possíveis problemas apresentados, está a ser preparado para futura publicação.

6.3 Trabalho Futuro

Como trabalho futuro e seguindo o problema e a solução expostos nos resultados obtidos, é necessário desenvolver uma componente de negociação do formato de *media*, dependendo da capacidade e disponibilidade de processamento do dispositivo. Esta componente tem como objectivo avaliar a qualidade de experiência do utilizador, no caso exposto, as métricas utilizadas apenas têm em conta garantir a reprodução correcta do vídeo (sem falhas nem pausas). No entanto, no futuro podem ser adicionadas novas métricas para melhorar a qualidade do vídeo dependendo das capacidades disponíveis no momento de estabelecimento da sessão (rede, capacidade de processamento do dispositivo, etc). Além da componente de controlo de qualidade de experiência proposta, é imporante estender à biblioteca SIP a capacidade de captura e transmissão de vídeo durante uma sessão SIP.

Por fim, também como trabalho futuro, é alertado a expansão da integração do trabalho de-

senvolvido nas restantes versões do Android. A expansão deste trabalho a novas versões do Android, tira partido da sua utilização em dispositivos mais potentes, o que possibilitará a utilização de formatos de vídeo com melhor qualidade e melhor compactação, fornecendo uma melhor experiência ao utilizador e um menor custo de transmissão.

Apêndice A

Resultados dos testes efectuados à biblioteca RTSP

Instante de reprodução (segundos)	***CPU (%)***	***CPU (%)***	***CPU (%)***	***CPU (%)***	***CPU (%)***
0	100	100	100	100	100
1	100	100	100	100	100
2	80	85	91	83	75
3	63	70	73	61	67
4	65	68	60	65	72
5	78	72	68	71	65
6	71	67	66	71	61
7	68	72	71	68	69
8	64	69	79	73	76
9	67	63	73	62	86
10	76	70	72	60	75
11	76	79	81	67	73
12	68	75	77	73	78
13	64	76	68	66	66
14	68	64	76	71	61
15	70	73	83	71	69
16	73	73	70	67	73
17	69	65	61	67	75
18	71	59	73	75	75
19	71	79	70	81	69
20	64	79	79	76	74
21	56	63	78	68	80
22	68	63	75	68	82
23	73	74	65	65	76
24	65	82	70	68	76
25	60	74	76	72	69
26	68	67	72	70	67
27	72	60	69	68	74
28	63	55	58	68	65
29	62	67	70	62	69
30	52	57	55	68	61
31	54	67	67	74	58
32	60	73	72	76	64
33	50	65	66	69	68
34	62	59	69	64	72
35	71	63	66	60	65
36	82	62	70	58	57
37	63	65	60	63	48
38	54	63	56	70	50

Tabela A.1: Testes ao processamento utilizado pela biblioteca RTSP durante a reprodução da stream 5.2

Instante de reprodução (segundos)	*CPU (%)*	*CPU (%)*	*CPU (%)*	*CPU (%)*	*CPU (%)*
0	100	100	100	100	100
1	100	100	100	100	100
2	81	81	71	62	84
3	69	69	78	68	71
4	72	73	69	72	64
5	75	80	64	71	68
6	69	75	69	75	72
7	72	71	73	67	77
8	65	66	78	61	73
9	64	65	79	64	69
10	76	66	72	68	73
11	71	69	77	70	79
12	76	75	71	76	83
13	64	74	75	85	77
14	84	71	75	77	71
15	79	64	74	61	68
16	73	73	70	66	67
17	65	69	65	75	72
18	71	63	64	85	75
19	77	59	67	74	73
20	81	60	71	67	72
21	82	68	79	73	62
22	79	73	76	70	60
23	77	77	67	76	67
24	72	81	67	80	75
25	67	73	69	75	78
26	79	63	65	75	72
27	73	51	64	75	74
28	75	66	61	67	81
29	78	75	59	54	71
30	74	80	65	59	68
31	71	77	69	67	73
32	65	67	68	73	46
33	59	71	72	63	97
34	53	63	67	61	76
35	52	54	59	69	60
36	51	61	53	87	60
37	39	67	55	70	72
38	63	75	58	60	66

Tabela A.2: Testes ao processamento utilizado pela biblioteca RTSP durante a reprodução da stream 5.2

Instante de reprodução (segundos)	*Média de Utilização do CPU (%)*	*Desvio padrão (%)*
0	100	0
1	100	0
2	79.3	8.15
3	68.9	4.79
4	68	4.37
5	71.2	5.26
6	69.6	4.25
7	70.8	2.97
8	69.4	7.26
9	69.2	7.89
10	70.8	5.03
11	74.2	4.85
12	75.2	4.05
13	71.5	6.96
14	71.8	6.65
15	71.2	6.53
16	70.5	2.92
17	67.3	4.06
18	71.1	7.49
19	72	6.39
20	72.3	6.93
21	70.9	8.86
22	71.4	6.93
23	71.8	4.96
24	73.6	6.13
25	71.3	5.25
26	69.8	4.83
27	68	7.69
28	65.9	7.68
29	66.7	7.42
30	63.9	8.80
31	67.7	7.07
32	66.4	8.62
33	68	12.06
34	64.6	6.65
35	61.9	6.11
36	64.1	11.99
37	60.2	10.25
38	61.5	7.55

Tabela A.3: Media e desvio padrão dos testes realizados à biblioteca RTSP durante a reprodução da media referida em 5.2

Instante de reprodução (segundos)	*Intervalo de confiança (%)*
0	0
1	0
2	5.05
3	2.97
4	2.70
5	3.26
6	2.63
7	1.84
8	4.50
9	4.89
10	3.12
11	3.00
12	2.50
13	4.32
14	4.12
15	4.05
16	1.81
17	2.51
18	4.64
19	3.96
20	4.29
21	5.49
22	4.30
23	3.08
24	3.80
25	3.25
26	2.99
27	4.77
28	4.76
29	4.60
30	5.45
31	4.38
32	5.38
33	7.48
34	4.12
35	3.79
36	7.43
37	6.35
38	4.68

Tabela A.4: Intervalo de confiança calculado com base no desvio padrão em A.3 e com uma significância de 95% dos resultados obtidos.

Apêndice B

Resultados dos testes efectuados à biblioteca SIP

Instante de reprodução (segundos)	***CPU (%)***	***CPU (%)***	***CPU (%)***	***CPU (%)***	***CPU (%)***
0	100	100	100	100	100
1	87	91	83	95	37
2	70	69	73	67	37
3	67	60	65	58	37
4	71	57	50	50	34
5	64	56	56	59	30
6	54	66	67	63	35
7	67	70	72	70	79
8	79	78	67	61	94
9	72	88	62	54	86
10	67	76	68	59	36
11	74	67	17	67	26
12	65	60	71	72	27
13	60	54	65	64	25
14	55	56	60	61	34
15	66	63	64	68	46
16	72	71	72	64	33
17	69	80	66	28	35
18	63	75	63	38	35
19	56	67	64	28	27
20	66	68	55	30	82
21	70	65	63	27	97
22	79	100	72	87	72
23	72	92	71	65	66
24	68	65	83	57	68
25	60	58	88	65	69
26	55	59	72	76	72
27	61	67	68	69	65
28	65	50	63	61	62
29	72	57	59	59	59
30	67	65	66	66	69
31	59	71	67	57	57
32	53	57	70	64	71
33	44	46	45	40	65
34	37	54	35	35	68
35	39	68	23	47	70
36	89	73	40	68	62
37	82	87	82	79	85
38	100	100	100	100	96

Tabela B.1: Testes ao processamento utilizado pela biblioteca SIP durante a reprodução da stream 5.2

Instante de reprodução (segundos)	***CPU (%)***	***CPU (%)***	***CPU (%)***	***CPU (%)***	***CPU (%)***
0	100	100	100	100	100
1	83	81	93	79	95
2	67	62	65	63	69
3	56	68	62	55	65
4	64	60	68	59	66
5	69	56	73	65	61
6	74	25	78	68	68
7	67	44	77	72	71
8	63	30	72	75	65
9	59	100	66	71	60
10	66	80	60	66	57
11	56	63	68	59	63
12	69	71	69	53	68
13	72	67	64	56	73
14	66	32	67	64	64
15	63	42	71	68	58
16	54	43	75	61	61
17	57	39	78	57	69
18	61	42	69	60	73
19	38	37	64	64	79
20	30	62	66	69	75
21	32	21	64	72	68
22	87	39	67	63	61
23	78	50	73	60	56
24	68	40	63	58	69
25	61	36	61	62	66
26	59	100	69	66	71
27	54	53	73	70	25
28	62	50	77	60	36
29	65	23	81	62	35
30	69	89	69	59	95
31	73	70	65	65	58
32	65	63	61	65	62
33	61	65	57	70	64
34	57	59	62	75	66
35	62	61	67	67	32
36	80	67	65	63	26
37	78	75	78	72	81
38	100	100	97	100	100

Tabela B.2: Testes ao processamento utilizado pela biblioteca SIP durante a reprodução da stream 5.2

Instante de reprodução (segundos)	***Média de Utilização do CPU (%)***	***Desvio padrão (%)***
0	100	0
1	82.4	17
2	64.2	10.11
3	59.3	9.04
4	57.9	10.95
5	58.9	11.69
6	59.8	17.10
7	68.9	9.55
8	68.4	16.63
9	71.8	14.94
10	63.5	12.04
11	61.4	13.54
12	62.5	13.79
13	60	13.73
14	55.9	12.70
15	60.9	9.63
16	60.6	13.75
17	57.8	18.20
18	57.9	14.46
19	52.4	18.33
20	60.3	17.49
21	57.9	23.75
22	72.7	16.97
23	68.3	11.88
24	63.9	11.04
25	62.6	12.67
26	69.9	12.60
27	60.5	14.13
28	58.6	11
29	57.2	16.80
30	71.4	11.34
31	64.2	6.11
32	62.1	4.65
33	55.7	10.90
34	54.8	14.47
35	53.6	17.06
36	63.3	18.28
37	79.9	4.48
38	99.3	1.50

Tabela B.3: Media e desvio padrão dos testes realizados à biblioteca SIP durante a reprodução da media referida em 5.3

Instante de reprodução (segundos)	***Intervalo de confiança (%)***
0	0
1	10.53
2	6.27
3	5.61
4	6.79
5	7.24
6	10.59
7	5.92
8	10.31
9	9.26
10	7.46
11	8.39
12	8.55
13	8.51
14	7.87
15	5.97
16	8.52
17	11.28
18	8.96
19	11.36
20	10.84
21	14.72
22	10.52
23	7.36
24	6.84
25	7.85
26	7.81
27	8.76
28	6.82
29	10.42
30	7.03
31	3.78
32	2.88
33	6.75
34	8.97
35	10.58
36	11.33
37	2.79
38	0.93

Tabela B.4: Intervalo de confiança calculado com base no desvio padrão em B.3 e com uma significância de 95% dos resultados obtidos.

Bibliografia

[1]]Zambelli Alex. *"The birth of smooth streaming*, Fevereiro 2009. http://alexzambelli.com/blog/2009/02/04/the-birth-of-smooth-streaming/.

[2] A. Johnston and R. Sparks. Session Description Protocol (SDP) Offer/Answer Examples. RFC 4317 (Informational), December 2005.

[3] Jason Chen. An introduction to android. Google I/O Sessions, 2008.

[4] David Sparks. Mastering the android media framework. Google I/O Sessions, 2009.

[5] pv. Opencore multimedia framework capabilities. OHA 2.07, rev 2, December 15, 2009.

[6] Object Factory. *Adapter Pattern*. http://www.dofactory.com/Patterns/PatternAdapter.aspx.

[7] HTC. *HTC G1*. http://www.htc.com/sea/product/dream/specification.html.

[8] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), April 1998.

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630.

[10] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327 (Proposed Standard), April 1998. Obsoleted by RFC 4566, updated by RFC 3266.

[11] Hechmi and G. Jean-Charles. Ims for enterprises. In *IEEE Communications Magazine*, 2004.

[12] *The eXtented eXosip stack*, 2008. `http://www.antisip.com/doc/exosip2/group__libeXosip2.html`.

[13] K. Sripanidkulchai, B. Maggs, and H. Zhang. An analysis of live streaming workloads on the internet. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 41–54. ACM, 2004.

[14] A. Zambelli. Iis smooth streaming technical overview. *Microsoft Corporation*, 2009.

[15] Z. Yetgin and G. Seckin. Progressive download for 3g wireless multicasting. In *Future Generation Communication and Networking (FGCN 2007)*, volume 1, pages 289–295. IEEE, 2007.

[16] R.S. Ramanujan, J.A. Newhouse, M.N. Kaddoura, A. Ahamad, E.R. Chartier, and K.J. Thurber. Adaptive streaming of mpeg video over ip networks. In *Local Computer Networks, 1997. Proceedings., 22nd Annual Conference on*, pages 398–409. IEEE, 1997.

[17] S.F. Chang and R. Kumar. Real-time content-based adaptive streaming of sports videos. In *Content-Based Access of Image and Video Libraries, 2001.(CBAIVL 2001). IEEE Workshop on*, pages 139–146. IEEE, 2001.

[18] C. Krasic, J. Walpole, and W. Feng. Quality-adaptive media streaming by priority drop. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 112–121. ACM, 2003.

[19] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rfc 3550: Rtp: A transport protocol for real-time applications. *IETF, July*, 2003.

[20] C. Liu. 2 multimedia over ip: Rsvp, rtp, rtcp, rtsp. *Handbook of emerging communications technologies: the next decade*, 6:29, 2000.

[21] Q. FANG, M. WANG, and Y. JI. Design and implementation of media on demand server based on rtsp/rtp [j]. *Computer Engineering and Design*, 1, 2006.

[22] M. Handley, C. Perkins, and V. Jacobson. Sdp: session description protocol. 2006.

[23] J. Postel et al. Rfc 768: User datagram protocol. *InterNet Network Working Group, Agustus*, 1980.

[24] C. Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley Professional, 2003.

[25] K. Singh and H. Schulzrinne. Interworking between sip/sdp and h. 323. In *Proceedings of the 1st IP-Telephony Workshop (IPtel 2000)*. Citeseer.

[26] K. Singh, G. Nair, and H. Schulzrinne. Centralized conferencing using sip. In *Internet Telephony Workshop*. Citeseer, 2001.

[27] T.T. Kwon, M. Gerla, and S. Das. Mobility management for voip service: Mobile ip vs. sip. *Wireless Communications, IEEE*, 9(5):66–75, 2002.

[28] E. Wedlund and H. Schulzrinne. Mobility support using sip. In *Proceedings of the 2nd ACM international workshop on Wireless mobile multimedia*, pages 76–82. ACM, 1999.

[29] Open Handset Alliance. *Android*. `http://www.openhandsetalliance.com/`.

[30] Android. *Android SDK*. `http://developer.android.com/sdk/index.html`.

[31] Ray Cromwell. Building applications with google apis. Google I/O Sessions, 2009.

[32] Dan Bornstein. Dalvik virtual machine internals. Google I/O Sessions, 2008.

[33] Android. *Android NDK*. `http://developer.android.com/sdk/ndk/overview.html`.

[34] Dan Galpin. Close to the metal: Building with the android ndk. Google I/O Sessions, 2010.

[35] Patrick Brady. Android anatomy and physiology. Google I/O Sessions, 2008.

[36] JAVIER TAPIA, JIM KOSMACH, DUSAN VESELINOVIC, GREG SHERWOOD, and RALPH NEFF. Introduction to the opencore audio components used in the android platform. In *AES 34th International Conference, Jeju Island, Korea*, 2008 August 28–30.

[37] C-Cast Project. *C-Cast*. `http://www.ict-ccast.eu/`.

[38] Sipdroid. *Sipdroid*. `http://sipdroid.org/`.

[39] mjsip. *MJSIP*. `http://www.mjsip.org/`.

[40] sipdroid. *sipdroid*. `http://code.google.com/p/sipdroid/wiki/ChangeLog`.

[41] S. Srinivasan. Design patterns in object-oriented frameworks. *Computer*, 32(2):24–32, 1999.

[42] E. Gamma, R. Helm, R. Johnson, J. Vlissides, et al. *Design patterns*, volume 1. Addison-Wesley Reading, MA, 2002.

[43] Wireshark. *Wireshark*. `http://www.wireshark.org/`.

[44] trixbox. *SIP Server*. `http://fonality.com/trixbox/`.

[45] S. Donovan. The SIP INFO Method. RFC 2976 (Proposed Standard), October 2000. Obsoleted by RFC 6086.