UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

AVALIAÇÃO DE QOS EM SISTEMA DE COMUNICAÇÃO VOIP UTILIZANDO PLATAFORMA EMBARCADA

ANDRÉ BISINOTO MATIAS

MONOGRAFIA APRESENTADA COMO REQUISITO NECESSÁRIO PARA CONCLUSÃO DO CURSO DE ENGENHARIA ELÉTRICA

APROVADA POR:

_____

ORIENTADOR: PROF. DR. FRANCISCO ASSIS DE OLIVEIRA NASCIMENTO

_____

CO-ORIENTADOR: OTÁVIO HENRIQUE GALEAZZI FRANCO

_____

EXAMINADOR INTERNO: EDSON MINTSU HUNG

MONOGRAFIA DE TRABALHO DE CONCLUSÃO DO CURSO DE ENGENHARIA ELÉTRICA

BRASÍLIA/DF: 12 - 2008

FICHA CATALOGRÁFICA

MATIAS, ANDRÉ BISINOTO

Avaliação de QoS em Sistema de Comunicação VoIP Utilizando Plataforma Embarcada [Distrito Federal] 2008.

xiii, 44p., 210 x 297 mm (ENE/FT/UnB, Engenheiro Eletricista, Engenharia Elétrica, 2008).

Monografia de Trabalho de Conclusão de Curso – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1. Desenvolvimento Linux embarcado
2. Voz sobre IP
3. Medidas de qualidade
4. Triple Play
I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MATIAS, A. B. (2008). Avaliação de QoS em Sistema de Comunicação VoIP Utilizando Plataforma Embarcada. Monografia de Trabalho de Conclusão do Curso de Engenharia de Elétrica, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 44p.

CESSÃO DE DIREITOS

AUTOR: André Bisinoto Matias.

TÍTULO: Avaliação de QoS em Sistema de Comunicação VoIP Utilizando Plataforma Embarcada.

GRAU: Engenheiro Eletricista ANO: 2008

_____

André Bisinoto Matias
SQN 116 Bloco I apto 406, Asa Norte
70773090. Brasília – DF – Brasil.
E-mail: andrebisinoto@yahoo.com.br

DEDICATÓRIA

Aos meus pais, José Bonifácio e Carmem Lúcia.

AGRADECIMENTOS

A Deus, Senhor Misericordioso, que leva minha vida em suas mãos.

À minha família, em especial meus pais, que são meus mestres e exemplos.

À Dani, minha amada, pela companhia, apoio e ajuda em todos os momentos.

Aos amigos que me acompanharam até aqui.

Aos Eng. Jorge Pereira e Otávio Franco e demais colaboradores da Wise Indústria de Telecomunicações, pelo aprendizado e pela experiência.

Ao orientador Prof. Dr. Francisco Assis Nascimento e ao Eng. Wagner Popov, pela disponibilidade e por toda ajuda dispensada.

RESUMO

AVALIAÇÃO DE QOS EM SISTEMA DE COMUNICAÇÃO VOIP UTILIZANDO PLATAFORMA EMBARCADA

Autor: André Bisinoto Matias

Orientador: Francisco Assis Nascimento

Brasília, dezembro de 2008

No anseio de tecnologias cada vez mais eficientes e lucrativas, a tecnologia *Voice over IP* (VoIP) surgiu como alternativa aos circuitos comutados da telefonia tradicional. Seu funcionamento é baseado em comutação de pacotes de voz em uma Rede IP e representa redução de custos e integração com outros serviços da Internet. Entretanto, a qualidade de uma chamada VoIP em relação a uma chamada da telefonia tradicional é consideravelmente inferior, pois a comunicação é sensível à perda de dados e aos atrasos da rede. Vários parâmetros objetivos de qualidade e modelos computacionais complexos foram criados com o objetivo de caracterizar uma chamada para que usuários e prestadoras de serviço tenham garantias de satisfação. Uma estratégia para certificar a qualidade adotada por prestadoras de serviço de telecomunicações é realizar diversos testes em diferentes pontos da rede (armário de distribuição, modem do assinante, etc.). Para isso, elas utilizam equipamentos de teste de mão (*hand-held test sets*). Este trabalho é um estudo acerca das medidas de qualidade mais comuns e a descrição da implementação de um teste de qualidade na plataforma embarcada do TSW800TP, produzido pela Wise Indústria de Telecomunicações.

ABSTRACT

QOS EVALUATION ON AN EMBEDDED PLATFORM VOIP COMMUNICATION SYSTEM

Author: André Bisinoto Matias

Supervisor: Francisco Assis Nascimento

Brasília, December of 2008

Yearning for more efficient and lucrative technologies, the *Voice over IP* (VoIP) technology has appeared as an alternative to the switched circuits of traditional telephony. Its functioning is based on switching voice packages on an IP Network and represents a cost reduction and an integration with other Internet services. However, the quality of a VoIP call is considerably inferior to the one of the traditional telephony call, because the communication is sensible to data loss and to network delays. Several objective parameters of quality and complex computational models were created with the objective of characterizing a call with satisfaction guarantee for the users and telecommunication operators. A strategy to certificate the quality adopted by the telecommunication operators is to make several tests in different network points (closets of distribution, subscriber modem, etc). For that, they use hand-held test sets. This work is a study about ordinary quality measures and the description of the implementation of a quality test on the embedded platform of TSW800TP, produced by Wise Indústria de Telecomunicações.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

ADPCM	*Adaptive Differential Pulse Code Modulation*
ADSL	*Assimetric Digital Subscriber Line*
ARM	*Advanced RISC Machine*
ATM	*Asynchronous Transfer Mode*
BSP	*Board Support Package*
CPU	*Central Processing Unit*
cRTP	*Compressed Real-Time Protocol*
DHCP	*Dynamic Host Configuration Protocol*
DNS	*Domain Name System*
DSLAM	*Digital Subscriber Line Access Multiplexer*
ETSI	*European Telecommunications Standards Institute*
FTTP	*Fiber To The Premises*
GNU	*General Public License*
GSM	*Global System for Mobile communications*
HTTP	*Hypertext Transfer Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
iLBC	*internet Low Bitrate Codec*
IP	*Internet Protocol*
IPTV	*TV over Internet Protocol*
IrDA	*Infrared Data Association*
ITU	*International Telecommunications Union*
ITU-T	*Telecommunication Standardization Sector*
LCD	*Liquid Crystal Display*
MB	*Megabyte*
MGCP	*Media Gateway Control Protocol*

MOS	*Mean Opinion Score*
MOS-CQ	*Mean Opinion Score Conversational Quality*
MOS-CQE	*Mean Opinion Score Conversational Quality Estimated*
MOS-LQ	*Mean Opinion Score Listening Quality*
NAT	*Network Address Translation*
NTP	*Network Time Protocol*
PCAP	*Packet Capture*
PCM	*Pulse-Code Modulation*
PLC	*Packet Loss Concealment*
QoE	*Quality of Experience*
QoS	*Quality of Service*
R	*Rating factor*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
RFC	*Request for Comments*
RR	*Receiver Report*
RS-232	*Recommended Standard 232*
RTCP	*Real-Time Control Protocol*
RTPC	Rede Telefônica Pública Comutada
RTP	*Real-Time Protocol*
RTCP XR	*Real-Time Control Protocol Extended Reports*
RTT	*Round-Trip Time*
SD	*Secure Digital* (*card*)
SIP	*Session Initiation Protocol*
SMTP	*Simple Mail Transfer Protocol*

SR	*Sender Report*
SRTP	*Secure Real-Time Protocol*
TCP	*Transmission Control Protocol*
TFTP	*Trivial File Transport Protocol*
TLS	*Transport Layer Security*
UA	*User Agent*
UAC	*User Agent Client*
UAS	*User Agent Server*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VAD	*Voice activity detection*
VoIP	*Voice over Internet Protocol*
WAV	*Waveform audio format*
WiMAX	*Worldwide Interoperability for Microwave Access*

1 – INTRODUÇÃO

1.1 – MOTIVAÇÃO

Desde a invenção do telefone em 1860, redes de comunicação de voz têm se desenvolvido e se diversificado. Sem dúvida, conversar com outra pessoa a quilômetros de distância se tornou um dos confortos do mundo moderno com o qual a maior parte das pessoas se acostumou a viver e ao qual ninguém renuncia.

Com retorno financeiro garantido, grandes empresas de telecomunicações se firmaram apoiadas nas redes de telefonia. O advento da Internet, porém, trouxe novos desafios para o ramo. Sob um novo paradigma, a tecnologia VoIP – *Voice over Internet Protocol* – possibilita chamadas com custo consideravelmente reduzido, integração de diferentes serviços da Internet e atrai a atenção tanto de usuários comuns como de grandes corporações.

Diante dos freqüentes problemas encontrados para a entrega dos pacotes de voz em tempo real, diversos modelos foram criados e diferentes parâmetros levantados para caracterizar uma chamada VoIP com relação à qualidade. Em princípio, desde a possibilidade da realização de uma chamada até a contagem de pacotes de voz perdidos são parâmetros para qualificar uma chamada. Entretanto, modelos mais complexos representam melhor a situação real do problema.

Perdas de pacotes e pacotes fora de seqüência, *jitter* e *delay* são parâmetros de fácil implementação e que trazem informações diferentes a respeito da chamada. Porém, sozinhos não são capazes de qualificar uma chamada plenamente. O principal parâmetro para qualificar uma chamada telefônica é o *Mean Opinion Score*.(MOS) [2]. Primeiramente, o MOS era um parâmetro subjetivo e, assim, difícil e custoso de se implementar. Posteriormente, surgiram modelos computacionais que possibilitaram a estimação do MOS para uma determinada chamada a partir dos parâmetros supracitados e de outros fatores como eco e qualidade dos codificadores de voz. O Modelo E, criado pelo ETSI e padronizado pela ITU-T, é um dos principais modelos existentes [4].

A fim de reduzir ainda mais os custos e assim aumentar os lucros, empresas prestadoras de serviços de telecomunicações vêm buscando integrar diferentes serviços em uma mesma plataforma. Este fenômeno é conhecido como Convergência em Telecomunicações e iniciou-se com a digitalização da rede de telefonia comutada (RTPC) na década de 1980. Redes convergentes têm as vantagens de possibilitar facilidade e simplicidade ao usuário final. Estudos realizados nos Estados Unidos e na Europa indicam que o índice de abandono voluntário de serviço é consideravelmente menor para usuários adeptos de soluções convergentes [19]. Ultimamente, as prestadoras vêm incorporando serviços de dados (Internet banda larga), voz (VoIP) e vídeo (IPTV). *Triple Play* foi o nome comercial dado a este conjunto de serviços.

Como toda tecnologia, serviços *Triple Play* necessitam de validação por meio de testes de campo. Para chegar a uma solução rápida e eficiente, quando houver reclamações de usuários, é importante dispor de vários parâmetros diferentes para a análise do problema. A utilização de *test sets* foi adotada já há algum tempo pelas empresas tradicionais de telecomunicações. Assim, encontramos no mercado vários exemplos de *test sets* que fazem medidas de qualidade da voz em uma chamada VoIP tais como o HST-3000 da JDSU® [18] e o CoLT-450P da EXFO® [6].

1.2 – OBJETIVOS

O objetivo principal deste trabalho é implementar um protótipo de um teste de qualidade de uma chamada VoIP em uma plataforma embarcada. Esta plataforma embarcada em questão é o *test set* TSW800TP, produzido pela Wise Indústria de Telecomunicações. O processador do TSW800TP é o i.MX21®, que é baseado na arquitetura ARM9.

Para programar o teste de qualidade, é necessário fazer um estudo preliminar sobre características de sistemas VoIP e de diferentes protocolos envolvidos na comunicação, assim como levantar os principais parâmetros de qualidade de uma chamada VoIP. Deseja-se também conhecer os padrões internacionais definidos pela *International Telecommunications Union* – ITU – para sistemas VoIP.

Como o desempenho do teste envolve muitos fatores, tais como características dos codificadores de áudio, dos protocolos de comunicação e do processador, optou-se por dar mais enfoque nas características da comunicação. O desempenho de codificadores e do processador será apenas ilustrado.

1.3 – APRESENTAÇÃO DO TRABALHO

O trabalho está disposto em seis capítulos. No capítulo 2 faz-se uma revisão geral de sistemas VoIP e se apresentam os principais protocolos utilizados no decorrer deste trabalho. No capítulo 3 revisam-se as medidas de qualidade de uma chamada VoIP, enfatizando-se as implementadas nos testes de qualidade. Por sua vez, nos capítulos 4 e 5 são descritos os passos e ferramentas da implementação do software e o resultado final obtido, respectivamente. As conclusões gerais do trabalho e perspectivas para a sua continuidade são apresentadas no capítulo 6.

2 – VOIP

A transmissão de voz em pacotes de IP iniciou-se praticamente junto com a Internet em 1973. Desde cedo, a idéia de se usar a Internet como um meio “gratuito” para se transmitir voz despertou o interesse dos pesquisadores. Ora, uma vez conectado à Rede, o usuário não mais precisa manter um canal único de comunicação como ocorre na Rede Telefônica Pública Comutada (RTPC). Assim, chamadas entre dois terminais VoIP não têm custo de conexão algum para o usuário final.

Isto revela também a maior vantagem deste sistema em relação à RTPC. Na RPTC o canal é reservado 100% do tempo para uma única chamada mesmo que em determinados momentos não haja nada para ser transmitido. No VoIP um mesmo canal pode ser compartilhado por várias chamadas, pois não há transmissão durante todo o tempo. Uma vez que certa rota usada entre os dois terminais estiver sobrecarregada, podem-se desviar os pacotes por rotas mais aliviadas. Mesmo uma chamada de um terminal VoIP para um número da RTPC tem o seu custo consideravelmente reduzido, especialmente quando em chamadas internacionais: do terminal VoIP até à rede RTPC local ao terminal da chamada não há custos, praticamente. É como se a chamada internacional tivesse o mesmo preço de uma chamada local.

Outras características marcantes do VoIP são listadas abaixo:

1. Capacidade de transmitir mais de uma chamada no mesmo meio físico e na mesma banda de freqüência;
2. Conferências, encaminhamento de chamadas, rediscagem automática, identificador de chamadas e outras implementações de software sem custo adicional das operadoras;
3. Integração com outros serviços de Internet – transmissão de vídeo, texto, arquivos.
4. Impossibilidade de identificar o endereço físico da chamada. Em situações de emergência, como numa ligação para o Corpo de Bombeiros, não há como saber o endereço do incidente por meio da chamada.

2.1 – DESCRIÇÃO DO FUNCIONAMENTO

Basicamente, a comunicação se dá pela digitalização da onda sonora produzida pela voz, compressão do sinal digitalizado, encapsulamento dos dados em pacotes e transmissão pela rede. O outro lado recebe os pacotes e realiza o processo inverso a fim de obter novamente a onda sonora. A Figura 2.1 mostra este esquema.

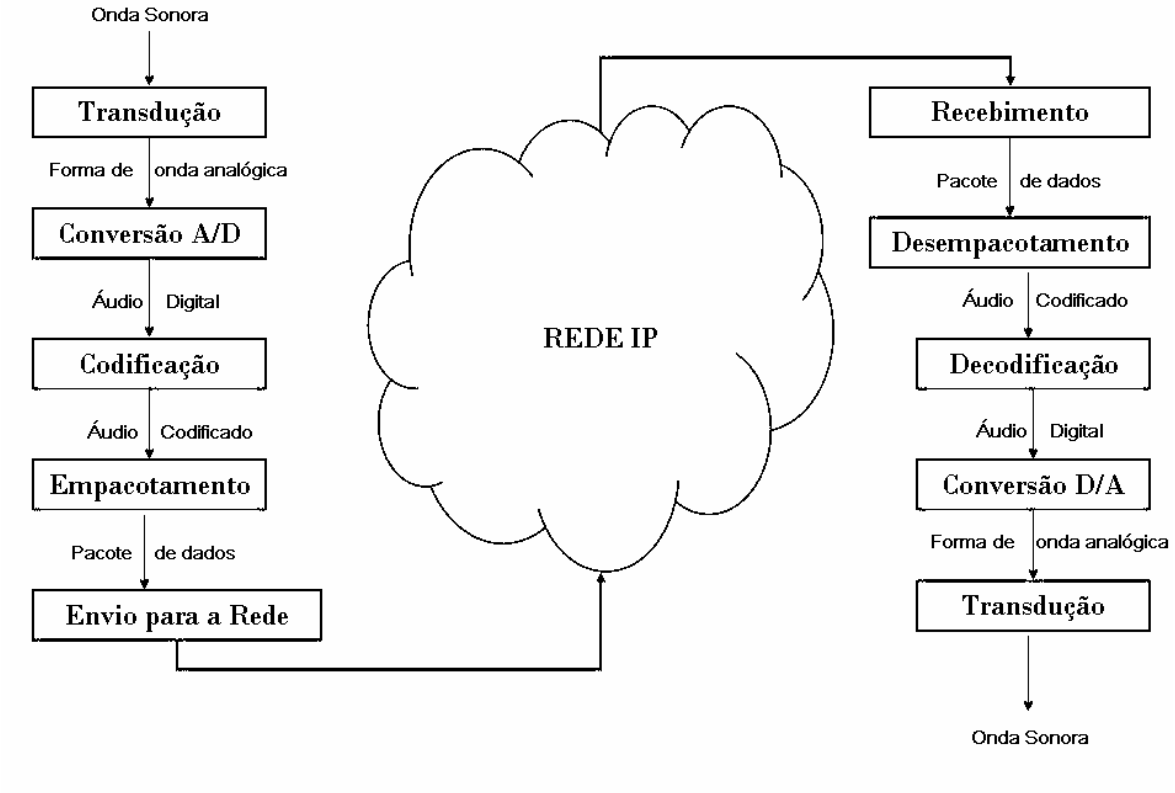

Figura 2.1 – Processo de comunicação de voz em uma Rede IP. [23]

Há diversos padrões para a compressão do sinal digital da voz. Cada um deles possui técnicas diferentes para diminuir o número de bits para representar o sinal. Conseqüentemente, diferentes taxas de transmissão, qualidade sonora subjetiva e processamento dispensado na compressão são obtidos quando se utilizam os diferentes CODECs (COdificador/DECodificador). Mais informações sobre características e desempenho de codecs podem ser encontradas em [23] e em [10].

De acordo com o modelo TCP/IP de camadas de comunicação, tais pacotes são descritos pelo *Real-Time Protocol* (RTP) na camada de aplicação e pelo *User Datagram Protocol* (UDP) na camada de transporte, além do *Internet Protocol* (IP) na camada de Internet, obviamente. Esses protocolos são, em geral, fixos. Os protocolos das camadas de enlace e

física podem ser os mais diversos. Exemplos de protocolos da camada de enlace incluem o *Asynchronous Transfer Mode* (ATM) e o *Ethernet* e da camada física incluem a *Assimetric Digital Subscriber Line* (DSL), o *Fiber To The Premises* (FTTP) e o *Worldwide Interoperability for Microwave Access* (WiMAX). Serão apresentados apenas os protocolos da camada de aplicação e de transporte. A Figura 2.1 ilustra essa associação de protocolos para um pacote de voz.

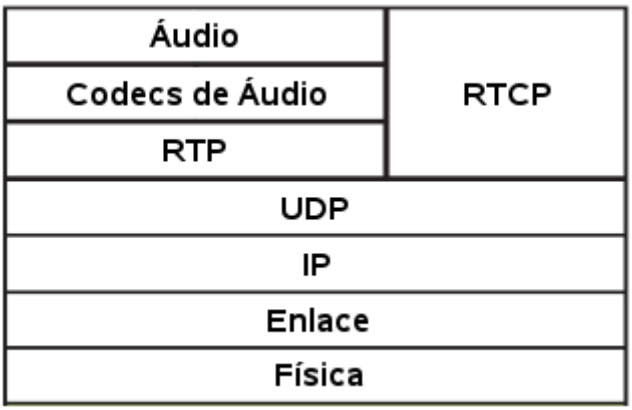

Figura 2.2 – Protocolos de comunicação de um pacote de voz. Modificado, [1]

Para a sinalização das chamadas, há três principais protocolos de uso comum em sistemas VoIP: o H.323, desenvolvido pela ITU; o *Session Iniciate Protocol* (SIP), desenvolvido pela IETF; e o *Media Gateway Control Protocol* (MGCP), desenvolvido pela IETF. Será descrito com mais detalhes somente o padrão SIP, que está se tornando o padrão mais comum e que foi escolhido como o protocolo de sinalização do software desenvolvido neste projeto.

2.1.1 – RTP

O RTP é descrito atualmente pela RFC 3550 e foi desenvolvido para dar suporte a aplicações de tempo real como transmissão de voz e vídeo entre dois *terminais* da rede. Esse suporte inclui identificação do tipo de dados (*payload*), numeração dos pacotes (*sequence numbering*), marcação do tempo (*timestamping*) e monitoração de entrega. Geralmente, ele é associado ao protocolo UDP e IP e pode tanto fornecer sessões *unicast* quanto *multicast*.

A RFC do RTP envolve dois tipos de pacotes de dados: o pacote RTP, que carrega os dados; e o pacote RTCP (*Real-Time Control Protocol*), que provê informações a respeito dos participantes da sessão e da qualidade do serviço.

Há também protocolos derivados do RTP, também usados em serviços VoIP:

- SRTP (*Secure Real-Time Protocol*), descrito pela RFC 3711, que provê criptografia aos dados transmitidos.
- RTCP XR (*Real-Time Control Protocol Extended Reports*), descrito pela RFC 3611, que fornece mais detalhes de qualidade. Esse protocolo possui funcionalidades específicas para chamadas VoIP.
- cRTP (*Compressed Real-Time Protocol*), descrito pela RFC 2508, que corresponde ao RTP com o cabeçalho comprimido, a fim de otimizar a largura de banda.

O cabeçalho RTP possui o esquema mostrado na Figura 2.3.

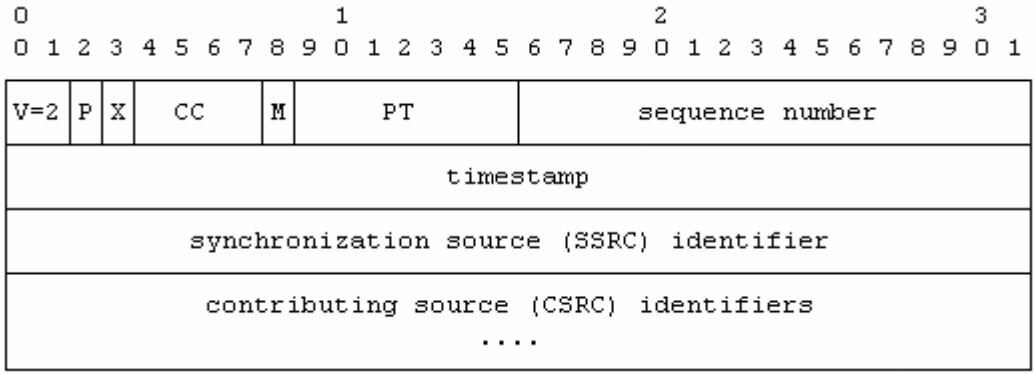

Figura 2.3 – Cabeçalho RTP. [21]

A RFC 3550 explica detalhadamente cada um desses campos. Serão detalhados, todavia, somente os campos que são importantes para o cálculo de parâmetros de qualidade.

- *payload type* (PT): 7 bits

Indica o formato dos dados do pacote RTP. Se o receptor não souber qual é o formato expresso nesse campo, deve ignorar o pacote.

- *sequence number*: 16 bits

Este campo é incrementado de um a cada pacote RTP enviado. Assim, o receptor pode verificar se os pacotes vieram na ordem correta e se algum pacote foi perdido. O valor inicial deste campo deve ser aleatório por motivos de segurança.

- *timestamp*: 32 bits

O *timestamp* representa o instante de amostragem do primeiro octeto de dados do pacote RTP. Esse instante deve ser derivado de um relógio que incrementa monotonicamente e linearmente no tempo. Ele é usado para a sincronização e cálculo de *jitter*, que será discutido no capítulo 3. A freqüência do relógio depende do formato dos dados e é especificado estaticamente ou dinamicamente por outros meios que não o RTP. Da mesma forma que o *sequence number*, o primeiro *timestamp* também deve ser aleatório.

- *Synchronization source* (SSRC): 32 bits

Esse número identifica uma fonte de dados em nível de sessão RTP. Exemplos de *synchronization source* podem ser um emissor de um *stream* de pacotes advindos de um microfone ou um *mixer* RTP.

O protocolo associado ao RTP – nesse caso, o UDP – deve prover uma diferenciação dos pacotes de dados e de controle. A RFC determina que, para o UDP, os pacotes RTP deveriam ser colocados em uma porta de número par e os pacotes RTCP na próxima porta, que será ímpar.

Esses pacotes têm quatro funções. A primeira é fornecer informações a respeito da qualidade da distribuição de dados. A segunda é ter um identificador chamado *canonical name* (CNAME), que se mantém por toda a sessão (um para cada participante). A terceira é controlar a taxa de pacotes RTCP enviados por cada terminal. A quarta função é fornecer informações mínimas do controle da sessão, como a identificação de participantes a ser mostrada na interface do usuário. Essa última função é opcional.

Os pacotes RTCP podem ser de cinco tipos diferentes:

- *Sender report* – SR. Carrega estatísticas de transmissão e recepção de participantes que são emissores ativos de dados.

- *Receiver report* – RR. Transmite estatísticas de recepção de participantes que não são emissores ativos.

- *Source description* – SDES. Carrega dados de descrição da fonte, como o CNAME.

- BYE. Indica a saída de um *terminal* da sessão RTP.

- APP. Carrega funções específicas da aplicação. Criado para uso experimental em novos desenvolvimentos.

Serão descritos somente os pacotes tipo SR, de onde se extrairá o parâmetro de qualidade *Round Trip Time*, explanado na seção 3.3.2. O formato dos pacotes SR está esquematizado na Figura 2.4.

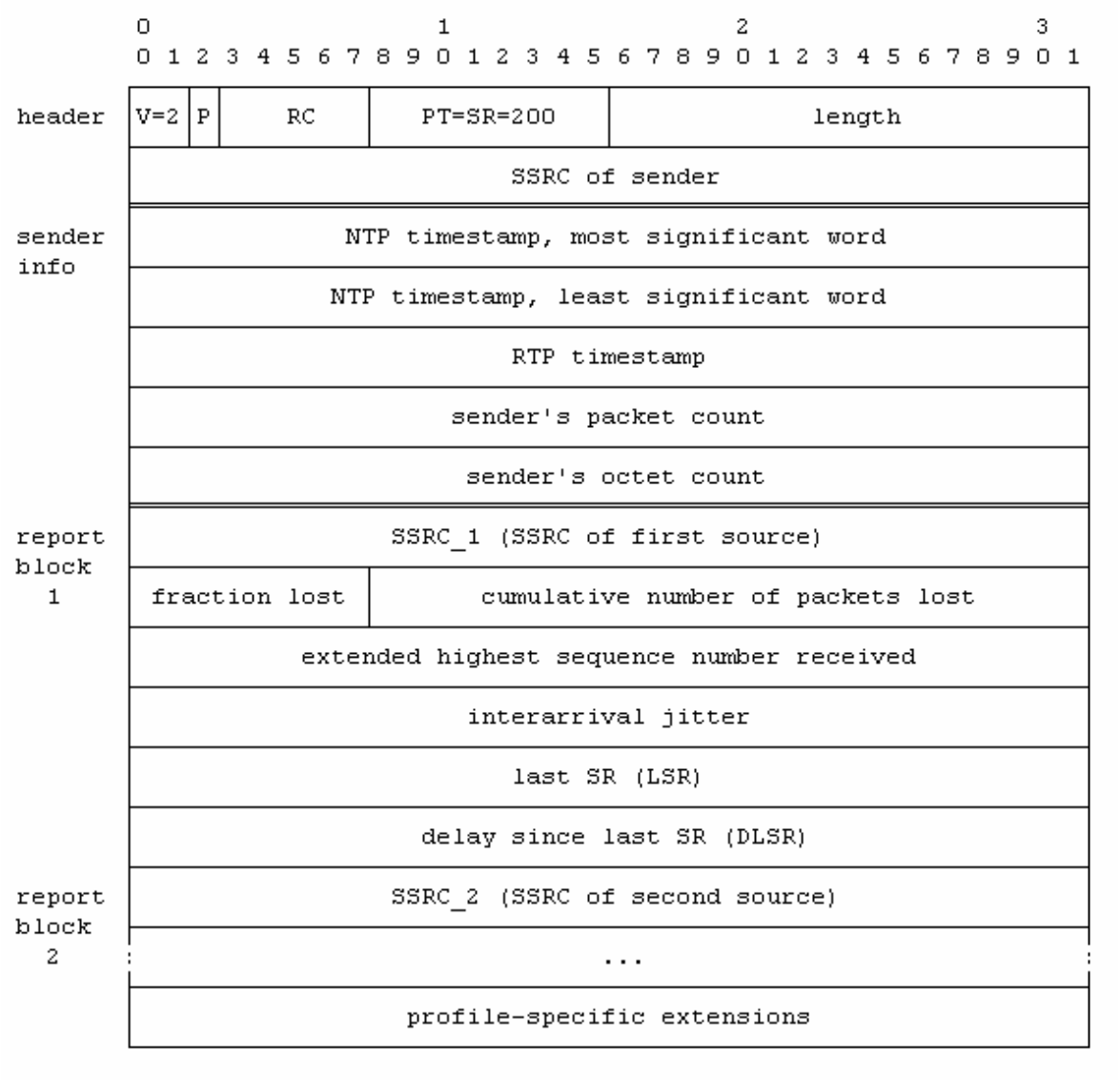

Figura 2.4 – Conteúdo do pacote RTCP tipo SR. [21]

Destacar-se-ão alguns campos que fornecerão informações importantes a respeito da qualidade do serviço:

- *packet type* (PT): 8 bits

Identificador do pacote SR RTCP, sempre igual a 200.

- SSRC *of sender*: 32 bits

SSRC do emissor deste pacote SR.

- NTP *timestamp*: 64 bits

Esse campo indica o tempo "*wallclock*" segundo o *Network Time Protocol* (NTP) no momento em que o pacote SR é enviado, ou seja, determina o tempo de envio do pacote.

- *sender's packet count*: 32 bits

Número de pacotes de dados RTP enviados pelo emissor deste pacote SR.

- *sender's octet count*: 32 bits

Número total de octetos de dados (voz, vídeo, etc.) transmitidos pelo emissor deste pacote SR desde o início da comunicação. Este valor é usado para estimar taxa média de transmissão dos dados.

- *interarrival jitter*: 32 bits

Jitter dos pacotes recebidos pelo emissor deste pacote SR reportado aos outros *terminais*. Cálculo de *jitter* será minuciosamente explicado na seção 3.3.

- *last* SR *timestamp* (LSR): 32 bits

32 bits retirados do *timestamp* NTP do pacote SR recebido da fonte SSRC_n. Se nenhum pacote deste tipo foi recebido ainda, o campo é zero.

- *delay since last* SR (DLSR): 32 bits

Atraso entre o recebimento do ultimo pacote SR da fonte SSRC_n e o envio deste pacote. Da mesma forma que no LSR, se nenhum SR foi recebido, o DLSR é zero.

2.1.2 – UDP

Descrito pela RFC 768, o UDP é um protocolo da camada de transporte que, ao contrário do TCP (*Transport Control Protocol*), não oferece garantias da entrega do pacote. Porém, esta característica o torna mais simples e mais rápido do que o TCP. Além disso, a retransmissão do pacote perdido que ocorre no TCP não é um esquema interessante para o VoIP, pois o pacote chega com um atraso grande e fora de seqüência. Assim, tal pacote pode vir a tornar-se um ruído inserido no sistema. Por isso é que o UDP foi escolhido para a transmissão de dados em tempo real, como voz e vídeo.

2.1.3 – SIP

O SIP é um protocolo de sinalização que se presta a estabelecer, modificar e interromper sessões multimídia na Internet. Ele é usado, por exemplo, no VoIP, em vídeo conferências, na transmissão de mensagens instantâneas e jogos em rede. Ele é um protocolo da camada de aplicação que pode ser associado ao UDP ou ao TCP e é baseado em texto da mesma forma que o HTTP e o SMTP. O documento que o descreve é a RFC 3261.

Para entender simplificadamente o funcionamento do SIP, alguns dos seus componentes serão enumerados:

- SIP *User Agent* (UA)

Um UA é um dispositivo no terminal de uma rede SIP. É do UA que se originam os pedidos para estabelecer a sessão de dados.
Um *User Agent Client* (UAC) é a parte do UA que realiza os pedidos e um *User Agent Server* (UAS) é a parte do UA que responde os pedidos de outro UAC. Todo UA é composto de um UAC e um UAS.

- Servidor *Proxy*

Um servidor *proxy* SIP recebe os pedidos vindos de um UA ou de outro servidor e encaminha o pedido para outra localização (outro UA, por exemplo).

- Registrador (*Registrar server*)

Servidor que recebe os pedidos de registro de um UA toma as medidas necessárias (verifica a senha de acesso, permite ao UA fazer chamadas, etc.).

O SIP é um protocolo muito versátil e fornece diversos serviços em diferentes configurações. Uma chamada VoIP usando a sinalização SIP será descrita a seguir. Essa chamada tem dois UA que se registram em um servidor e realizam a chamada por meio de uma sessão RTP. O servidor em questão faz o papel do registrador e do *proxy*.

No primeiro momento, os UA mandam um pedido de registro ao servidor e cada um recebe uma mensagem (“200 OK”), confirmando o registro. Depois, o UA 1 inicia o processo da chamada, mandando um “INVITE” para o servidor, que, por sua vez, encaminha esse

pedido ao UA 2. Após o UA 2 aceitar a chamada, uma sessão RTP é iniciada entre os dois UA. Por fim, um “BYE” é enviado de um usuário diretamente ao interlocutor para encerrar a chamada. A Figura 2.5 ilustra o processo.

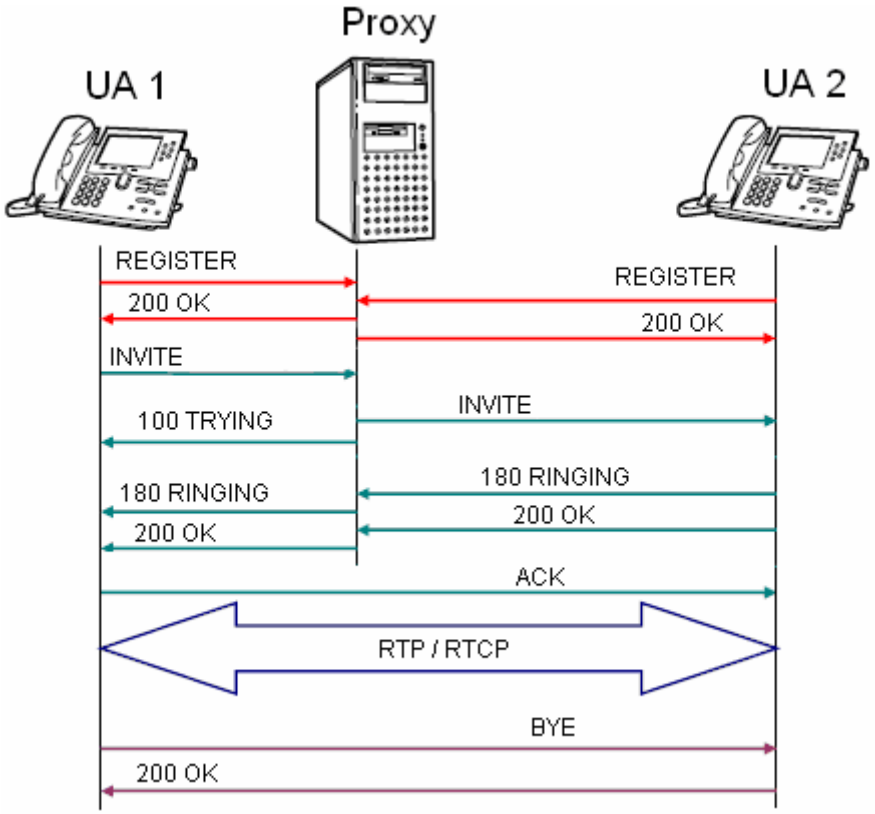

Figura 2.5 – Processo de registro e sinalização de chamada VoIP por meio de *proxy* SIP

Outra configuração possível é a realização de chamadas sem o uso do servidor. Desde que os dois usuários estejam na mesma sub-rede, as mensagens podem ser trocadas diretamente entre os dois usuários.

3 – QUALIDADE DE SERVIÇO DE UM SISTEMA VOIP

Da mesma forma que em qualquer tecnologia, a prestadora de serviços precisa se certificar da qualidade do sistema no momento e no local de sua instalação. Ao resolver problemas (*troubleshoot*) inesperados, ela também precisa de parâmetros que indiquem uma solução eficiente.

Em geral, a insatisfação do cliente advém de uma experiência subjetiva do serviço. A respeito disso, há o termo QoE (Qualidade de Experiência). Em oposição, o termo QoS (Qualidade de Serviço) engloba parâmetros objetivos diretamente relacionados à experiência do usuário.

Os parâmetros apresentados a seguir não são, obviamente, todos os parâmetros desenvolvidos para qualificar uma chamada VoIP. Entretanto, são os principais encontrados em inúmeras referências.

3.1 – PERDA DE PACOTES

Pacotes podem se perder, basicamente, por dois motivos: por descarte nos roteadores da rede ou no *buffer* de recepção, ou por erro de transmissão de bit [4]. Eles são facilmente contados, verificando-se o *sequence number* dos pacotes RTP. Da mesma forma, pode-se verificar se os pacotes chegaram fora de seqüência.

Perdas de pacotes de dados trazem um claro prejuízo à conversação, pois são dados que não chegam ao receptor. Entretanto, o dano subjetivo é às vezes diferente para cada codec, já que alguns possuem estratégias diversas para reparar os efeitos dos pacotes perdidos [4]. Por isso, a quantidade relativa de pacotes perdidos aceitável varia de acordo com o codec, porém sem ultrapassar 1% [5]. Vários codecs possuem ainda um PLC (*Packet Loss Concealment*) para encobrir os efeitos da perda de pacotes.

3.2 – *DELAY*

Em uma chamada telefônica, certamente um alto atraso no recebimento da voz torna a conversação cansativa e ininteligível. "*Delay*" ou atraso é justamente a medida de tempo usada para qualificar a comunicação sob este aspecto.

3.2.1 – ATRASO FIM A FIM

O atraso fim a fim é tratado na recomendação G.114 da ITU-T. Ele é analisado com base no Modelo E. Essa recomendação traz a relação entre o seu valor e a qualidade subjetiva da chamada e os seus limites aceitáveis. Transcrevemos a Figura 3.1 da recomendação G.114 para ilustrar a relação entre o atraso e o valor de *R* sem considerar outros fatores. O valor de referência para um atraso ótimo é 150ms e para um atraso ruim é 400ms. Entretanto, em transmissões de longas distâncias, especialmente envolvendo *links* de comunicação via satélite, toleram-se valores maiores do que 400ms [14].

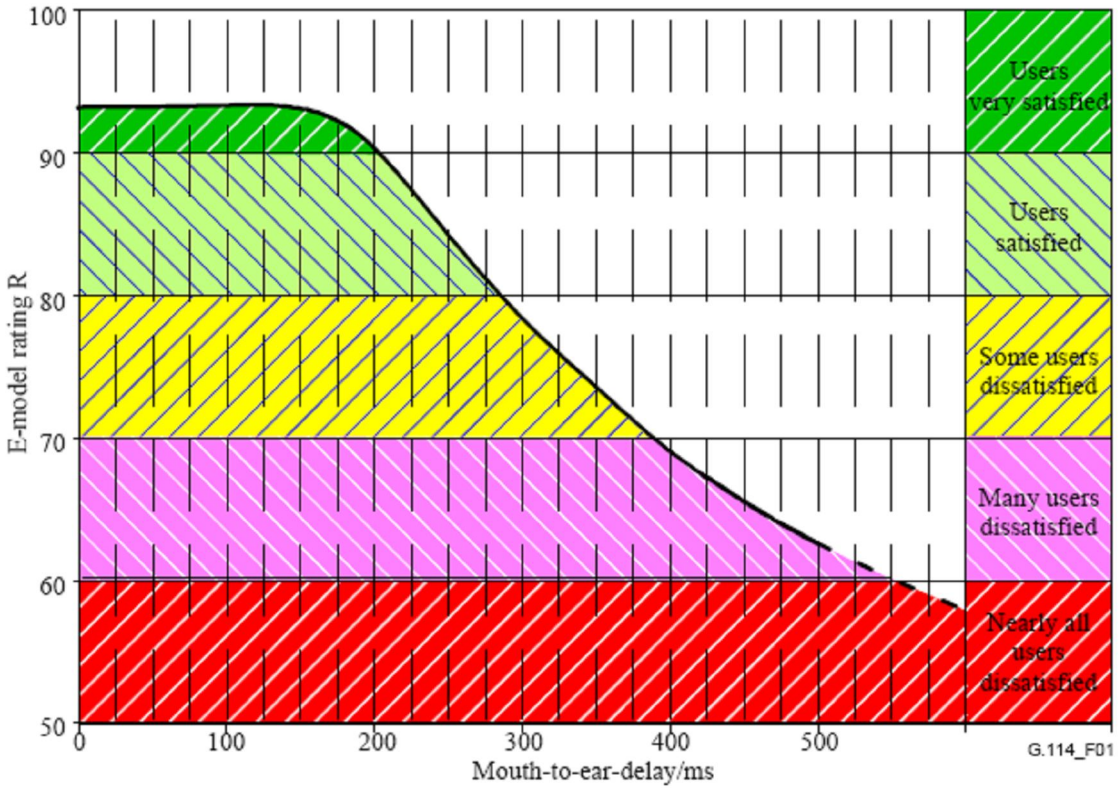

Figura 3.1 – Determinação dos efeitos do atraso pelo Modelo E. [14]

O atraso total na comunicação na verdade é a soma dos vários atrasos ocorridos ao longo do sistema [14]: devido à codificação, nos circuitos de transmissão, devido ao empacotamento dos dados em ambientes IP, devido ao *buffer* de variação do recebimento

de pacotes (*jitter*). A recomendação descreve cada um destes atrasos e traz várias tabelas com valores comuns em vários ambientes para a estimativa do atraso total.

3.2.2 – *ROUND TRIP TIME* (RTT)

Round trip time numa rede IP é o tempo decorrido desde o envio de um pacote por um usuário e o retorno de outro pacote enviado imediatamente em resposta pelo interlocutor até o usuário que enviou o primeiro. Como a rede IP em geral não é simétrica e as rotas percorridas pelos pacotes são estocásticas, o RTT não é o dobro do atraso na rede [4].

O RTCP possibilita medida de RTT, por meio dos pacotes tipo SR. No cálculo do RTT, seguem-se os passos:

1. Armazenar o valor do tempo do recebimento do pacote SR;
2. Subtrair o valor do tempo do recebimento do pacote SR (LSR) recebido pelo outro terminal;
3. Subtrair a diferença de tempo entre o pacote SR recebido e o pacote enviado (DLSR).

A Figura 3.2 retirada da RFC 3550 exemplifica o procedimento.

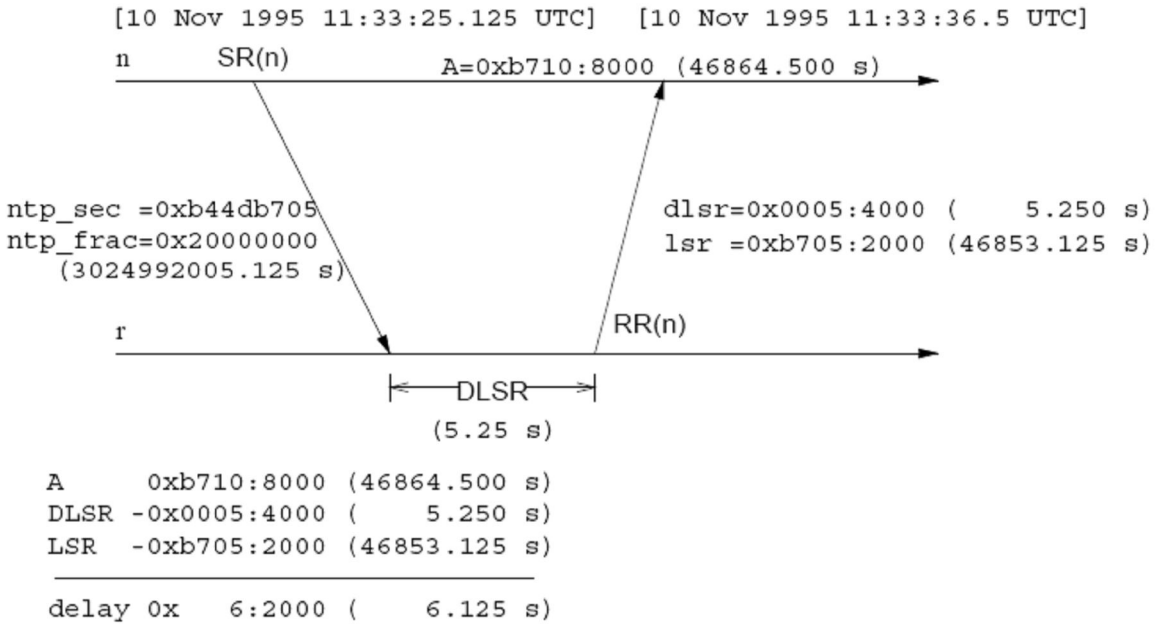

Figura 3.2 – Exemplo de cálculo de RTT. [21]

Onde se observa o receptor enviar o pacote RR(n), pode-se interpretar como um pacote do tipo SR, pois tanto pacotes RR quanto pacotes SR têm os campos DLSR e LSR.

3.4 – *JITTER*

A palavra *jitter* designa a variação do atraso de recebimento dos pacotes sucessivos de dados. Quando essa variação é alta, a voz é recebida com interrupções e a conversação se torna difícil. Este efeito é causado pelo congestionamento das redes e causa a rejeição de pacotes na recepção. Por meio dele, podemos ter noção do congestionamento da rede. Para amenizar os efeitos do *jitter*, coloca-se um *buffer* no recebimento dos pacotes.

Seja t_i o tempo de recebimento do i-ésimo pacote e r_i o *timestamp* retirado do cabeçalho RTP deste mesmo pacote. Seja também Δt_i a diferença do tempo de recebimento entre dois pacotes sucessivos e Δr_i a diferença do tempo de envio dos mesmos pacotes. O *jitter* instantâneo do i-ésimo pacote é, por definição,

$$J_i = \Delta t_i - \Delta r_i \tag{3.1}$$

A RFC 3550, no entanto, recomenda que o *jitter* instantâneo seja passado por um filtro passa-baixas, para que o valor possua uma correta interpretação [17]. Ela considera que o *jitter* deve ser filtrado para que sejam retirados ruídos da medição [21]. Assim, o *jitter* segundo a RFC é

$$\begin{aligned} j_1 &= J_1 \\ j_i &= \frac{15}{16} j_{i-1} + \frac{1}{16}\left|J_i\right| \end{aligned} \tag{3.2}$$

A Cisco® avalia em seus livros como *jitter* ótimo o que é menor ou igual a 30 ms [5].

3.5 – MOS

O *Mean Opinion Score* (MOS) é o mais popular parâmetro subjetivo para qualificação da voz. Ele é especificado pela recomendação P.800 da ITU-T e foi criado inicialmente para qualificar codificadores de voz. Consiste numa escala de 1 a 5 em que 1 é uma qualidade muito ruim e 5 é excelente. A Tabela 3.1 ilustra melhor.

MOS	Opinião
5	Excelente
4	Bom
3	Razoável
2	Pobre
1	Ruim

Tabela 3.1 – Opiniões correspondentes a diferentes valores de MOS

A recomendação P.800.1 faz uma discriminação do MOS em MOS-LQ e MOS-CQ, em que LQ (*Listening Quality*) representa um parâmetro de qualidade na escuta apenas e CQ (*Conversational Quality*) representa um parâmetro de qualidade na conversação. Cada tipo é dividido ainda em S (*Subjective*), O (*Objective*) e E (*Estimated*). Assim, o MOS é subdividido em seis tipos diferentes.

A Figura 3.3, retirada da recomendação G.108, mostra um esquema com a relação entre os diferentes tipos de MOS e a maneira de se determinar cada um.

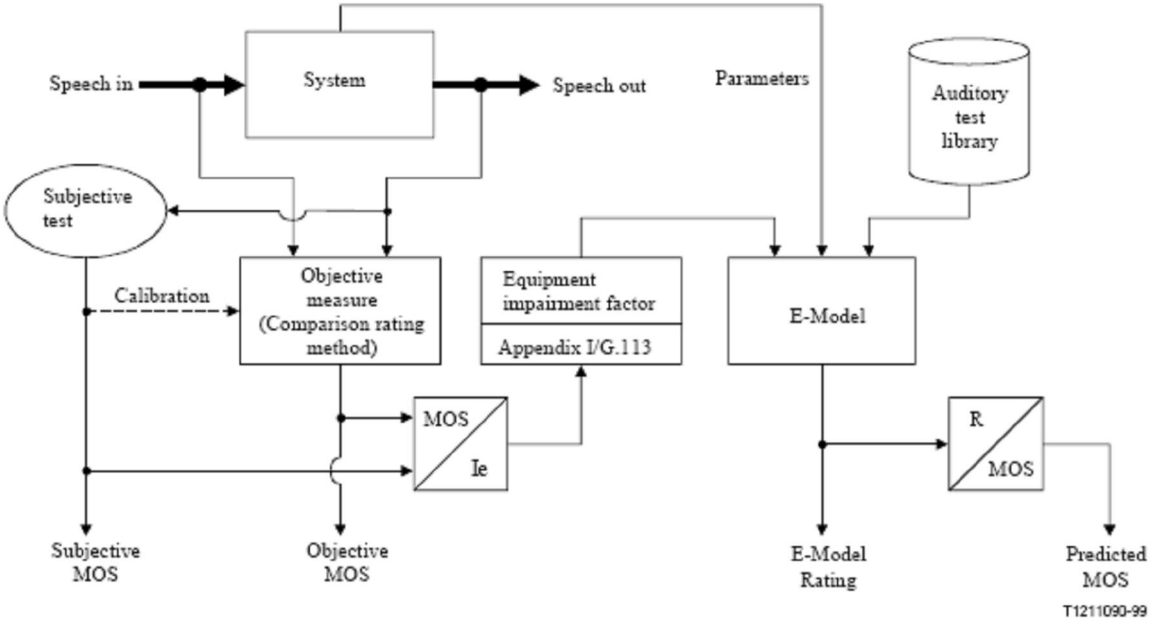

Figura 3.3 – Relação entre alguns qualificadores de MOS. [12]

3.6 – MODELO E

A recomendação G.107 da ITU-T apresenta um modelo computacional para análise preditiva de um sistema de comunicação de voz. Este modelo é chamado Modelo E e foi desenvolvido pelo ETSI em 1996. Sua versão mais recente data de 2005. Ele foi projetado inicialmente apenas para a fase de implantação dos serviços de telefonia. Em 2000, a ETSI propôs uma versão do método para o monitoramento de redes já existentes. Esta versão ainda não foi incorporada pela ITU-T.

O parâmetro de saída do Modelo E é o *Rating Factor – R*. Ele varia de 0 a 100 segundo a Tabela 3.2.

R	Categoria de qualidade
90-100	Muito boa
80-90	Boa
70-80	Razoável
60-70	Baixa
0-60	Pobre

Tabela 3.2 – Relação entre *R* e categorias de qualidade

E é calculado da seguinte maneira:

$$R = Ro - ls - ld - le + A \tag{3.3}$$

Cada parâmetro da equação vem explicado a seguir:

- *Basic signal-to-noise ratio, Ro*

Ro é a relação sinal-ruído básica, incluindo ruídos de circuitos e da sala em que é feita a chamada. Ele é calculado como:

$$Ro = 15 - 1{,}5(SLR + No) \tag{3.4}$$

em que *No* é a adição de potência de diferentes fontes e *SLR* (*Send Loudness Rating*) é a taxa de sonoridade do emissor. *SLR* é tabelado e o cálculo de *No* também envolve parâmetros tabelados. Para uma chamada VoIP típica, considera-se o valor de *Ro* pode ser, em geral, constante e igual a 94,8 [4].

- *Simultâneos impairment factor, Is*

Este é o fator de perdas simultâneas e é a soma das perdas devidas a volume excessivamente alto, perdas causadas por problemas de efeito local e perdas devidas a distorção de quantização na digitalização do sinal de voz. Quando calculado com valores típicos fornecidos pela versão mais recente da G.107, *Is* é igual a 1,41 [4].

- *Delay impairment factor, Id*

Fator de perdas causadas pelos atrasos e ecos ocorridos na comunicação. Envolve tanto os atrasos no recebimento dos pacotes quanto os atrasos introduzidos pelos codecs. Tipicamente pouco significativo no valor total de *R* [17], é calculado pela equação (3.5).

$$Id = Idte + Idle + Idd \tag{3.5}$$

Idte e *Idle* são as perdas devido a ecos no transmissor e receptor, respectivamente. É possível calcular estas duas perdas por valores tabelados. *Idd* representa as perdas devido ao atraso fim a fim, para atrasos maiores que 100 ms.

- *Equipment impairment factor, Ie*

Ie é o fator de perdas causadas por equipamentos, no caso, os codecs. Os valores de *Ie* dependem de testes subjetivos (MOS). A recomendação G.113 Apêndice I apresenta tabelas com valores de *Ie* para diferentes codecs sob diferentes condições de perdas de pacotes e de erros na transmissão.

- *Advantage factor, A*

O fator de vantagem é usado para expressar o grau de tolerância que o usuário espera para determinada tecnologia. Por exemplo, a G.107 recomenda que *A* seja igual a 20 em ligações via satélite, em que é sabida a dificuldade da realização da chamada. Para ligações

VoIP, *A* é igual a zero. Entretanto, esse valor pode ser considerado como uma variável cultural e pode assumir valores positivos [4].

3.6.1 – RELAÇÃO ENTRE MOS E *R*

A recomendação G.107 traz ainda a relação entre *R* e o MOS-CQE, por meio das seguintes fórmulas:

$$\begin{aligned} &\text{Para } R < 0: && \text{MOS} = 1 \\ &\text{Para } 0 < R < 100: && \text{MOS} = 1 + 0{,}035R + R(R-60)(100-R)\cdot 7\cdot 10^{-6} \\ &\text{Para } R > 100: && \text{MOS} = 4{,}5 \end{aligned} \tag{3.6}$$

O gráfico da Figura 3.4, retirado de G.107, mostra a relação dos dois parâmetros.

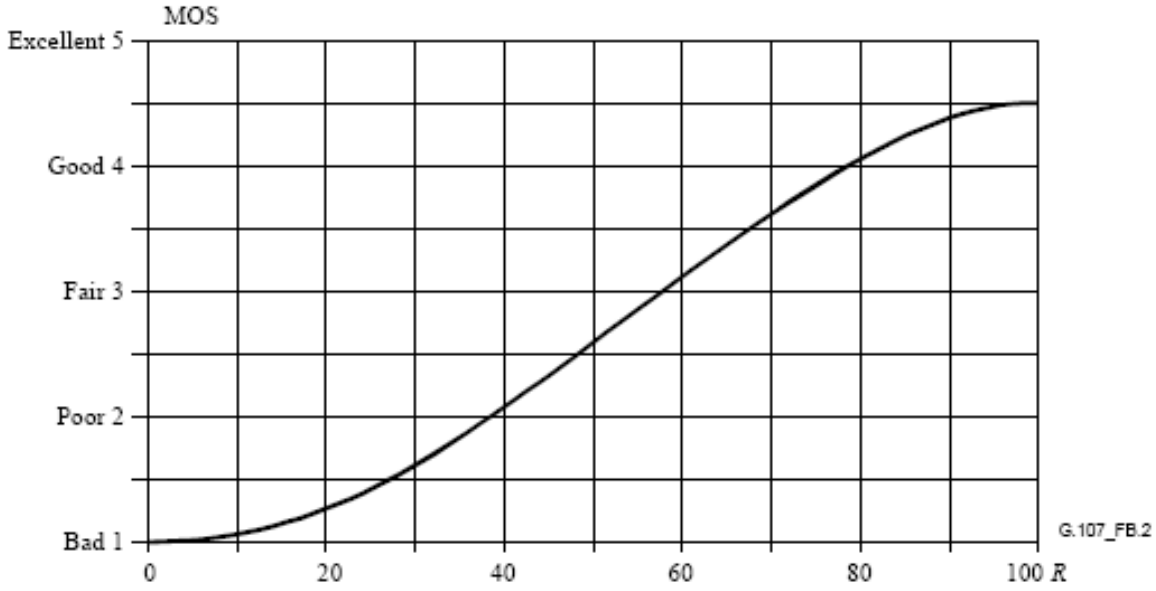

Figura 3.4 – Relação entre MOS e *R*. [11]

4 – IMPLEMENTAÇÃO DO SOFTWARE

Em posse dos conceitos acerca das medidas de qualidade e dos processos envolvidos no estabelecimento da chamada, iniciou-se a criação do módulo de software para o TSW800TP. Descreveremos, primeiramente, as características gerais deste *test set*. Depois, será analisado sumariamente o aplicativo `pjsua` e narrada toda a trajetória para programar o módulo de teste de VoIP. Por fim, será apresentado o resultado "visual" do trabalho, telas e funcionalidades do módulo.

4.1 – TSW800TP

O TSW800TP é um equipamento de teste de serviços *Triple Play*. Ele é utilizado para instalação, manutenção ou reparo de linhas ADSL, ADSL2 e ADSL2+, para que as operadoras de telecomunicação certifiquem a qualidade de serviços de transmissão de dados, áudio (VoIP) e vídeo (IPTV). Testes de qualidade de VoIP são o objeto de estudo do presente trabalho e ainda não estão disponíveis ao usuário. A foto do TSW800TP vem na Figura 4.1.

Figura 4.1 – TSW800TP

A versão atual do hardware TSW800TP consiste em cinco placas de circuito impresso que englobam os seguintes componentes de hardware: modem ADSL, CPU, LCD, teclado alfanumérico, interface Ethernet (IEEE 802.3/802.3u *Fast Ethernet*) e interface serial RS-232. O equipamento usado nos testes de VoIP conta ainda com interfaces de áudio (microfone e fone de ouvido Jack 2,5mm), interface BlueTooth, interface infravermelho (IrDA), outra interface Ethernet e um cartão SD. A Figura 4.2 mostra o painel traseiro deste hardware de testes.

Figura 4.2 – Painel traseiro do hardware para testes de VoIP

A CPU do equipamento consiste de um processador i.MX21® cujo núcleo é composto do microprocessador ARM926EJ-S® fabricado pela Freescale®. Este processador de arquitetura RISC pode chegar a 266MHz e possui baixo consumo de potência. Ele possui barramento de 32 bits e suas instruções podem ser de 32 bits ou de 16 bits (instruções Thumb®). Como podemos ver em [8], o i.MX21® possui várias funcionalidades de hardware tais como *Digital Audio Mux*, *enhanced Multimedia Accelerator* (eMMA), controlador USB On-The-Go, entre outras.

A Freescale® fornece em seu sítio diversificado suporte de software aos seus produtos. O TSW800TP tem o sistema operacional *Linux Board Support Package* (BSP) cujo Kernel é o da versão 2.4.20. O sistema de arquivos e o Kernel ficam armazenados na memória *flash* de 24 MB e o equipamento também possui memória RAM de 32 MB, responsável pela

execução das aplicações. O módulo do Kernel responsável pelo áudio é o `dbmx-wm8731-audio`, fornecido pela Freescale®.

Para instalar o módulo de áudio deve-se executar o comando `insmod dbmx-wm8731-audio` e criar dois *links* simbólicos no diretório `/dev` utilizando os comandos `ln -s /dev/sound/dsp /dev/dsp` e `ln -s /dev/sound/mixer /dev/mixer`.

Tudo o que é mostrado no LCD do TSW800TP provém de dois aplicativos que rodam sobre o Linux: `tswzGui` e `konqueror`. O `konqueror` é um dos principais navegadores do Linux e é usado no teste de Internet. O `tswzGui` é um aplicativo programado em C++ usando bibliotecas da interface gráfica do sistema multiplataforma Qt 2.3. Toda a gerência de telas e interface com o usuário é tarefa do `tswzGui`.

A conexão ADSL se dá por meio de um modem embutido no equipamento. A transmissão de dados para a rede ADSL se dá pela transmissão de pacotes Ethernet ao modem. Ou seja, o modem é de certa forma independente do processador e apenas modula os pacotes Ethernet que chegam a ele. Assim, para programar uma aplicação que envie dados pela rede ADSL, basta configurar corretamente o modem, conectar a linha ADSL no equipamento e enviar os pacotes via Ethernet para o modem. O esquema da Figura 4.3 esclarece esta situação.

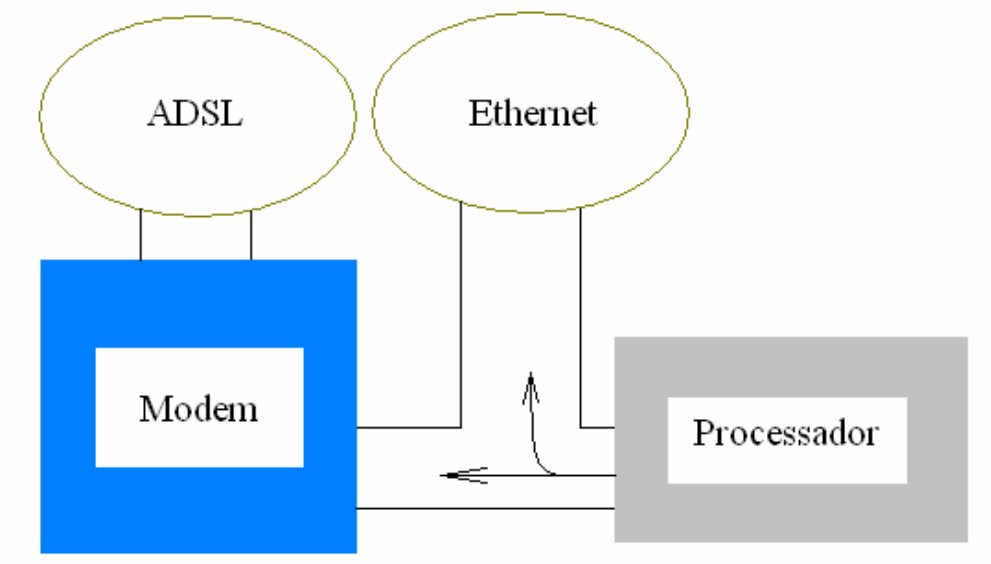

Figura 4.3 – Esquema de funcionamento da interface ADSL do TSW800TP

Desta forma, todo o desenvolvimento foi baseado na interface Ethernet e não foram feitos testes mais profundos na interface ADSL.

4.2 – FERRAMENTAS DE SOFTWARE

Serão descritos agora dois elementos essenciais no desenvolvimento do software deste trabalho.

4.2.1 – *PIPE*

Pipe ou "canalização" é um mecanismo para comunicação entre interprocessos; um processo escreve dados em um lado do "cano" e outro processo lê esses dados do outro lado. Ele é criado para comunicação entre processos "pai" e "filho" (subprocesso) ou entre dois processos "irmãos". Será descrito aqui o *pipe* da biblioteca C do GNU.

A função que cria o *pipe* é declarada em `unistd.h` e possui o escopo mostrado em (4.1). Ela cria o *pipe* e coloca os descritores de arquivo (*file descriptor*) para ler e escrever nas pontas do *pipe*, `filedes[0]` e `filedes[1]`, respectivamente. Quando ocorre um erro, `pipe` retorna `-1`; caso contrário, retorna `0`.

```
int pipe(filedes[2])
```
(4.1)

Para criar o subprocesso, foi utilizada a função `fork`, também de `unistd.h`, mostrada em (4.2). Esta função retorna o tipo inteiro `pid_t`, que é 0 no processo filho e 1 no processo pai.

```
pid_t fork(void)
```
(4.2)

Um exemplo simples a respeito dessas funções pode ser encontrado em [9].

4.2.2 – PCAP

Com o intuito de observar e capturar o tráfego de pacotes de uma rede, foram criados vários aplicativos e bibliotecas de capturas de pacotes, entre eles a biblioteca em linguagem C `libpcap`, desenvolvida pelo Network Research Group do Lawrence Berkeley Laboratory.

Basicamente, para capturar pacotes usando `libpcap` deve-se primeiro definir a interface de rede (`eth0`, por exemplo) e o filtro que vai permitir exclusão na captura de pacotes que não interessam. Por fim, programam-se as funções que serão executadas a cada pacote capturado, por exemplo, um método que calcule o *jitter* a partir dos dados do pacote RTP [3].

4.3 – APLICATIVO `PJSUA`

Para a realização das chamadas e registro no servidor *proxy*, procuramos por um aplicativo (*softphone*) de software livre que pudesse ser cross-compilado para ARM9. O escolhido foi o `pjsua`, aplicativo escrito na linguagem C a partir das bibliotecas do projeto PJSIP.

Dentre as muitas funcionalidades e características do `pjsua`, podemos destacar:

- Transporte UDP, TCP e TLS.
- Resolução de DNS para servidores SIP.
- Múltiplas chamadas.
- Chamadas em multiconferência.
- Codecs Speex, iLBC, GSM, G.711, G.722 e L16.
- Transmissão e gravação de arquivo WAV.
- Uso de RTCP e monitoração de qualidade (medidas de perdas de pacotes, *jitter* e RTT).
- Geração de tons.
- Resposta automática (*auto-answer*).
- Cancelamento de eco.
- *Buffer* adaptativo de *jitter*.
- Uso de PLC.
- Simulação de perdas de pacote.

- Suporte a SRTP.

A versão do `pjsua` utilizada nos testes foi a 0.9.0, que era a versão mais atual na época do início do projeto. Pode-se adquirir livremente todo o pacote do software no sítio do projeto [18] em formato `tar.bz2`. Para cross-compilar o `pjsua` para ARM9 em um ambiente Linux, deve-se decompactar o pacote de software, dirigir-se ao diretório `pjproject/` e digitar o comando `./configure --host=arm-linux`. Este comando irá definir as variáveis de ambiente necessárias à execução dos programas na arquitetura escolhida. Vale lembrar que é necessário instalar o cross-compilador `arm-linux` no ambiente de trabalho. Terminado este processo, digita-se `make dep && make` para que o compilador crie os binários dos programas. O executável `pjsua` gerado possui o tamanho de 1,2 MB.

Os arquivos (`pjsua` e bibliotecas) devem, então, ser transferidos para a plataforma embarcada, para serem executados. Há duas maneiras principais de fazer isso: via serial ou via Ethernet. A maneira mais rápida encontrada para realizar esta tarefa foi transferir dois pacotes compactados via Ethernet por meio do protocolo TFTP: um com o `pjsua` e outro com as bibliotecas. Foram criados *shell scripts* com o intuito de realizar essas tarefas automaticamente.

Para executar o `pjsua` num terminal Linux é só estar no diretório em que ele está alocado e digitar o comando `./pjsua` seguido das opções e do número que se deseja chamar. Se o número for omitido, o programa apenas iniciará e fará o registro no *proxy*, caso haja a opção com uma conta a se registrar. Optamos por executar o aplicativo na memória RAM, visto que o acesso a ela é mais rápido do que o acesso à memória *flash*. Entretanto, devemos armazenar uma cópia do executável na memória *flash*, pois a memória RAM perde os dados quando desligamos o equipamento.

O `pjsua` também oferece a opção de se colocar todas as opções em um arquivo e passar o nome do arquivo como opção de entrada. Assim, executamos o `pjsua` alocado na memória RAM (montada em `/tmp`) pelo comando `/tmp/pjsua --config-`

`file=tsw.cfg` <*url_sip*>, sendo `tsw.cfg` um arquivo de texto simples com todas as configurações iniciais desejadas ao programa e *url_sip* um endereço SIP com o qual se deseja iniciar uma chamada.

Dentre as configurações iniciais possíveis, destacam-se aquelas que foram usadas, essencialmente ou não, durante o projeto:

- `--registrar=url` – define a URL do servidor registrador
- `--id=url` – define a URL local
- `--realm=string` – define o domínio
- `--username=string` – define o nome de usuário para autenticação
- `--password=string` – define a senha para autenticação
- `--dis-codec=name` – desabilita o codec especificado
- `--clock-rate=N` – força o relógio do sistema para `N` Hz
- `--no-vad` – desabilita detector de silêncio (VAD)
- `--ec-tail=MSEC` – define o comprimento do final do cancelador de eco
- `--rtp-port=N` – configura a porta dos pacotes RTP como sendo igual a `N`
- `--auto-answer=code` – responde automaticamente a convites de chamadas remotas com o código `code` (“200 OK”, por exemplo).

O `pjsua` não possui interface gráfica. A aparência do menu de opções em um terminal é mostrada na Figura 4.4. Outra característica interessante é que o `pjsua` mostra na tela os eventos que ocorrem: início e término da chamada, do registro, chamando, etc. Além disso, o `pjsua` pode salvar o *log* de eventos em um arquivo.

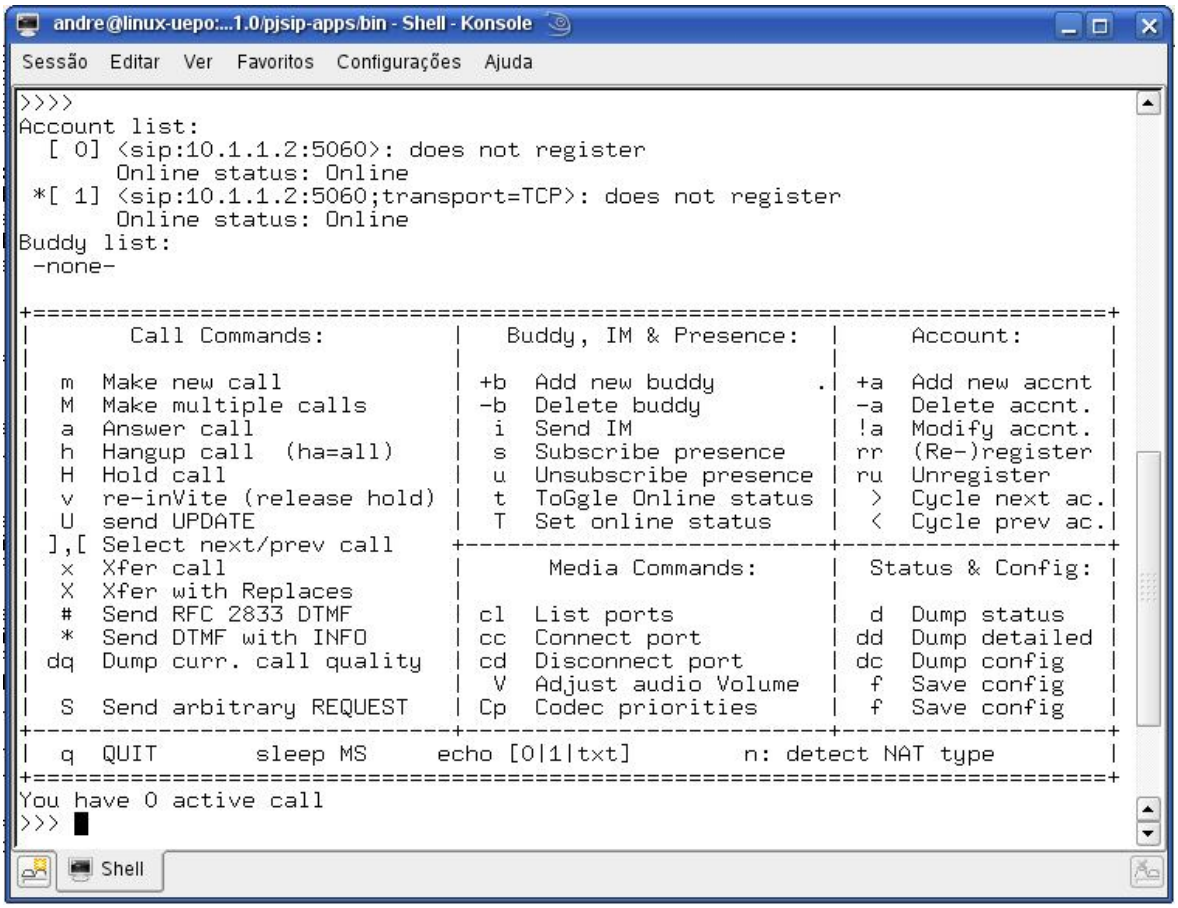

Figura 4.4 – Aparência do `pjsua` em um terminal Linux

Principais comandos de usuário usados durante o trabalho:

- `m` – faz uma chamada para uma determinada URL.
- `a` – responde à chamada. O usuário deve explicitar a resposta com o código SIP desejado: “200” se ele deseja atender o convite, “400” se deseja recusar, etc.
- `h` – encerra a chamada corrente (`ha` encerra todas as chamadas).
- `dq` – mostra as medidas de qualidade da chamada (Figura 4.5).
- `V` – ajusta o volume do microfone e do fone de ouvido.
- `Cp` – muda as prioridades dos codecs.

```
>>> dq
16:26:07.617    pjsua_app.c
 [CONFIRMED] To: <sip:10.1.1.2>;tag=1370017a-f275-4c36-aed1-3d883f67ccfb
   Call time: 00h:00m:14s, 1st res in 9378 ms, conn in 9380ms
   SRTP status: Not active Crypto-suite: (null)
   #0 speex @16KHz, sendrecv, peer=-
      RX pt=103, stat last update: never
         total 1pkt 0B (40B +IP hdr) @avg=0bps/21bps
         pkt loss=0 (0.0%), discrd=0 (0.0%), dup=0 (0.0%), reord=0 (0.0%)
               (msec)    min     avg     max     last    dev
         loss period:   0.000   0.000   0.000   0.000   0.000
         jitter     :   0.000   0.000   0.000   0.000   0.000
      TX pt=103, ptime=20ms, stat last update: never
         total 1pkt 0B (40B +IP hdr) @avg 0bps/21bps
         pkt loss=0 (0.0%), dup=0 (0.0%), reorder=0 (0.0%)
               (msec)    min     avg     max     last    dev
         loss period:   0.000   0.000   0.000   0.000   0.000
         jitter     :   0.000   0.000   0.000   0.000   0.000
     RTT msec       :   0.000   0.000   0.000   0.000   0.000
```

Figura 4.5 – Comando `dq`

As medidas de qualidade que o `pjsua` disponibiliza são as seguintes:

- Número de pacotes recebidos, perdidos, descartadas, duplicados e reordenados.
- Tempo de conversação perdido (*loss period*) de receptor e do transmissor mínimo, médio, máximo, atual e desvio padrão em milisegundos.
- *Jitter* do receptor e do transmissor mínimo, médio, máximo, atual e desvio padrão em milisegundos.
- RTT mínimo, médio, máximo, atual e desvio padrão em milisegundos.

4.4 – DESENVOLVIMENTO DO SISTEMA

Em fevereiro de 2008 foi feito um estudo preliminar a respeito de testes de qualidade de VoIP e decidiu-se implementar medidas de *jitter* e perdas de pacote de uma chamada. Medidas de *delay*, MOS e *R* seriam implementadas posteriormente. Sem conhecimento de aplicativos que pudessem rodar na plataforma e realizar as chamadas, imaginou-se programar métodos para enviar pacotes com a sinalização SIP e capturar os pacotes RTP enviados pelo outro terminal para determinar as medidas de qualidade. O outro módulo do TSW800TP, IPTV, realiza seus testes de uma forma semelhante. Assim, adaptaríamos a biblioteca `libMonitor` desse módulo para obterem-se as medidas de qualidade do VoIP.

Esta biblioteca, escrita na linguagem C, é baseada na biblioteca `libpcap`. Como `libMonitor` já possuía análise dos cabeçalhos RTP, o trabalho seria relativamente fácil.

Quando se chegou ao conhecimento de um aplicativo que fizesse chamadas VoIP e retornasse os parâmetros de qualidade (`pjsua`), a idéia de usar `libMonitor` foi abandonada. O trabalho foi iniciado, então, com a interface gráfica (descrita no próximo tópico). Depois, o `pjsua` foi cross-compilado para a arquitetura ARM9 e foram feitos diversos testes para verificar seu desempenho. Testes para verificar a qualidade da chamada foram feitos entre dois terminais `pjsua` na rede interna da Wise, sem registro em um *proxy*, e também foram feitos testes para verificar se o terminal conseguia se registrar.

O primeiro problema encontrado foi que o módulo do Kernel responsável pelo áudio é incompatível com o módulo da buzina do equipamento. A buzina foi desativada e a resolução do problema prorrogada.

O segundo problema foi que os seguintes sinais de erro eram emitidos antes da realização da chamada:

```
14:04:13.990 transport_srtp Failed to initialize libsrtp: algorithm failed test routine
```

```
14:04:14.009 pjsua_call.c Error initializing media channel: algorithm failed test routine [status=259810]
```

Ficou claro que o erro era emitido na inicialização da biblioteca `libsrtp`, responsável pelo suporte ao protocolo SRTP. Antes de inicializar a biblioteca, o `pjsua` tem uma rotina de testes para verificar se o sistema vai funcionar corretamente. O erro era emitido em um dos pontos dessa rotina mesmo com o SRTP desabilitado.

A solução mais rápida e eficiente encontrada para contornar o problema foi comentar o trecho de código referente à verificação do SRTP e recompilar o aplicativo, já que não se pretendia implementar essa funcionalidade num primeiro momento. No futuro, será preciso descobrir as reais causas do problema e atacar as raízes da falha.

Verificou-se que quando ativado o cancelamento de eco e o detector de silêncio do `pjsua` a qualidade da chamada era prejudicada devido ao aumento de consumo do processador. Decidiu-se então desativar estas opções.

Em relação ao desempenho dos codecs, era esperado que nem todos funcionassem devido ao fato de que o processador não tem bom desempenho com operações com ponto flutuante [9], visto que para realizar tais operações ele precisa emular o ponto flutuante. Mesmo assim, foi testado cada codec e detectado que somente os codecs G.711 (PCM Lei μ e Lei A), G.722 (ADPCM) e GSM funcionariam satisfatoriamente.

Depois de concluir a interface gráfica e de se familiarizar com o `pjsua`, iniciou-se tentativas de ligar os dois aplicativos por meio de um *pipe*. Primeiramente, construiu-se um *pipe* de leitura e executou-se o `pjsua` direcionando sua saída padrão para o *pipe* a fim de ler as mensagens enviadas e mostrar na tela o status da ligação.

Essa última etapa foi concluída satisfatoriamente. A tentativa de construir um *pipe* para a escrita na entrada padrão do `pjsua`, entretanto, não obteve sucesso. Sem poder enviar informações ao `pjsua`, não seria possível requisitar as medidas de qualidade ou encerrar uma chamada sem encerrar o `pjsua` ou ajustar o volume durante uma chamada. A solução ideal passaria por fazer um estudo mais aprofundado sobre *pipe* e assim descobrir onde estaria o erro ou usar as bibliotecas do PJSIP para incorporar o papel do `pjsua` ao próprio aplicativo `tswzGui`.

Devido ao fato das duas opções acima demandarem muito tempo de desenvolvimento, optou-se por voltar à idéia original e realizar as chamadas por meio do `pjsua` e capturar os pacotes RTP por meio de `libMonitor`. Além de não conseguir encerrar a chamada sem encerrar o `pjsua` e não ajustar o volume, essa solução nos obriga a responder chamadas apenas pela resposta automática. Outro detalhe é que, se quando se encerra o `pjsua` com uma chamada em andamento, o outro terminal apenas pára de receber os pacotes RTP. Ele não sabe que o terminal não está mais ativo, pois não recebeu o “BYE” característico do fim da chamada.

Para capturar os pacotes, `libMonitor` roda em duas linhas de execução (*threads*), uma para capturar os pacotes recebidos e outra para capturar os pacotes enviados. Cada linha de execução possui um filtro diferente: a primeira filtra o IP de destino e a porta pela qual os pacotes RTP são recebidos; e a segunda filtra o IP da fonte e a porta pela qual os pacotes RTP são recebidos. O dois endereços IP são o mesmo endereço da interface Ethernet do TSW800TP. Exemplo de filtro para capturar os pacotes recebidos está na equação (4.3) e para capturar pacotes enviados está na equação (4.4).

```
src host 192.168.1.2 and src port 4000          (4.3)
dst host 192.168.1.55 and dst port 4000         (4.4)
```

A medida de *delay* seria inicialmente o RTT. Depois, incluiríamos a análise dos atrasos de processamento nos codecs e de *buffer*. Para medir o RTT, precisaríamos capturar os pacotes RTCP enviados e recebidos, retirar os valores dos campos DLSR e LSR e realizar o cálculo descrito em 3.2.2. Por falta de tempo, a medida de *delay* não foi implementada ainda.

4.5 – DESCRIÇÃO DAS TELAS E FUNCIONALIDADES

A idéia foi construir seis telas para todo o módulo de software de teste de uma chamada VoIP: a primeira tela com codec escolhido e o estado e duração da chamada (VOIP PHONE – Figura 4.6); a segunda com os endereços de IP e servidor de DNS recebidos no processo de DHCP (IP RESULTS – Figura 4.7); a terceira com os parâmetros de QoS – pacotes enviados, recebidos, fora de seqüência e perdidos, *jitter* e *delay* (QOS – Figura 4.8); a quarta com o *log* de eventos do teste (LOG – Figura 4.9); a penúltima com os parâmetros de configuração da chamada (VOIP CONFIG – Figura 4.10); e a última tela para o usuário editar o nome do arquivo que salva o teste (SAVE – Figura 4.11).

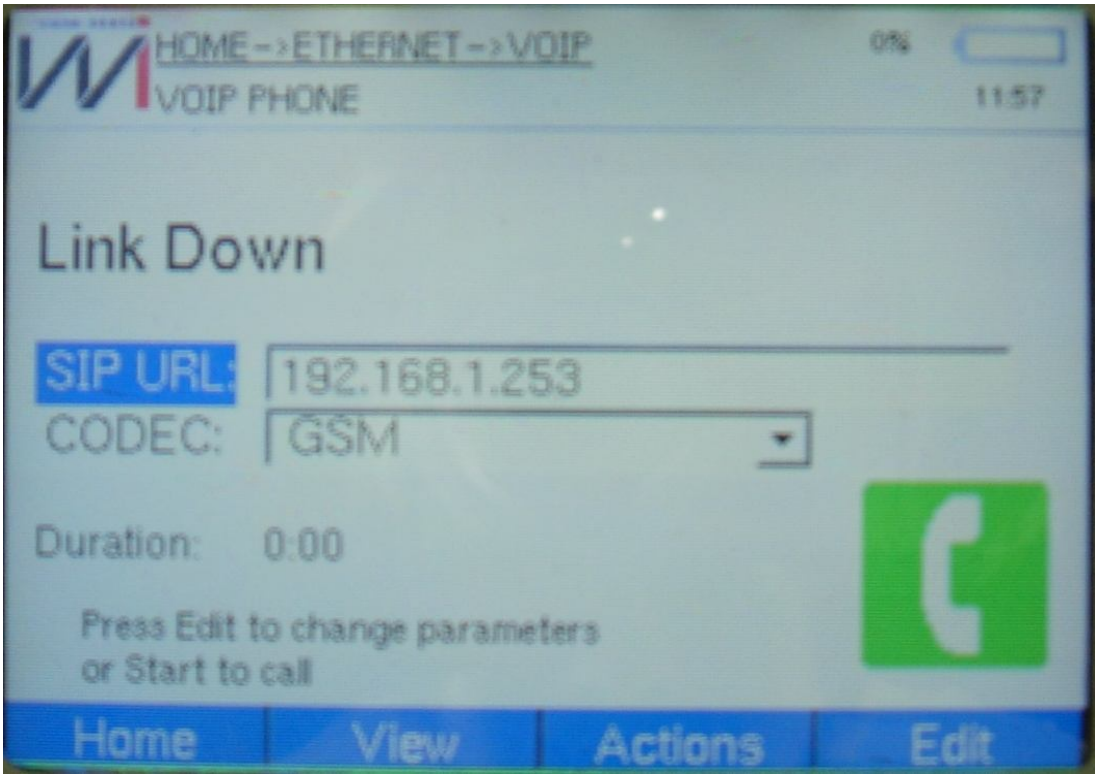

Figura 4.6 – Tela VOIP PHONE

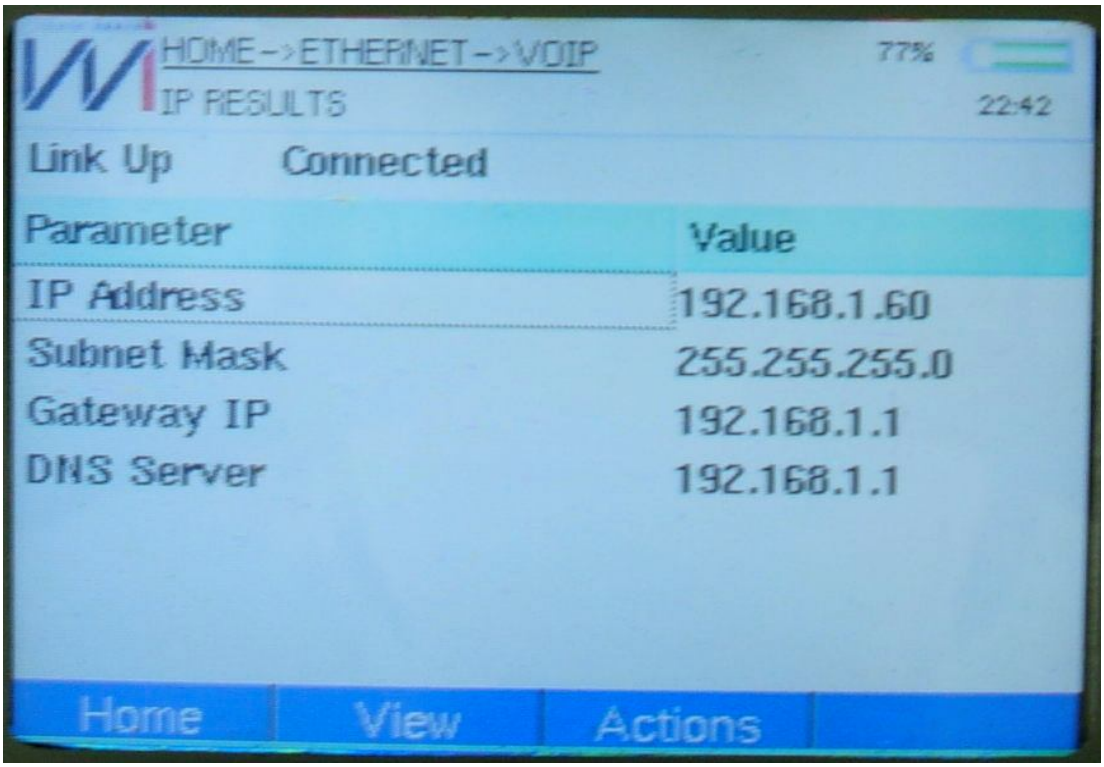

Figura 4.7 – Tela IP RESULTS

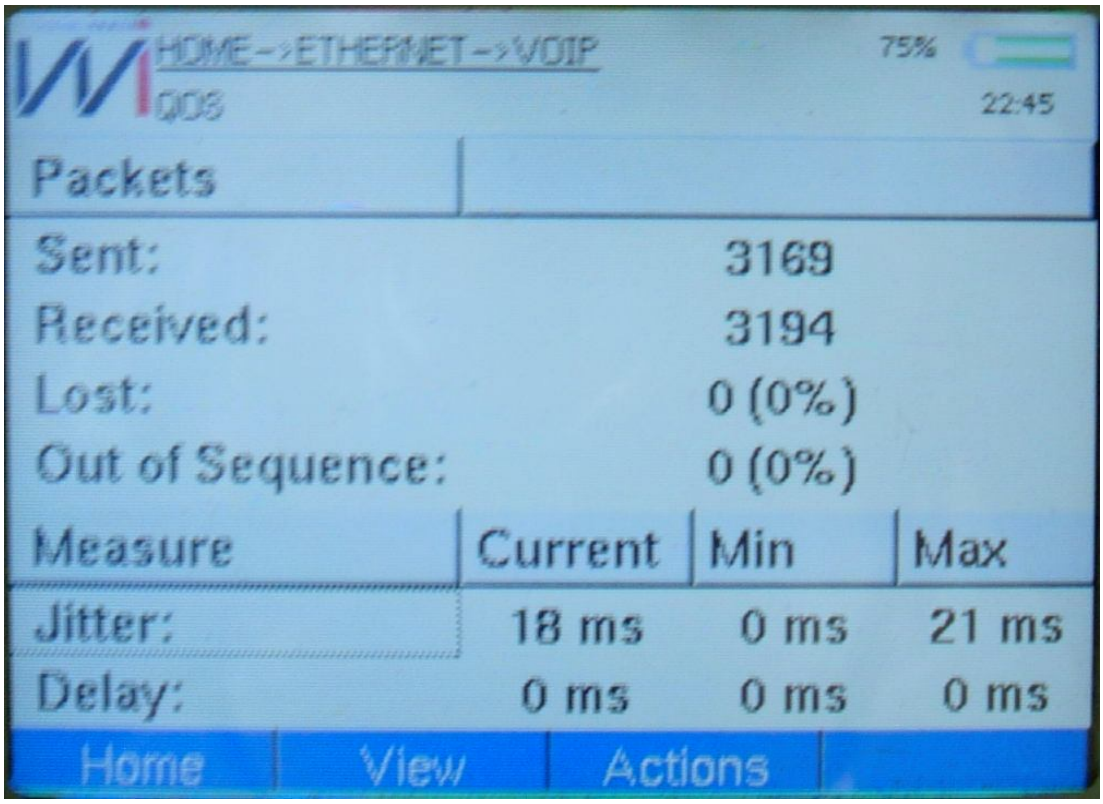

Figura 4.8 – Tela QOS

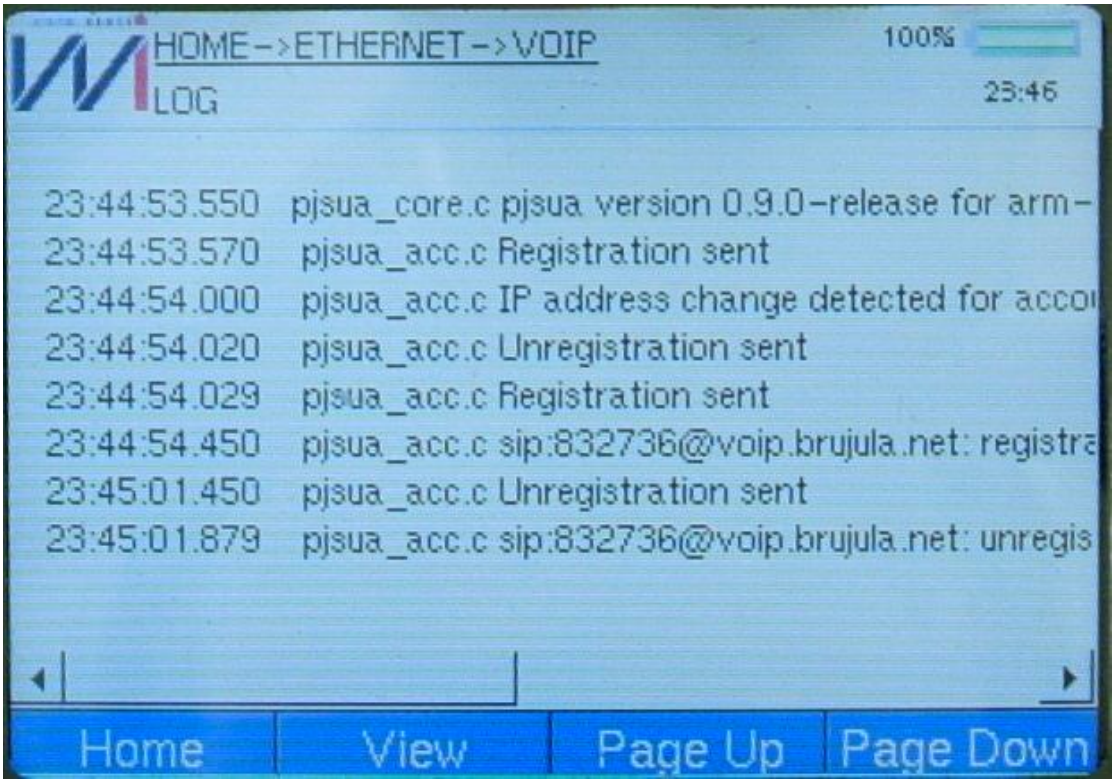

Figura 4.9 – Tela LOG

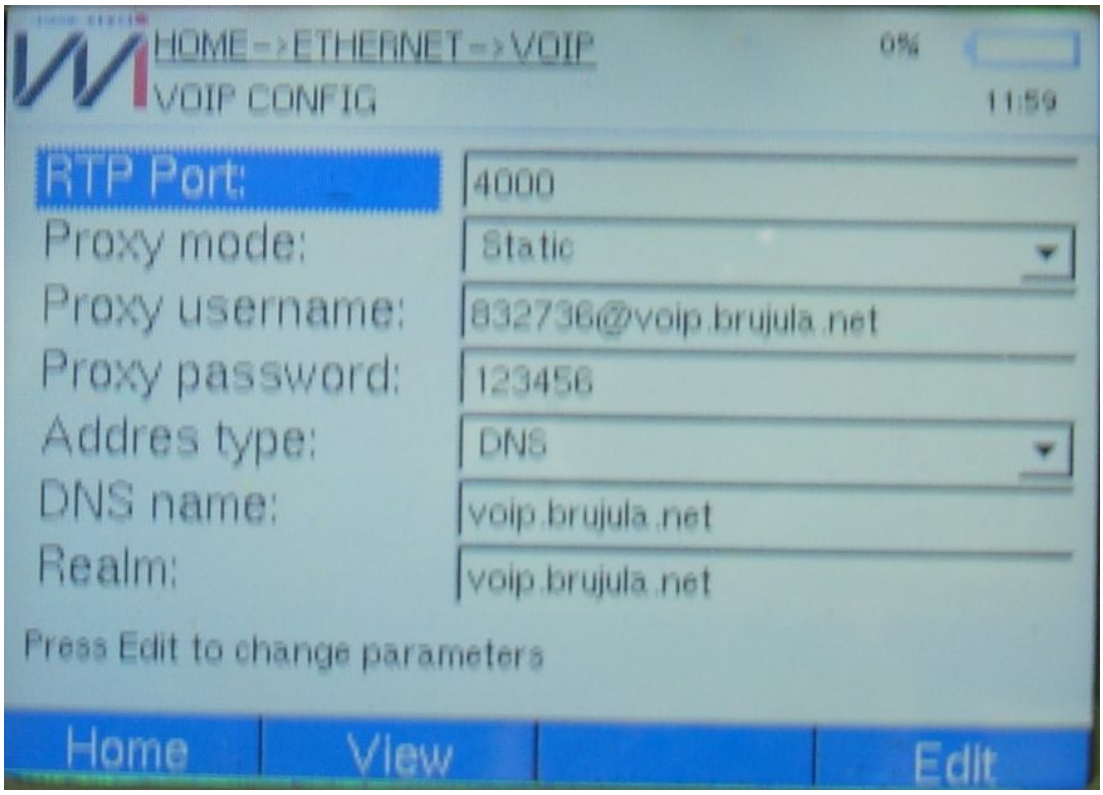

Figura 4.10 – Tela VOIP CONFIG

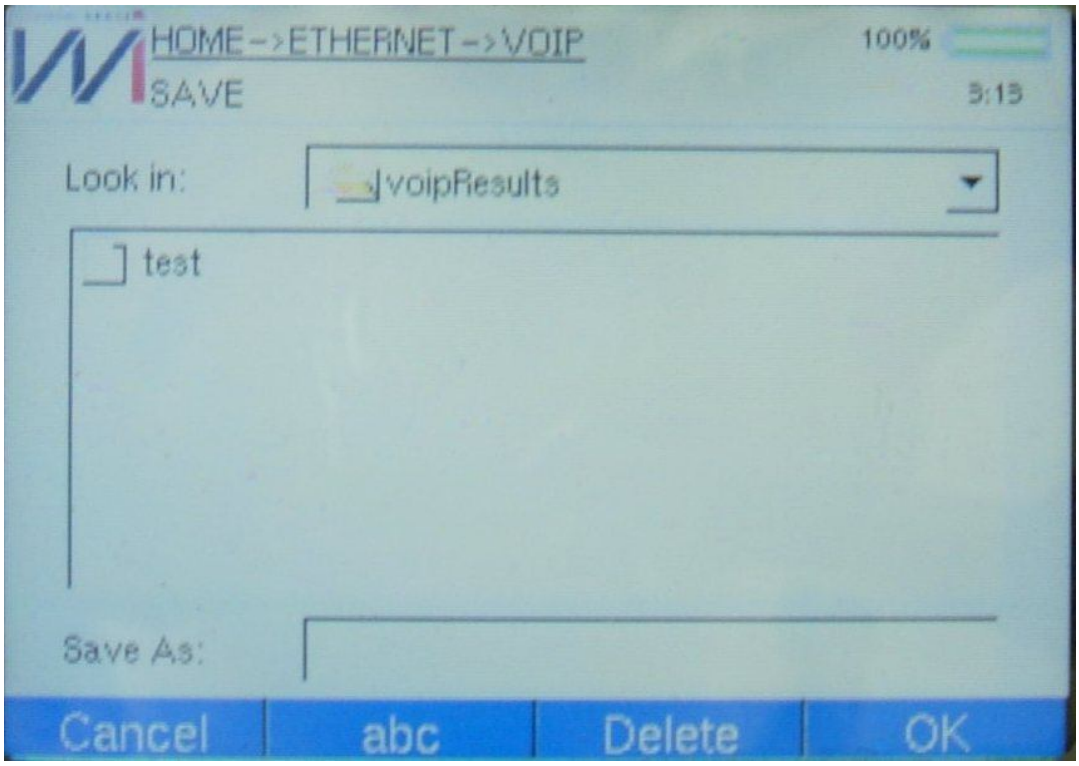

Figura 4.11 – Tela SAVE

Para fazer um teste, o usuário deve primeiramente configurar a interface Ethernet se o teste será via Ethernet, ou ADSL se o teste será via ADSL. Depois disso, ele deve acessar o módulo de teste de VoIP na aba Ethernet ou ADSL e configurar a porta RTP e outros parâmetros SIP na tela VOIP CONFIG. Terminada essa configuração, ele deve voltar à tela PHONE e realizar a chamada. Para visualizar as medidas de qualidade é só acessar a tela QOS. Se o usuário desejar visualizar os eventos das chamadas ele deve ir à tela LOG; se quiser salvar o teste, deve chamar a função “Save” do menu “Actions”.

5 – RESULTADOS

O primeiro resultado de um teste a se considerar é se o UA SIP, no caso o TSW800TP, consegue fazer o registro no servidor. Foram feitos testes de registro no servidor argentino `voip.brujula.net` com total sucesso. As mensagens são mostradas corretamente na tela conforme o andamento do processo. Foram feitos testes tanto pela interface Ethernet na rede interna da Wise quanto pela interface ADSL em uma linha de testes fornecida por uma prestadora de serviços de telecomunicações à empresa.

Na realização das chamadas, enfrentou-se o conhecido problema com a tradução de endereços da rede interna para a Internet [6]. O PJSIP possui uma biblioteca específica para tradução de endereços quando a rede possui NAT (*Network Address Translator*) e o `pjsua` possui funções incorporadas dessa biblioteca. Como o objetivo não era explorar esta problemática, realizaram-se todas as chamadas diretamente entre dois UA da rede interna, sem o uso de *proxy*. Mesmo as chamadas de teste na interface ADSL foram realizadas utilizando o DSLAM ligado a essa mesma rede.

A tela QOS foi criada de forma a mostrar o número de pacotes enviados, recebidos, perdidos e fora de seqüência e os valores de *jitter* de recepção e *delay* mínimos, máximos e atuais. Além disso, os pacotes perdidos e fora de seqüência também têm sua quantidade relativa aos pacotes totais mostrada em percentagem. Os resultados são mostrados de um em um segundo. Apenas o valor de *delay* não está sendo calculado e mostrado. A precisão dos resultados é de um pacote e um por cento para as medidas de pacotes e de um milisegundo para o valor de *jitter*.

A Tabela 5.1 mostra os valores medidos por `libMonitor` de uma chamada de 2,5 minutos utilizando o codec G.711 Lei μ. A Figura 5.1 mostra os resultados medidos pelo `pjsua`. Esta chamada foi feita transmitindo somente o som do ambiente, devido ao fato de um arquivo WAV com essa duração possuir vários megabytes de tamanho. Como o detector de silêncio estava desativado, pacotes RTP eram enviados o tempo inteiro, possibilitando o teste. Entretanto, o teste mostrado possui um cunho de demonstração, visto que não foi vivenciada uma situação de grandes perdas de pacotes e *jitter* elevado. Além disso, não foi feita uma análise entre diferentes codecs.

Pacotes	
Enviados:	7352
Recebidos:	7364
Perdidos:	0
Fora de seqüência:	0
Jitter de recebimento (ms)	
Atual:	19
Mínimo	0
Máximo:	20

Tabela 5.1 – Medidas obtidas por `libMonitor` para uma chamada usando G.711 Lei μ

```
[DISCONNCTD] To: <sip:192.168.1.235>;tag=4d706111-2e7b-4672-ba79-80a2c564f4bb
  Call time: 00h:02m:27s, 1st res in 2895 ms, conn in 2928ms
  SRTP status: Not active Crypto-suite: (null)
  #0 PCMU @8KHz, sendrecv, peer=192.168.1.235:4000
     RX pt=0, stat last update: 00h:00m:00.540s ago
        total 7.3Kpkt 1.17MB (1.47MB +IP hdr) @avg=63.8Kbps/79.8Kbps
        pkt loss=0 (0.0%), dup=0 (0.0%), reorder=0 (0.0%)
              (msec)    min     avg     max     last    dev
        loss period:   0.000   0.000   0.000   0.000   0.000
        jitter     :   0.000  15.011 167.125  19.500   3.405
     TX pt=0, ptime=20ms, stat last update: 00h:00m:01.811s ago
        total 7.3Kpkt 1.17MB (1.47MB +IP hdr) @avg 63.9Kbps/79.9Kbps
        pkt loss=0 (0.0%), dup=0 (0.0%), reorder=0 (0.0%)
              (msec)    min     avg     max     last    dev
        loss period:   0.000   0.000   0.000   0.000   0.000
        jitter     :   0.000  15.424  21.500  15.000   3.661
     RTT msec      :   0.808  11.709  29.406  14.770   7.252
```

Figura 5.1 – Medidas obtidas pelo `pjsua` para uma chamada usando G.711 Lei μ

Comparando os resultados obtidos nas capturas com os valores mostrados pelo `pjsua` no outro terminal, percebe-se que os valores referentes à quantidade de pacotes transmitidos foram muito próximos. Com efeito, a ordem de grandeza é a mesma, mas os dois terminais possuem precisão diferente. Os valores de *jitter* obtidos por `libMonitor` não estão de acordo com os do `pjsua`, conforme percebido pelos seus valores mínimo e máximo. Por falta de tempo no desenvolvimento do trabalho, não foi possível verificar as causas do erro e corrigi-lo. Há que se melhorar também a análise estatística dos dados, conforme o aplicativo `pjsua` faz. Poder-se-ia colocar a média e o desvio padrão do *jitter*.

A Tabela 5.2 mostra o uso de processador e memória RAM dos processos `pjsua` e `tswzGui` durante uma chamada feita com o codec G.711 Lei μ. Estes dados foram obtidos por meio do comando `top` do Linux.

pjsua	
% CPU:	91,5
% Memória:	5,6
tswzGui	
% CPU:	1,6
% Memória:	12,7

Tabela 5.2 – Uso de CPU e de memória para uma chamada usando G.711 Lei μ

O `pjsua` é um aplicativo um tanto robusto que demanda muito processador. Trabalhar no limite pode representar um risco ao desempenho do sistema, pois situações de mais esforço computacional podem ocorrer e indisponibilizá-lo. Por outro lado, `libMonitor` não torna o `tswzGui` um processo pesado. Deve-se verificar se compensa deixar o TSW800TP como um *softphone* e correr riscos de ter medidas duvidosas devido a um baixo desempenho do processador.

O teste, por enquanto, salva somente os endereços IP mostrados na tela IP RESULTS. Futuramente, ele salvará a duração da chamada, o codec utilizado, todas as medidas mostradas em na tela QOS, o *log* de eventos e as configurações SIP.

6 - CONCLUSÕES

O objetivo principal do projeto foi alcançado: a implementação de um protótipo de teste de qualidade de uma chamada VoIP no TSW800TP. Além disso, foram tecidas bases para melhorias do sistema, tais como o cômputo de *R* e de *delay*.

O principal e mais crítico problema encontrado durante o desenvolvimento foi o insucesso na escrita do *pipe* e no acesso às funções do menu do `pjsua`. Sem dúvida, um teste assim seria de certa forma inviável para uma empresa que utilizasse o TSW800TP.

Apesar de paliativa e inevitavelmente provisória, a solução adotada para obter os parâmetros de qualidade permitiu a familiaridade com técnicas de capturas de pacotes e resultou medidas de qualidade satisfatórias. Futuramente, pode-se realizar tais capturas não mais para analisar qualidade, mas para salvar a captura a fim de análise posterior. Esta funcionalidade é observada em concorrentes ao TSW800TP.

Como continuidade do trabalho, levantamos os seguintes pontos principais:

- Compatibilizar os módulos do Kernel referentes à buzina e ao áudio.
- Solucionar corretamente o problema da escrita no *pipe* ou incorporar as funções do `pjsua` ao `tswzGui`.
- Melhorar as análises estatísticas das medidas de qualidade.
- Acrescentar as medidas de *delay*, *R* e MOS.
- Salvar corretamente todos os resultados do teste.
- Implementar, na tela, facilidades do `pjsua` que não foram consideradas.
- Acrescentar outros codecs ao teste.
- Realizar chamadas em outras sinalizações (H.323, MGCP).

Acredita-se que assim que os provedores de serviço de telecomunicações migrarem da RTPC para o VoIP, as bases lançadas aqui serão suficientes para o rápido desenvolvimento das funcionalidades do teste demandadas por este novo paradigma.

BIBLIOGRAFIA

[1] ALUWIHARE. A. et al. **Triple-Play Service Deployment**. 2007, 291p. Disponível em <http://www.jdsu.com/products/communications-test-measurement/forms/triple-play-vice-deployment-form.asp>. Acesso em 25 nov. 2008.

[2] ANDRADE, L. **Análise do Impacto da Migração em Larga Escala dos Serviços de Voz da RTPC para a Rede IP de um Provedor de Serviço.** Publicação PPGENE.DMxxxA/08, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, Jun. 2008, 185p.

[3] CARSTENS**,** T. **Programming with pcap**. <http://www.tcpdump.org/pcap.htm> Acesso em 26 nov. 2008.

[4] CARVALHO, L. **Implementação do Modelo E para Avaliação Objetiva da Qualidade da Fala em Redes de Comunicação VoIP**. Dissertação (Mestrado), Universidade Federal do Amazonas, Instituto de Ciências Exatas, Departamento de Ciência da Computação, Manaus, AM, Set. 2004, 169p.

[5] CISCO. **Cisco AVVID Network Infrastructure Enterprise Quality of Service Design**. Ago. 2002.

[6] **DSL Triple-Play Test Set—CoLT-450P.** <http://www.exfo.com/en/Products/Products.aspx?Id=353> Acesso em 7 Dez. 2008.

[7] FONG, Paul J. et al. **Configuring CISCO Voice Over IP**, 2a ed. USA, Syngress Publishing, 2002. 544p. ISBN 1-93-1836-64-7.

[8] FREESCALE SEMICONDUCTOR. **MC9328MX21**. Data Sheet: Technical Data, Rev.3.2, Jun. 2008, 99 p.

[9] **GNU Libtool Manual** <http://www.gnu.org/software/libtool/manual/libc/Creating-a-Pipe.html> Acesso em 26 nov. 2008.

[10] GOLLO, B. L.; JÚNIOR, J. M. P. R. **Análise do desempenho de codecs de voz usando uma plataforma embarcada com processador ARM e software livre**. Trabalho de graduação em Engenharia Elétrica, Publicação PPGENE.DM, Departamento de Engenharia Elétrica - Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, Dez. 2007, 40 p.

[11] ITU-T Recommendation G.107. **The E-Model, a computational model for use in transmission planning**. Genève, Mar. 2005.

[12] ITU-T Recommendation G.108. **Application of the E-Model: A planning guide**. Genève, Set. 1999.

[13] ITU-T Recommendation G.113 Apendix I. **Provisional planning values for the equipment impairment factor *Ie* and packet-loss robustness factor *Bpl***. Genève, Mai. 2002.

[14] ITU-T Recommendation G.114. **One-way transmission time**. Genève, Mai. 2003.

[15] ITU-T Recommendation P.800. **Methods for subjective determination of transmission quality**. Genève, Ago. 1996.

[16] ITU-T Recommendation P.800.1. **Mean Opinion Score (MOS) Terminology**. Genève, Mar. 2003.

[17] JDSU. **An Explanation of VoIP Statistics.** USA, 2005, 16p. Disponível em <http://www.jdsu.com/product-literature/voipstats_an_acc_tm_ae.pdf>. Acesso em 25 Nov. 2008.

[18] **JDSU HST-3000 Handheld Services Tester** <httphttp://www.jdsu.com/products/communications-test-measurement/products/a-z-product-list/hst-3000.html>. Acesso em 7 Dez. 2008.

[19] LEONEL, J. et al. **Triple Play, um fenômeno sem volta na indústria de telecomunicações**. 2007. Disponível em <www.promon.com.br/portugues/noticias/download/Triple%20play_10.pdf> Acesso em 6 Nov. 2008.

[20] **PJSIP**. <http://www.pjsip.org> Acesso em 26 Nov. 2008.

[21] **RFC 3550**. RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force. SCHULZRINNE, H. et al. Jul 2003. Disponível em <http://rfc-ref.org/RFC-TEXTS/3550/index.html>. Acesso em 25 Nov. 2008.

[22] **RFC 3261**. SIP: Session Iniciation Protocol. Internet Engineering Task Force. ROSENBERG, J. et al. Jun 2002. Disponível em <http://rfc-ref.org/RFC-TEXTS/3261/index.html>. Acesso em 25 Nov. 2008.

[23] TAVARES, G. P. **Implementação de Sistema VoIP em Plataforma Embarcada Utilizando Código Aberto**. Trabalho de graduação em Engenharia Elétrica, Publicação PPGENE.DM, Departamento de Engenharia Elétrica - Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, Jul. 2007, 60 p.