Dherik Barison

Avaliação da qualidade de chamadas VoIP cifradas usando *Mean Opinion Score* e *Traffic Control*

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Telecomunicações e Telemática.

Orientador: Leonardo de Souza Mendes

Campinas, SP
2010

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Barison, Dherik
B239a Avaliação da qualidade de chamadas VoIP cifradas usando Mean Opinion Score e Traffic Control / Dherik Barison. --Campinas, SP: [s.n.], 2010.

Orientador: Leonardo de Souza Mendes.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Criptografia. 2. Redes de computação - Protocolos. 3. Telefonia pela Internet. I. Mendes, Leonardo de Souza. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Evaluation of quality in encrypted VoIP calls using Mean Opinion Score and Traffic Control
Palavras-chave em Inglês: Cryptography, Computer network - Protocols, Internet telephony
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Rodolfo Miranda de Barros, Gean Davis Breda
Data da defesa: 15/12/2010
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Dherik Barison

Data da Defesa: 15 de dezembro de 2010

Título da Tese: "Avaliação da qualidade de chamadas VoIP cifradas usando Mean Opinion Score e Traffic Control"

Prof. Dr. Leonardo de Souza Mendes (Presidente): _____

Prof. Dr. Rodolfo Miranda de Barros: _____

Prof. Dr. Gean Davis Breda: _____

Resumo

A proposta desta dissertação é avaliar a qualidade de chamadas VoIP cifradas com diferentes algoritmos de criptografia através do OpenVPN, com o objetivo de identificar as diferenças de resultados entre os algoritmos de criptografia e também entre as chamadas cifradas e as não cifradas. Esta avaliação ocorrerá utilizando o MOS (*Mean Opinion Score*), um método que permite indicar a satisfação do usuário quanto a qualidade da comunicação. As chamadas VoIP cifradas irão ocorrer em diferentes cenários de rede que apresentam diversos problemas, tais como perda de pacotes, pacotes fora de seqüência, atraso, largura de banda de rede, etc. Estes cenários foram baseados em algumas situações reais de uso e serão emulados através da ferramenta *Traffic Control* do Linux, capaz de manipular os pacotes enviados por qualquer uma das interfaces de rede. Os cenários também terão diferentes larguras de banda de rede, para avaliar a influência das mesmas em algumas situações.

Palavras-chave: VoIP, VPN, MOS, Traffic Control.

Abstract

The purpose of this work is to evaluate the quality of encrypted VoIP calls with different cipher algorithms through OpenVPN software, in order to identify differences in results between encryption algorithms and also differences between non-encrypted and encrypted calls. This evaluation will do by the MOS (Mean Opinion Score), a method to indicate user satisfaction of communication quality. The encrypted VoIP calls will occur in different network scenarios that present different problems, like packet loss, out-of-order packets, delay, network bandwidths, etc. These scenarios were based on some real situations of use and will be emulated with the Traffic Control tool from Linux, able of handling the packages sent by any available network interface. The scenarios will also have different network bandwidths to assess its importance in some situations.

Keywords: VoIP, VPN, MOS, Traffic Control.

Agradecimentos

Ao meu orientador Prof. Leonardo Mendes. Sou grato pela orientação a apoio durante todo o trabalho.

Aos meus colegas Rodrigo S. Miani e Bruno B. Zarpelão, que me auxiliaram na correção deste trabalho.

Aos demais colegas de pós-graduação, pelas críticas e sugestões.

A minha família pelo apoio durante esta jornada.

Aos meus pais, irmãos e amigos

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

CBC - Cipher Block Chaining

IETF - The Internet Engineering Task Force

IPSec - IP Security

MIKEY - Multimedia Internet KEYing

MOS - Mean Opinion Score

PSQM - Perceptual Speech Quality Measure

PSTN - Public Switched Telephone Network

RTCP - RTP Control Protocol

RTP - Real-time Transport Protocol

SIP - Session Initiation Protocol

SRTP - Secure Real-Time Transport Protocol

TBF - Token Bucket Filter

TLS - Transport Layer Security

VoIP - Voice over Internet Protocol

Lista de Símbolos

K_i	-	i-ésima chave
M	-	Mensagem em texto puro
C	-	Mensagem cifrada
C_0	-	i-ésimo processo de cifragem/decifragem
M_0	-	i-ésima mensagem cifrada/decifrada
R_0	-	Fator que expressa a taxa básica de sinal-ruído
I_s	-	Fator que representa todas as perdas que ocorrem simultâneamente com o sinal de voz
I_d	-	Fator somatório de todos as perdas devidas ao atraso e eco. Na prática trata-se apenas do atraso envolvido na transmissão
$I_{e,eff}$	-	Fator de perdas relacionadas a codificador da voz. Estes valores são fixos de acordo com o *codec* usado, e podem ser consultados na recomendação ITU-T G.113 Apêndice I (ITU-T, 2001)
I_e	-	Outra notação para $I_{e,eff}$
A	-	Fator de vantagem.
$\overline{x}$	-	Média aritmética
$\overline{x}_{alg}$	-	Média aritmética do algoritmo de criptografia alg
σ	-	Desvio padrão
c_v	-	Coeficiente de variação percentual

Trabalhos Publicados Pelo Autor

1. Dherik Barison, Rodrigo Sanches Miani, Leonardo de Souza Mendes: "Evaluation of Quality and Security of a VoIP Network based on Asterisk and OpenVPN". *International Conference on Security and Cryptography* (SECRYPT 2009), pg 144-147, Julho 2009.

Capítulo 1

Introdução

O sistema de telefonia vem passando por uma profunda transformação em seu modo de uso. Com o mundo cada vez mais conectado através da Internet, a tecnologia VoIP (*Voice Over IP*) está sendo mais requisitada para realizar as comunicações por voz. Uma pesquisa realizada pela empresa Infonetics Research levantou alguns números de crescimento do uso do VoIP em empresas de pequeno, médio e grande porte (Lincoln, 2009). Estes números podem ser observados na Figura 1.1. A grande vantagem deste sistema está em sua flexibilidade e custo reduzido quando comparado ao sistema de telefonia convencional (Unuth, 2010).

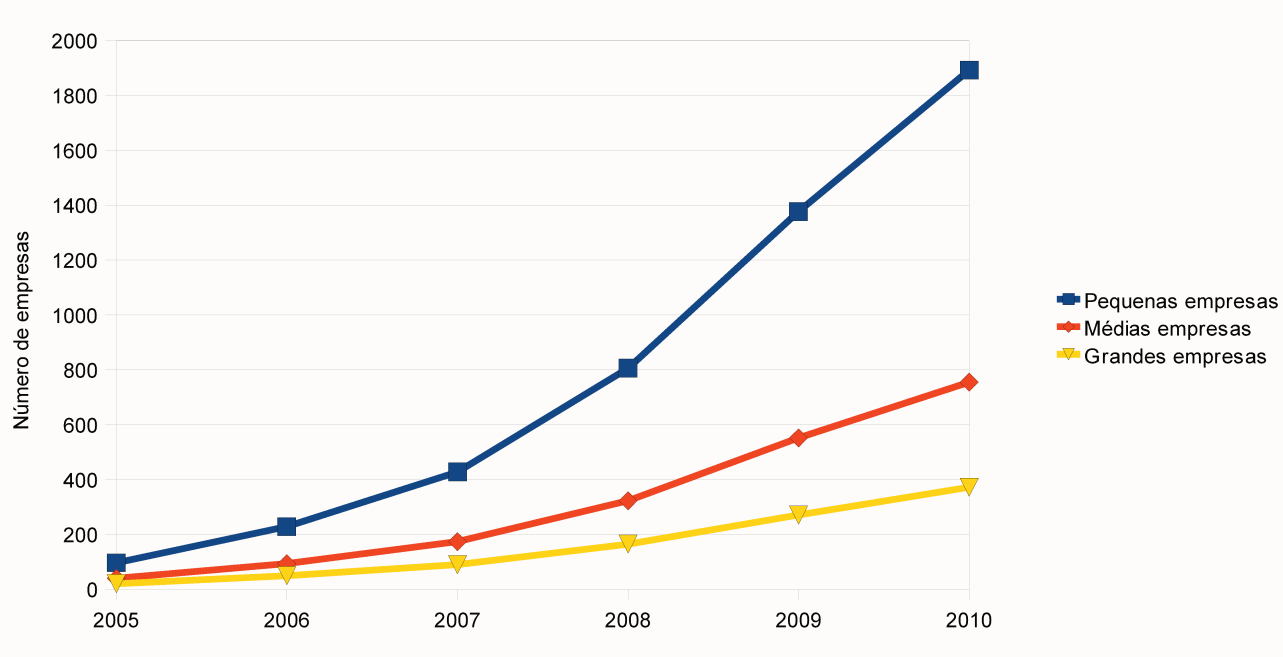

Fig. 1.1: Taxa de crescimento de empresas nos Estados Unidos com serviços VoIP hospedados

Assim como ocorre no sistema de telefonia convencional, no VoIP é possível realizar uma “escuta”

telefônica e ter acesso ao conteúdo da conversa que está trafegando na rede. Na tecnologia VoIP, por usar a arquitetura de rede TCP/IP, este tipo de interceptação de dados é muito mais simples de ser realizada do que no sistema analógico de telefonia, que exige o acesso físico ao cabeamento da rede telefônica. Já a interceptação de uma comunicação VoIP pode ser realizada de qualquer lugar de uma rede, e se esta rede for a Internet isto pode ser feito de qualquer lugar do mundo. A escuta pode ser efetuada com um capturador de pacotes eficaz, como o Wireshark (Wireshark, 2008), que além de capturar estes pacotes também é capaz de montar e reproduzir o conteúdo de aúdio da conversa.

Se o ambiente em questão é uma rede corporativa, e os dados trocados entre as estações não estão protegidos, um funcionário pode facilmente ter acesso a qualquer tipo de informação, incluindo de ligações internas realizadas via VoIP. Nestas condições, é necessário garantir a proteção em tais dados de voz. Uma solução comumente adotada por empresas é a implementação de redes privadas virtuais cifradas como forma de proteger estes dados sigilosos.

Contudo, um problema pode ser observado no uso da criptografia. Ela pode ter um impacto negativo na qualidade da comunicação de voz, pois o atraso gerado ao cifrar os dados no remetente e decifrar os dados no destinatário pode ser percebida pelos participantes a ponto de criar um incômodo aos mesmos ou, no pior caso, impossibilitar a conversa. Aliado a isto, o VoIP cifrado ainda concorre com outros tráfegos na mesma rede. Embora as políticas de Qualidade de Serviço garantam a prioridade dos pacotes de voz em detrimento de outros tipos de dados, existem determinados problemas na rede que nenhum dado trafegado está imune, são eles: perda de pacotes, duplicação de pacotes, corrupção de pacotes, pacotes fora de sequência, atraso e variação do atraso. Assim, é prudente analisar a intensidade deste impacto na transmissão dos dados VoIP cifrados com cada algoritmo criptográfico disponível, e verificar como cada um deles se comportará ao garantir a qualidade da comunicação VoIP, uma vez que realizar testes de desempenho de uma comunicação VoIP criptografada sem qualquer problema na rede, como atraso, não reflete corretamente uma situação real de uso.

O nosso trabalho se baseia neste contexto. Serão emulados diversos problemas comuns em uma rede com a ferramenta NetEm, tais como: perda de pacotes, pacotes fora de ordem, duplicação de pacotes, atraso e variação de atraso com redes de diferentes larguras de banda. Para cada ambiente de rede criado foi dada a designação de “cenário”, e os problemas de rede mencionados serão inseridos nestes cenários. A introdução destes problemas é baseada em dados obtidos de medições em redes privadas e de provedores, pois a intenção é aproximar nossas emulações de ambientes reais. O objetivo é verificar e avaliar o comportamento de diferentes algoritmos criptográficos cifrando a comunicação VoIP nestes cenários, e determinar até que ponto a qualidade da comunicação pode ser afetada e se algum dos algoritmos criptográficos irá se mostrar mais eficiente que outro, nestas situações de rede específicas.

A eficiência será medida usando o *Mean Opinion Score* (MOS) (Bergstra and Middelburg, 2003), método já estabelecido pelo ITU-T (*Telecommunication Standardization Sector*) que permite uma avaliação confiável da qualidade de chamadas realizadas por meio do VoIP. Com base nestes resultados iremos averiguar qual algoritmo se destacou obtendo os melhores resultados e em quais situações isto ocorreu, quais deles tiveram maior variação em seus resultados entre outras contribuições.

Outra contribuição do trabalho será a demonstração prática de como estes cenários podem ser montados usando as ferramentas escolhidas. Acreditamos que muitos pesquisadores e até mesmo administradores de rede podem tirar proveito de nossos testes, podendo conhecer de forma simples, detalhada e objetiva os recursos que o controle de tráfego com a ferramenta *Traffic Control*, com destaque ao recurso NetEm, podem oferecer ao realizar a emulação das condições de uma rede, uma vez que a documentação disponível não é satisfatória e, assim como em outros trabalhos, desejamos agregar experiência e compartilhar o conhecimento obtido.

No Capítulo 2, serão apresentados os trabalhos relacionados, abordando os conceitos de VoIP, Segurança em VoIP, Criptografia e qualidade de voz. O Capítulo 3 citará todos os programas utilizados para realização dos testes propostos. O Capítulo 4 mostrará como os cenários de rede foram montados, como os testes foram executados e os resultados obtidos. Por fim, a conclusão e sugestões de trabalhos futuros poderão ser consultados no Capítulo 5.

Capítulo 2

Trabalhos relacionados

Neste capítulo iremos falar dos principais assuntos e pesquisas relacionadas ao nosso trabalho. Os conceitos que envolvem o VoIP serão apresentados, tais como: protocolos de comunicação em tempo real, tecnologias de codificação de voz, técnicas de avaliação da qualidade de uma chamada VoIP, etc.

Apresentamos, como esperado, uma seção sobre criptografia, com uma introdução mostrando os algoritmos de criptografia AES, Blowfish e Triples DES, que serão utilizados no decorrer dos testes. Também apresentaremos uma introdução ao TLS (*Transport Layer Security*).

2.1 VoIP

O VoIP (*Voice Over IP* - Voz sobre IP) é o processo onde o emprego da telefonia ocorre através da tecnologia IP. É um dos setores da tecnologia de comunicação que mais crescem no momento, tanto que este crescimento ocorre a uma taxa superior a da telefonia móvel (Hersent et al., 2002).

Basicamente, os seguintes passos são executados para que os dados de voz sejam transmitidos por uma rede IP (Barbieri et al., 2002):

1. Digitalização do sinal analógico, usualmente realizada na frequência de 8KHz com 8 bits por amostra, gerando assim um fluxo de dados de 64 Kbps (sem compressão);

2. Geração do pacote do sinal digital conforme os protocolos TCP-UDP/IP;

3. Transmissão dos pacotes através da rede;

4. Recepção do pacote no destinatário e reconstrução do sinal analógico no destino;

Item	**Comutação de circuitos**	**Comutação de pacotes**
Configuração de chamadas	Obrigatória	Não necessária
Caminho físico dedicado	Sim	Não
Cada pacote segue a mesma rota	Sim	Não
Os pacotes chegam em ordem	Sim	Não
A falha de um *switch* é fatal	Sim	Não
Largura de banda disponível	Fixa	Dinâmica
Momento de possível congestionamento	Durante a configuração	Em todos os pacotes
Largura de banda potencialmente desperdiçada	Sim	Não

Tab. 2.1: Comparação entre a rede de comutação de circuitos com a rede de computação de pacotes (Tanenbaum, 2003).

2.1.1 VoIP e PSTN

As redes de telefonia comutadas por circuitos (PSTN - *Public Switched Telephone Network*) foram por muito tempo usadas como principal meio para que as chamadas telefônicas pudessem ser realizadas. Em uma chamada entre 2 participantes, a rede PSTN cria um circuito dedicado entre eles e este circuito só é finalizado quando a chamada termina. Este circuito dedicado corresponde também a um atraso muito baixo na conversação, pois todos os recursos necessários são alocados no início da chamada.

Contudo, o sistema PSTN tradicional evoluiu também. Uma primeira melhoria foi multiplexar várias conversações no mesmo cabo, usando uma frequência de transporte separada para cada sinal (Hersent et al., 2002).

Atualmente temos uma mistura do antigo sistema telefônico analógico com um sistema digital. Embora a linha do usuário possa ser analógica, o sinal é convertido para digital na primeira central telefônica, e o sinal assim permanece até a central telefônica destino, que vai converter o sinal para analógico novamente para que o usuário do outro lado da conversação possa receber a informação de voz.

O VoIP usa a tecnologia de comutação por pacotes. Assim, ao invés de dedicar um circuito todo para uma única chamada, uma transmissão VoIP pode levar pacotes de diferentes conversações em uma única linha. Outra característica importante do VoIP é que a transmissão de informação é toda digitalizada, ou seja, as informações de voz são digitalizadas por meio de um *codec* e são transmitidas.

A Tabela 2.1 faz uma comparação do sistema de comutação de circuitos clássico (sem a parte digital, presente nos sistemas atuais) com o de comutação de pacotes.

De uma forma geral, a única coisa que difere atualmente o VoIP do PSTN é o sinal analógico

que liga o usuário a central telefônica. Conforme mencionado anteriormente, depois da central telefônica toda comunicação é digital, assim como ocorre no VoIP. Contudo, embora compartilhem os mesmos canais, as operadoras de telefonia que também oferecem acesso a Internet tratam (e tarifam) de maneiras diferentes os 2 tráfegos. Uma ligação acaba sendo então sempre mais cara se realizada pelo telefone comum do que pelo VoIP através da Internet, fazendo com que esta torne-se uma opção interessante, mesmo que a qualidade da comunicação não seja a mesma de uma ligação convencional.

Assim, não só usuários como empresas acabam adotando o VoIP através da Internet para amenizar os custos de ligações nacionais e internacionais. Ao instalar um programa VoIP no computador dos participantes da conversação, só é preciso pagar pelo serviço de Internet para se comunicarem, independente de onde estiverem. Se houver a necessidade de realizar ligações de um cliente VoIP para telefones fixos ou telefones celulares, existem empresas VoIP que realizam estas chamadas cobrando tarifas expressivamente menores.

2.1.2 Protocolos VoIP: H.323 e SIP

Quando o VoIP começou a ganhar destaque, era necessário criar algum tipo de protocolo comum, que descrevesse como estabelecer chamadas telefônicas na Internet, videoconferências e outras conexões multimídia (Tanenbaum, 2003). Só assim todos os dispositivos que desejassem participar da comunicação poderiam se intercomunicar. Desta necessidade surgiram 2 protocolos: o H.323 e o SIP.

O H.323 foi criado pelo do ITU-T (*International Telecommunication Union Telecommunication Standardization Sector*), sendo lançado como a recomendação H.323 no ano de 1996, intitulada "*Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service*".

O SIP (*Session Initiation Protocol*) foi projetado pelo IETF (*The Internet Engineering Task Force*) e está descrita na RFC 3261. Ele surgiu após o H.323, no ano de 1999, com o objetivo de se tornar uma alternativa ao mesmo, porém sendo algo mais simples e mais modular de utilizar voz sobre IP (Tanenbaum, 2003).

Ambos os protocolos tem características parecidas e atendem os requisitos para uma rede VoIP. A Tabela 2.2 apresenta uma comparação dos principais atributos de cada protocolo.

Item	**H.323**	**SIP**
Projetado por	ITU	IETF
Compatibilidade com PSTN	Sim	Ampla
Compatibilidade com a Internet	Não	Sim
Arquitetura	Monolítica	Modular
Completeza	Pilha de protocolos completa	Apenas configuração
Negociação de parâmetros	Sim	Sim
Formato de mensagens	Binário	ASCII
Transporte de mídia	RTP/RTCP	RTP/RTCP
Chamadas de vários participantes	Sim	Sim
Conferência de multimídia	Sim	Não
Endereçamento	Número de *host* ou telefone	URL
Término de chamadas	Explícito ou encerramento por TCP	Explícito ou por *timeout*
Transmissão de mensagens instantâneas	Não	Sim
Criptografia	Sim	Sim
Implementação	Grande e complexa	Moderada

Tab. 2.2: Comparação entre H.323 e SIP (Tanenbaum, 2003).

2.1.3 RTP e RTCP

O RTP (*Real-time Transport Protocol*) (Schulzrinne et al., 2003) é um formato de pacote criado pelo IETF no ano de 1996 para transmissão de áudio e vídeo pela rede. Os detalhes de sua estrutura e funcionamento pode ser vistos na RFC 3550.

O RTCP (*RTP Control Protocol*) (Schulzrinne et al., 2003) é um formato de pacote criado pelo IETF na mesma RFC que descreve o pacote RTP (RFC 3350), e sua função é atuar junto com os pacotes RTP, fornecendo informações sobre a transmissão de áudio/vídeo que está ocorrendo no momento. Sua função primária é fornecer informações sobre a qualidade da distribuição de dados (Qualidade de Serviço) aos participantes da comunicação.

O RTP/RTCP é extensamente usado em comunicações por meio da Internet, tanto que é o padrão utilizado pelo H.323 e SIP.

2.1.4 *Codecs* de áudio

Em uma comunicação VoIP os dados de voz dos participantes da conversação precisam trafegar por uma rede TCP/IP, na qual os dados são divididos em pacotes. Estes pacotes carregam dados digitais, ou seja, para que as informações de uma conversação sejam transportadas por pacotes é necessário que ocorra a digitalização do sinal de voz. Esta digitalização é realizada por um *codec* de áudio, que é capaz de codificar uma informação analógica (como a voz de uma pessoa) para o formato digital, e vice-versa.

Contudo, é natural que nesta codificação haja alguma perda de informação. Alguns codecs optam por realizar esta codificação tentando preservar ao máximo a qualidade da voz. Esta escolha resulta em um áudio muito próximo ao original, porém a informação gerada pelo *codec* necessita de uma largura de banda maior.

Outros *codecs* tentam realizar esta codificação tentando comprimir ao máximo o volume final de dados digitais de voz gerados. Embora ocorra uma economia significativa de largura de banda de rede, o resultado pode ser uma qualidade de voz mais baixa e/ou um tempo maior gasto nesta etapa, que dependendo do caso pode gerar um atraso não desejado na conversação VoIP.

A Tabela 2.3 apresenta alguns dos *codecs* mais comuns disponíveis, com algumas de suas características.

Padrão	**G.711**	**G.726 ou G.721**	**G.728**	**G.729**	**G.723.1**
Data de aprovação	1972	1990 (1984)	1992	1995	1995
Taxa de transmissão (em Kbps)	64	16/24/32/40	16	8	6,3-5,3
Qualidade da voz (em MOS)	4,2	2 / 3,2 / 4 / 4,2	4,0	4,0	3,9 / 3,7

Tab. 2.3: Alguns dos *codecs* existentes, com a data de aprovação do seu padrão, taxa de transmissão e o resultado de *Mean Opinion Score* obtido pelo *codec* (Hersent et al., 2002).

Codec G.711

A tecnologia de codificação e decodificação de voz usado em nosso trabalho é o *codec* G.711. Ele é o *codec* obrigatório de muitos outros protocolos, como da especificação H.323 (Tanenbaum, 2003). Também é muito utilizado em redes ISDN (*Integrated Services Digital Network*) e na maioria dos *backbones* telefônicos digitais (Hersent et al., 2002).

O codec G.711 gera um fluxo de dados de 64 Kbps, devido ao processo de digitalização da voz do *codec*. O G.711 captura 8.000 amostras do áudio por segundo, cada amostra contém 8 bits de dados. Ou seja, o *codec* consegue capturar 64.000 bits por segundo de conteúdo de aúdio digitalizado, que

corresponte a 64 kbps. Estes conteúdo é carregado pelo *payload*[1] do pacote. Embora essa taxa de 64 kbps seja uma das maiores entre os *codecs* existentes, isto garante uma alta qualidade da chamada, devido ao pouco atraso na etapa de digitalização dos dados analógicos de voz realizada por este *codec* e pela pouca perda no espectro sonoro da voz no momento da codificação, uma vez que é usado o método de codificação de forma de onda, em que o objetivo é reproduzir com precisão a forma de onda da voz com a menor quantidade de bits possível (Tanenbaum, 2003). Esta etapa de digitalização da voz usa uma escala semilogarítmica, conhecida como PCM (*Companded Pulse Code Modulation*).

A pontuação MOS típica do G.711 é de aproximadamente 4,2 (Hersent et al., 2002), ou seja, ele pouco influencia na qualidade da chamada de voz, o que é ideal para o nosso trabalho, pois assim temos a garantia de que ele não irá interferir negativamente nos resultados de MOS.

2.2 Segurança em VoIP

Basicamente, as mesmas ameaças que afetam as tradicionais redes IP afetam também o VoIP: vírus, ataques de negação de serviço, acesso não-autorizado, etc. Porém, há também as vulnerabilidades específicas do VoIP. As quatro maiores ameaças de segurança do VoIP são (Abdelnur et al., 2006):

- Escuta de uma comunicação de voz privada: uma pessoa não autorizada a participar da conversa consegue, normalmente por meio de um programa capturador de pacotes, ter acesso a conversação;
- Uso abusivo do serviço: chamadas de longa duração ou chamadas constantes (uma após a outra);
- *Spam* sobre VoIP: recebimento de chamadas não-solicitadas;
- Ataques de negação de serviço: funciona exatamente como qualquer ataque do tipo em uma rede de computadores. O equipamento VoIP, que tem um endereço IP, recebe mais requisições do que pode responder fazendo com que o mesmo pare de responder ás requisições legítimas;

Confidencialidade e integridade são um dos principais aspectos da área de segurança em se tratando de VoIP. Um grande número de soluções podem ser consideradas para garantir a confidencialidade e integridade das comunicações VoIP. Algumas destas técnicas (Cao and Malik, 2006):

- IPSec (*IP Security*) fornece serviços de segurança para o tráfego na camada IP. Podendo atuar no modo transporte ou modo túnel, consegue prover autenticação, confidencialidade e integridade;

[1] Trata-se apenas da informação do pacote, sem os cabeçalhos UDP e RTP

- TLS (*Transport Layer Security*) para conexões seguras ponto-a-ponto. Como é o protocolo usado pelo OpenVPN, uma das ferramentas usadas em nossos testes, é possível ver mais detalhes de seu funcionamento na Seção 2.4;

- SRTP (*Secure Real-Time Transport Protocol*) fornece, para o fluxo multimídia, confidencialidade, autenticação das mensagens e proteção contra ataques de repetição (Gupta and Shmatikov, 2007);

- MIKEY (*Multimedia Internet KEYing*) para gerenciar e distribuir as chaves entre os participantes da comunicação, de uma forma simples e eficiente, usando pouca banda de rede e pouco poder computacional (Arkko et al., 2004);

- Distribuição e gerenciamento da distribuição de chaves dependente do protocolo, tal como no SIP e no H.323;

Neste trabalhos iremos estritamente abordar a parte de criptografia usando TLS, pois é a tecnologia usada pelo OpenVPN, programa utilizado em nossos testes para criar a VPN (*Virtual Private Network*) cifrada.

2.3 Criptografia

A criptografia é um ramo da criptologia[2] que lida com o projeto de algoritmos para cifragem e decifragem de informações, no intuito de garantir o segredo e/ou a autenticidade de mensagens (Stallings, 2005).

A maioria das pessoas acredita que o tema criptografia está apenas interessada em manter as comunicações privadas, contudo isto é apenas parte do que é a criptografia. Ao lado da criptografia temos a criptonálise, que consiste no estudo de como comprometer os mecanismos de criptografia (RSA Laboratories, 2000), no intuito de ter acesso as informações cifradas sem conhecer o segredo (chave) usado no momento da criptografia.

A criptografia atualmente tem como objetivo garantir (Tanenbaum, 2003):

- Confidencialidade: manter as informações longe de usuários não-autorizados. É a essência da criptografia;

- Autenticação: processo de determinar com quem o usuário está se comunicando antes de revelar informações sigilosas;

[2]O termo criptologia é dado à disciplina de criptografia e criptoanálise juntas

- Não-repúdio: trata-se de assinaturas das mensagens. O não-repúdio garante que determinada mensagem foi realmente enviada por determinado usuário, ou seja, o usuário que enviou a mensagem não pode negar que a mensagem partiu dele;

- Integridade: garante que a mensagem é realmente legítima e não foi alterada ou criada por um usuário mal-intencionado;

Na área de computação, a criptografia pode ser realizada na camada de rede, na camada de transporte ou na camada de aplicação no TCP/IP.

Na camada de rede temos como opção o uso do protocolo IPSec (IP Security) (Kent and Seo, 2005), desenvolvido pelo IETF (*The Internet Engineering Task Force*), que é capaz de adicionar segurança a todos os dados da comunicação de forma transparente ao usuário, pois a segurança dos dados independe da aplicação. Contudo, o sistema operacional deve ter o suporte ao IPSec habilitado para que ele possa ser utilizado.

Na camada de transporte, temos o protocolo de criptografia TLS (*Transport Layer Security*) (Dierks and Rescorla, 2008), antigo protocolo SSL (*Secure Sockets Layer*) (Freier et al., 1996). Em nosso trabalho usaremos o TLS para realizar a criptografia na camada de transporte, por meio do OpenVPN.

A criptografia na camada de aplicação exige que cada aplicativo cuide da própria segurança dos seus dados transmitidos. Um bom exemplo de programa que fornece segurança na camada de aplicação é o PGP (*Pretty Good Privacy*), que segue o padrão OpenPGP, descrito pela RFC 4880 (Callas et al., 2007).

Existem 2 tipos de sistemas de criptografia: por meio de chave secreta e por chave pública. A primeira é realiza pelos algoritmos chamados “simétricos” e a segunda pelos algoritmos “assimétricos”.

A criptografia simétrica realiza as operações de cifrar e decifrar utilizando a mesma chave secreta. Esta chave secreta deve ser passada para as entidades autorizadas na comunicação de uma forma segura, pois o acesso a chave secreta por um usuário mal-intencionado permitirá que ele possa cifrar e decifrar as mensagens se conseguir um meio de interceptá-las, podendo assim acessar ou mesmo alterar as informações que estão sendo transmitidas. Este compartilhamento da chave secreta deve ocorrer com grande precaução. Normalmente, ela é compartilhada através de um algoritmo assimétrico como o RSA (Rivest et al., 1978). A Figura 2.1 ilustra a comunicação de uma mensagem cifrada entre Ana e Beto, onde ambos utilizam a mesma chave secreta para cifrar e decifrar a mensagem.

Dentre os mais populares algoritmos simétricos podemos citar o Triple DES, AES, Blowfish e DES. Em nosso trabalho iremos usar para realizar a criptografia dos dados o Triple DES, AES e Blowfish. Para cada um destes algoritmos criptográficos criamos uma seção dedicada em nosso trabalho, com uma breve explicação do seu funcionamento e com os critérios que nos levou a optar por cada um deles.

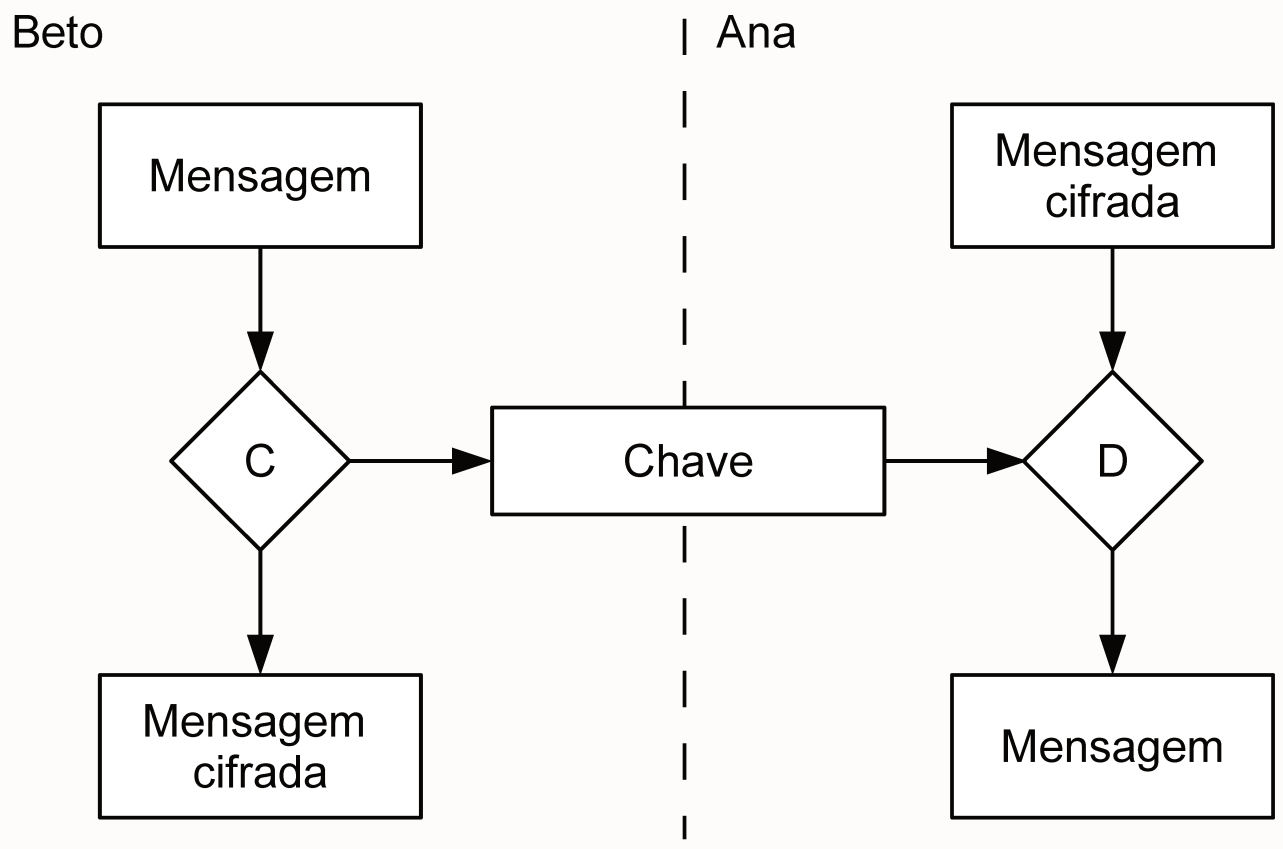

Fig. 2.1: Transmissão de mensagem cifrada utilizando criptografia simétrica entre Ana e Beto.

A criptografia assimétrica consiste no uso de duas chaves: uma nomeada de chave pública e outra de chave privada. Como os nomes nos remetem a induzir, a chave pública é a chave que pode ser compartilhada com os outros participantes da comunicação e não prejudica em nenhum grau a segurança do algoritmo, conforme iremos explicar melhor a seguir. A chave privada, por sua vez, deve ser secreta: só o usuário que é dono da chave pode conhecê-la. Todo participante deverá ter estas duas chaves em posse. A chave pública é usada para cifrar os dados e a chave privada para decifrá-los, diferente dos algoritmos simétricos que usam a mesma chave tanto para cifrar como para decifrar os dados.

Para que a comunicação cifrada possa ocorrer nos algoritmos assimétricos, a participante Ana compartilha sua chave pública com o participante Beto. Este compartilhamento pode ocorrer com a simples disponibilização da chave pública em uma página na Internet ou até por e-mail, por exemplo. Assim, o participante Beto obtém a chave pública da participante Ana e com esta chave cifra os dados confidenciais que deseja proteger do acesso de terceiros. O resultado desta cifragem é uma mensagem ilegível, que pode ser passada em meio inseguro, como a Internet, pois apenas a participante Ana, detentora da chave privada, pode decifrar o conteúdo da mensagem que foi cifrada com sua chave

pública e assim ter acesso a informação da mensagem. A Figura 2.2 ilustra esta comunicação cifrada por meio de chave pública e privada entre Ana e Beto.

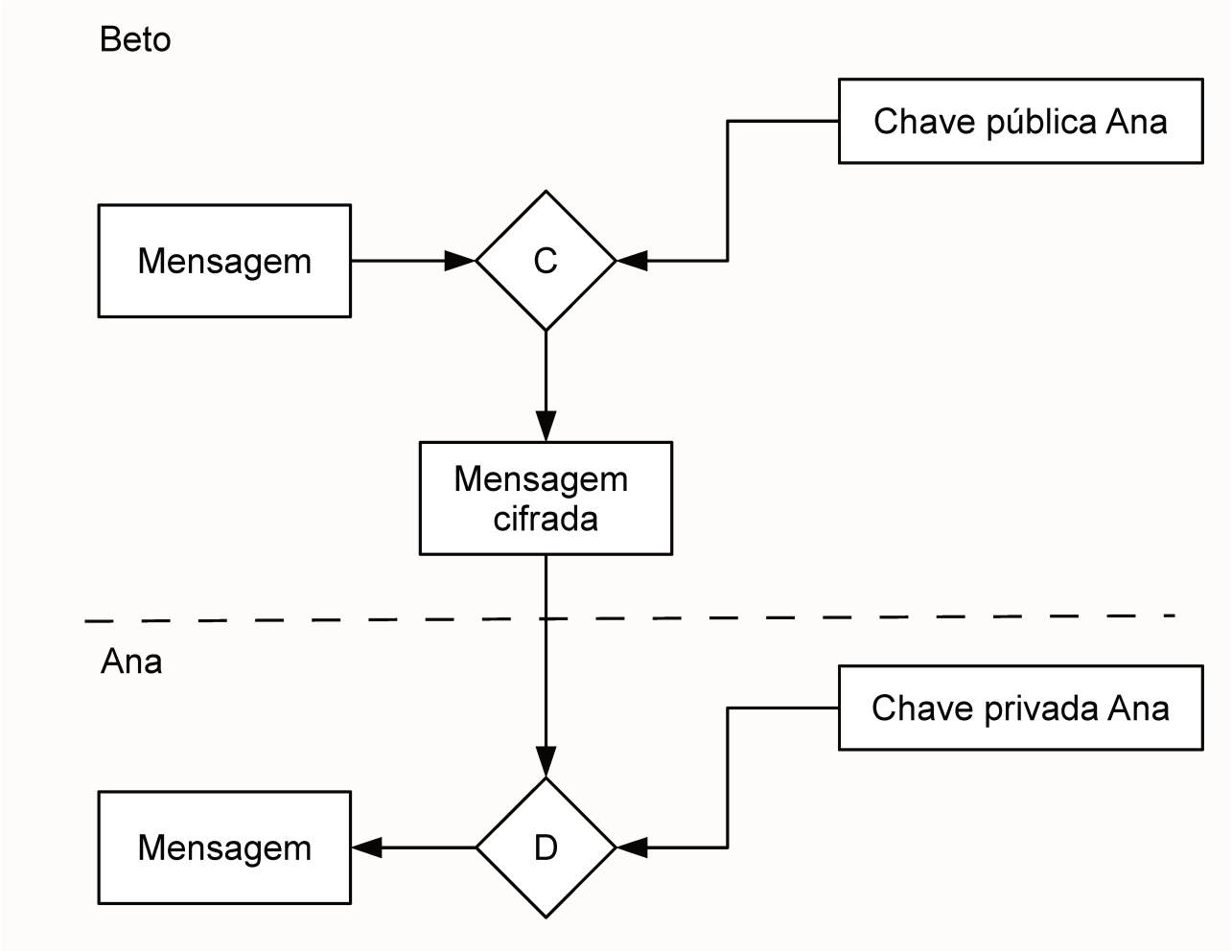

Fig. 2.2: Transmissão de mensagem cifrada utilizando criptografia assimétrica entre Ana e Beto.

Entre os algoritmos assimétricos mais populares podemos mencionar o RSA (Rivest et al., 1978), DSA (*Digital Signature Algorithm*) (National Institute of Standards and Technology, 1994), ECC (*Elliptic Curve Cryptography*) (Miller, 1986) (Koblitz, 1987) e Diffie-Hellman (Diffie and Hellman, 1976).

Outro importante recurso dos algortimos assimétricos é a capacidade de realizar a assinatura de uma mensagem. A assinatura neste caso ocorre quando o usuário dono das chaves pública e privada deseja garantir que determinada mensagem foi enviada por ele. Para fazer isto ele faz o processo inverso da criptografia de chave pública: ele usa sua chave privada para "cifrar" os dados. Neste caso não há sigilo da mensagem, uma vez que qualquer pessoa teoricamente tem acesso a chave pública e pode decifrar a mensagem. Porém apenas a chave pública dele pode decifrar aqueles dados, ou seja,

temos a garantia de que aquela mensagem foi realmente enviada por ele.

É importante lembrar que existem alguns algoritmos assimétricos que não são capazes de cifrar dados, sendo capazes de apenas realizar a assinatura de uma mensagem. Um exemplo de algoritmo com este comportamento é o DSA (*Digital Signature Algorithm*). Os algoritmos RSA e Diffie-Hellman são capazes de realizar tanto a criptografia com a chave pública como com a chave privada. Em nosso trabalho usamos o algoritmos Diffie-Hellman para o compartilhamento da chave privada.

A criptografia em nosso trabalho será realizada por intermédio do OpenVPN (OpenVPN, 2007). Existem vários algoritmos simétricos suportados pelo mesmo, porém escolhemos determinados algoritmos baseados em características peculiares de cada um. Os algoritmos simétricos escolhidos para realizarem a criptografia das chamadas VoIP no trabalho foram o AES, DES e Blowfish, que foram projetados para serem algoritmos cifradores de bloco (mais detalhes na Seção 2.3.4).

O ideal para o trabalho seria usar um algoritmo de criptografia do tipo cifrador de fluxo, já que o conteúdo da VPN (*Virtual Private Network*) no trabalho será de transmissões VoIP. Geralmente, algoritmos cifradores de fluxo não só são mais rápidos que os cifradores de bloco (Robshaw, 1995), mas também mais tolerantes a falhas, pois eles cifram 1 bit por vez, sendo que algoritmos cifradores de bloco cifram um bloco de bits por vez. Assim, um erro no pacote invalida apenas 1 bit com um cifrador de fluxo, enquanto no cifrador de bloco todo o bloco de bits é invalidado, causando mais perda de informação.

Entretanto, o OpenVPN, programa usado em nosso trabalho para cifrar as informações, não suporta nenhum algoritmo cifrador de fluxo. De fato, a indústria evita o uso de algoritmos de fluxo pois trata-se de um padrão pouco definido se comparado aos cifradores de bloco, além de que os cifradores de fluxo são desenvolvidos como uma aproximação do *one-time pad*, gerando certa insegurança teórica (RSA Security, 2009b).

A seguir, iremos descrever os algoritmos criptográficos AES, Tiple DES e Blowfish, que foram usados neste trabalho, trazendo uma introdução sobre cada um deles e alguns detalhes do seu método de operação interna.

2.3.1 AES

O AES (*Advanced Encryption Standard*) é um algoritmo de criptografia cifrador de blocos, criado por Vincent Rijmen e Joan Daemen para um rigoroso concurso do governo dos Estados Unidos realizado no ano de 2001 (of Standards and Technology, 2001), que tinha como proposta escolher um novo algoritmo de criptografia que tornaria-se o novo algoritmo padrão do governo norte-americano para proteger documentos sigilosos. O algoritmo a ser substituído na ocasião era o Triple DES, que já apresentava problemas de segurança e tinha um desempenho abaixo do esperado. Vencedor do

concurso, o AES mostra-se até hoje resistente a ataques de criptoanálise, tornando-se a opção mais popular de empresas e usuários para cifrar seus dados.

Diferente do Blowfish e do Triple DES, o AES não usa a estrutura de Feistel (Feistel, 1973) como base do algoritmo. O AES usa uma rede de substituição-permutação, e o seu processo de cifragem pode ser dividido em 4 etapas:

- SubBytes: cada byte no vetor é substituído usando uma caixa de substituição (S) de 8 bits.
- ShiftRows: cada byte na linha realiza uma troca cíclica para a esquerda, sendo que cada linha opera uma troca diferente, variando o número de posições alteradas.
- MixColumns: cada coluna é multiplicada por uma função polinomial $c(x)$.
- AddRoundKey: cada byte do vetor realiza uma operação XOR ($\oplus$) com a sub-chave. Esta sub-chave é derivada da chave principal, e é alterada em cada turno;

Em cada turno do processo de cifragem do algoritmo AES, estas 4 etapas são executadas. A exceção se aplica no primeiro e último turno do algoritmo. Todas estas etapas estão ilustradas nas Figuras 2.4, 2.5, 2.6 e 2.7. O processo ocorre na seguinte sequência:

1. KeyExpansion: as sub-chaves que serão usadas em cada turno pela etapa AddRoundKey são criadas.
2. Turno inicial
    - AddRoundKey
3. Turnos
    - SubBytes
    - ShiftRows
    - MixColumns
    - AddRoundKey
4. Turno final
    - SubBytes
    - ShiftRows
    - AddRoundKey

O diagrama de fluxo de todo o processo pode ser consultado na Figura 2.3.

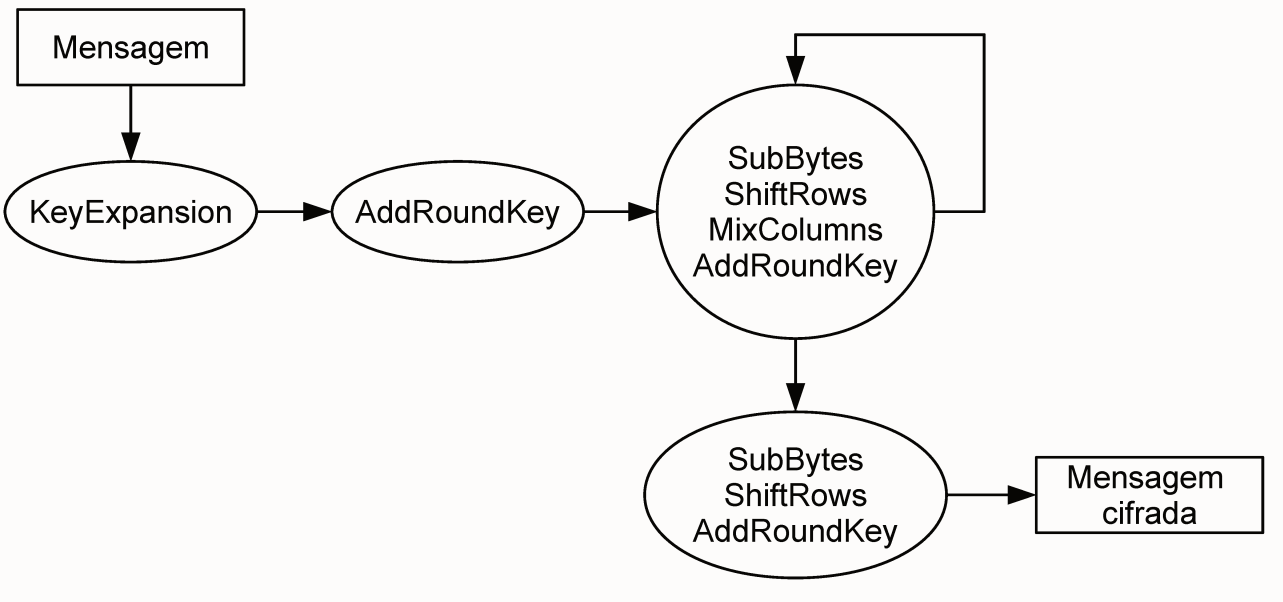

Fig. 2.3: Diagrama de fluxo do processo de criptografia do algoritmo AES.

2.3.2 Triple DES

O Triple DES (Voorhees and Buckland, 2004), ou 3DES, é um algoritmo criptográfico derivado do algoritmo DES (National Institute of Standards and Technology, 1999), desenvolvido pela IBM. Foi concebido para substituir o próprio DES devido aos problemas de segurança e limitações do mesmo.

O Triple DES cifra um bloco de 64 bits de tamanho por vez, assim como o algoritmo de criptografia Blowfish, porém diferente do algoritmo AES, pois este último que usa blocos de 128 bits.

Uma vez que é derivado do DES ele é baseado na estrutura de criptografia de Feistel (Feistel, 1973). A Figura 2.8 ilustra o funcionamento da estrutura do cifrador de Feistel. Inicialmente o cifrador divide a mensagem em 2 blocos de bits. Como o Triple DES trabalha cifrando 64 bits de dados por vez, uma mensagem de 64 bits é dividida em 2 blocos de 32 bits para que a operação de cifragem possa ocorrer.

Em cada iteração do algoritmo, estes 2 blocos de 32 bits sofrem algumas operações. Acompanhando a Figura 2.8, a saída da esquerda é apenas uma cópia da saída da direita. A saída da direita é formado pelo resultado do OR exclusivo ($\oplus$) bit a bit aplicado à entrada da esquerda e a uma função da entrada da direita com a sub-chave deste estágio, K_i. Estas sub-chaves K_0,...,K_i,..., K_n são geradas antes de cifrar a mensagem, e é gerada uma sub-chave diferente para cada iteração do algoritmo, que no DES consiste em 16 iterações idênticas que utilizam sub-chaves diferentes, e 3 distintas (a primeira, a penúltima e a última iteração, que não usam sub-chaves).

O tamanho da chave padrão para o algoritmo Triple DES é de 168 bits. Na prática esta chave é dividida em três chaves, cada uma de 56 bits. O Triple DES recebe este nome pois para cada bloco

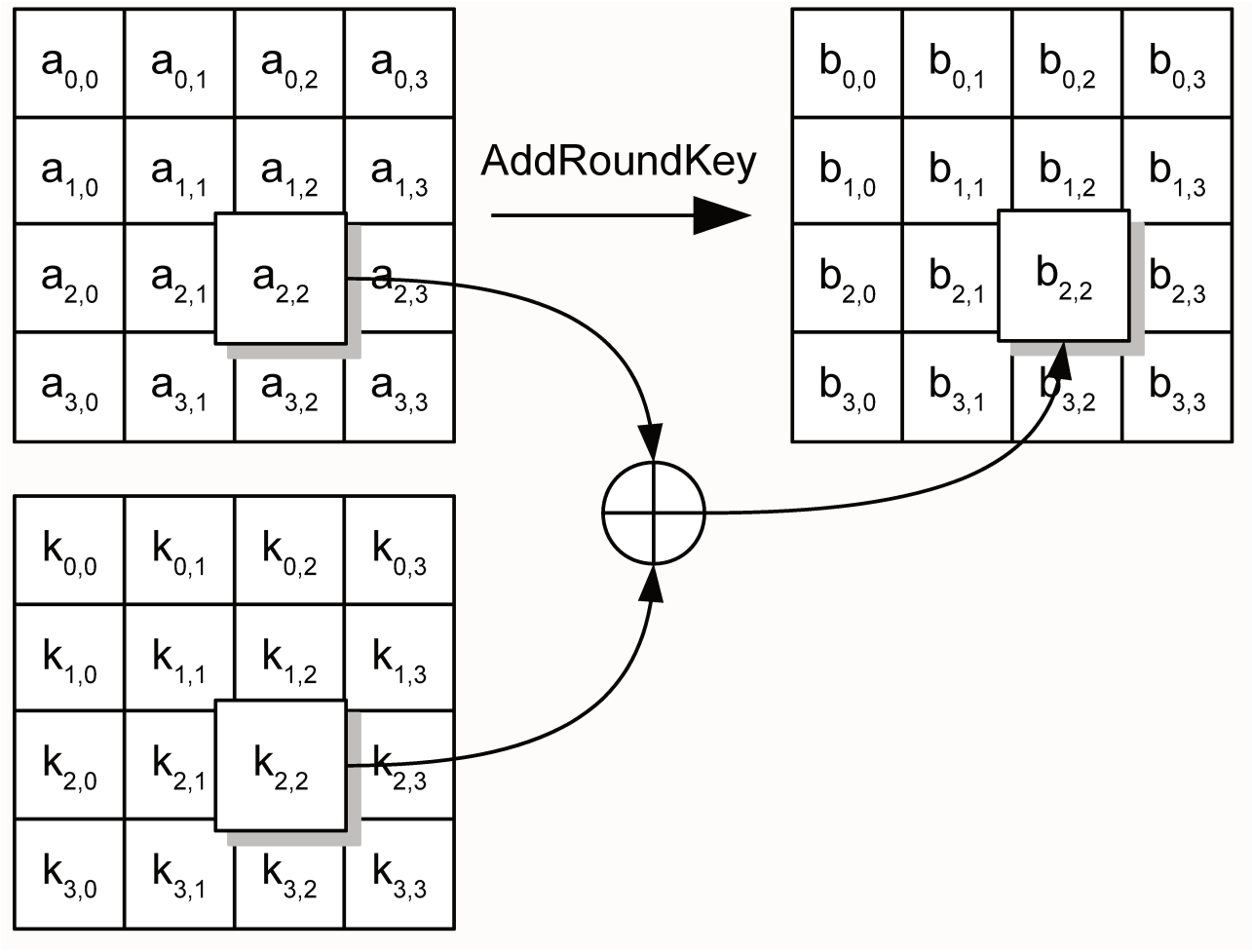

Fig. 2.4: Etapa AddRoundKey do algoritmo de criptografia AES.

de dados ele aplica o algoritmos DES três vezes, usando em cada etapa uma destas chaves de 56 bits. Ele foi criado desta maneira para preservar a compatibilidade com equipamentos e *softwares* antigos, criados para operarem com o algoritmo DES.

Vale ressaltar que estes 168 bits é o tamanho efetivo da chave do Triple DES, pois na realidade sua chave tem no total 192 bits. A explicação para isto é que o DES tem uma chave de 64 bits na verdade, porém 8 destes bits são usados para paridade e não são usados na cifragem de dados. Assim, o DES tem apenas 56 bits de tamanho de chave efetivo. Como o Triples DES aplica o DES três vezes, temos 192 bits de tamanho total da chave, nos quais 24 bits são ignorados no processo de cifragem, restando uma chave de 168 bits para cifrar os dados.

A cifragem no Triples DES ocorre da seguinte maneira:

1. Na primeira etapa o algoritmo usa a primeira chave K_1 de 56 bits e cifra a mensagem M;

2. Na segunda etapa usa a segunda chave K_2 de 56 bits e decifra os blocos. Veja que ao invés

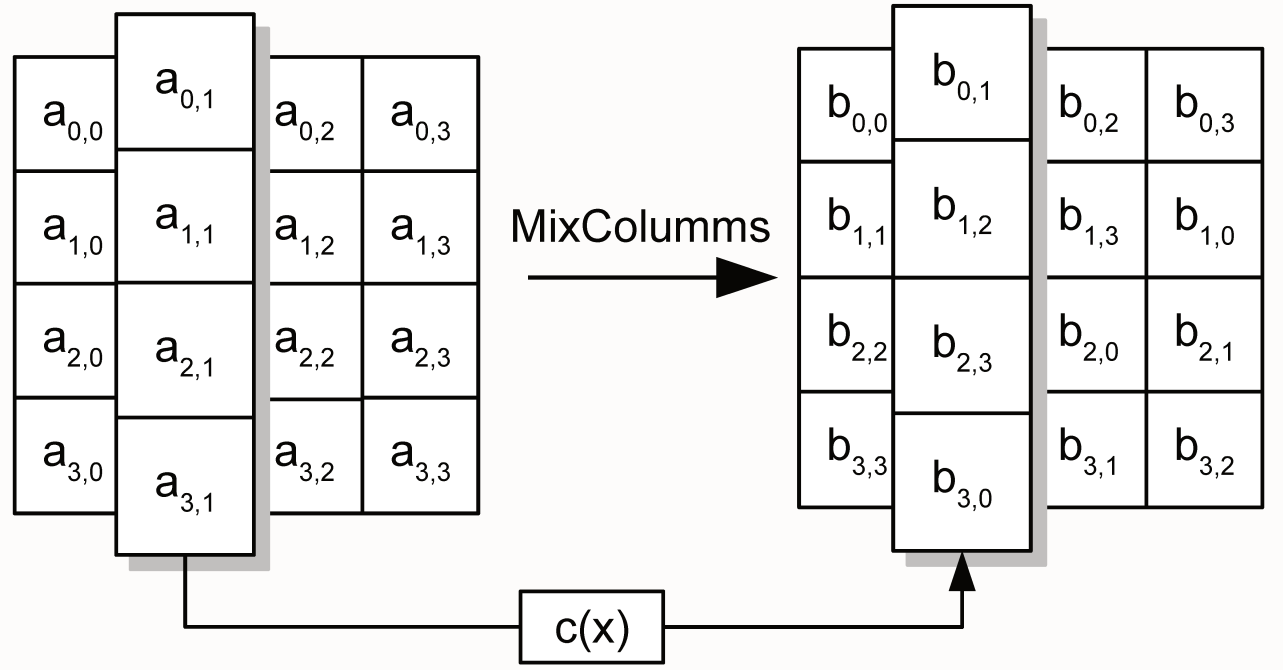

Fig. 2.5: Etapa MixColumns do algoritmo de criptografia AES.

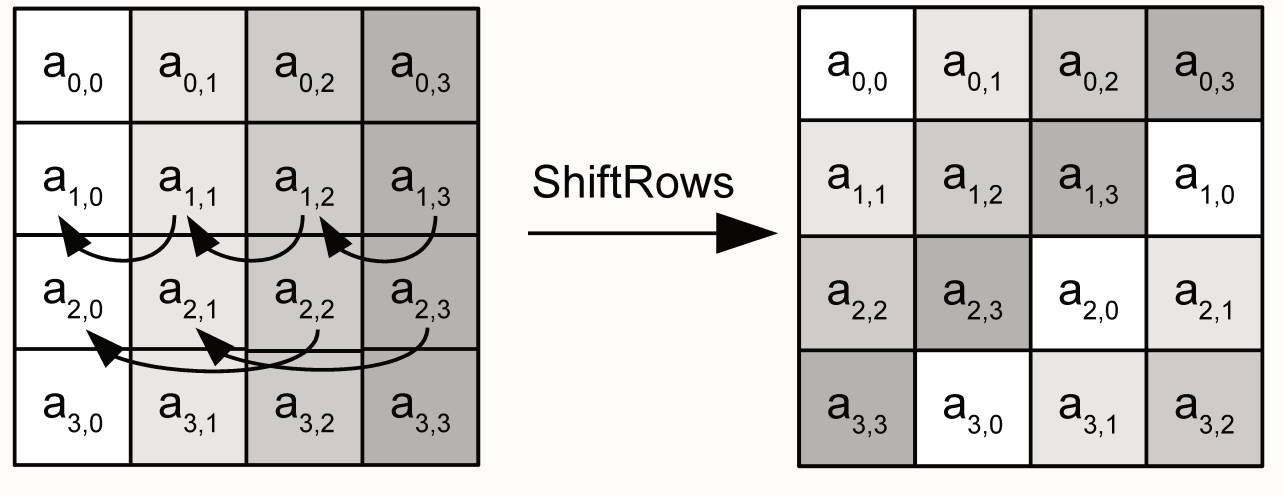

Fig. 2.6: Etapa ShiftRows do algoritmo de criptografia AES.

de cifrar com uma chave diferente, ele decifra com uma chave diferente. O efeito real é uma nova cifragem, pois ao tentar decifrar com uma chave K_2 um bloco cifrado com uma chave K_1 teremos os dados mais uma vez "embaralhados";

3. Na terceira etapa a terceira chave K_3 é usada para cifrar novamente o resultado da etapa anterior e gerar a mensagem cifrada C;

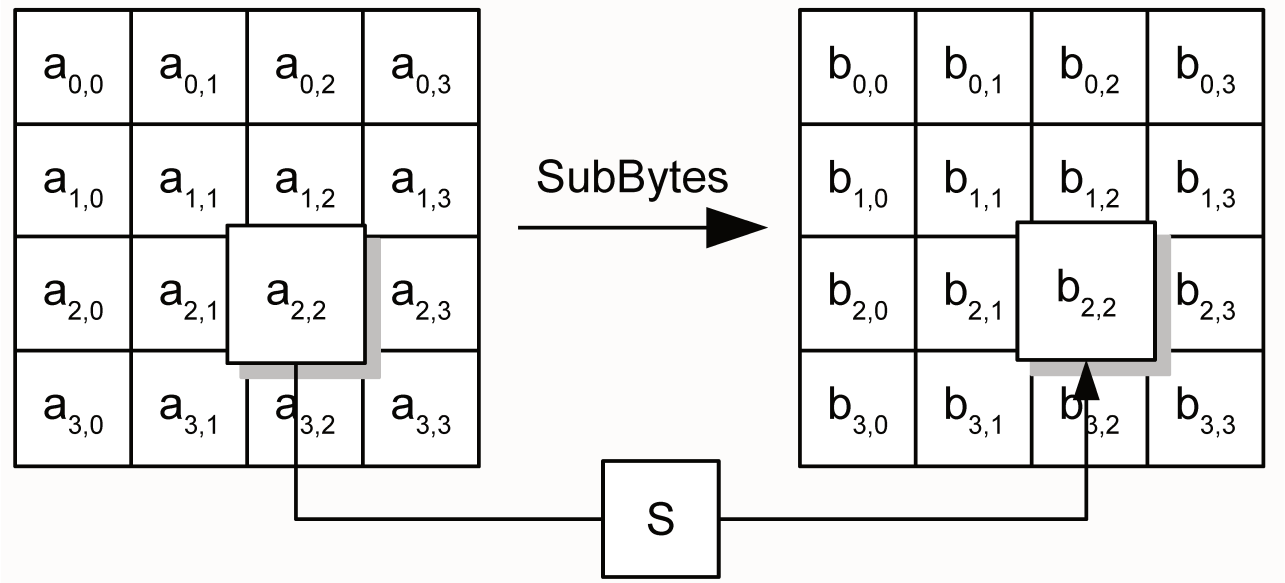

Fig. 2.7: Etapa SubBytes do algoritmo de criptografia AES.

O processo de decifragem é o processo inverso da cifragem. A Figura 2.9 mostra como ocorre a cifragem e decifragem.

A retrocompatibilidade do Triple DES com o DES pode ser alcançada quando $K_3 = K_2$ ou $K_2 = K_1$. O Triple DES também pode ser usado com uma chave de 112 bits de tamanho, aplicando-se o DES apenas 2 vezes. Basicamente é o mesmo processo descrito para o Triple DES no momento de cifragem e decifragem, excluindo a última e primeira etapa do processo uma vez que com uma chave de 112 bits temos disponíveis apenas 2 chaves de 56 bits.

Embora o Triple DES seja um algoritmo antigo, ele ainda oferece segurança suficiente pois todos os ataques descritos contra este algoritmo são impraticáveis, não podendo serem aplicados (Stallings, 2005). Uma das provas desta segurança é que ele continua ser usado atualmente em certificados bancários para proteger as informações dos clientes, e também está relacionado como algoritmo aprovado na versão mais nova do padrão de especificações dos cartões com circuito integrado (cartões EMV), usado em sistemas de pagamento (EMVCo, 2008). O NIST também afirma que o Triples DES é um algoritmo de criptografia que pode ser usado com segurança até 2030 (National Institute of Standards and Technology, 2007)

2.3.3 Blowfish

O algoritmo de criptografia Blowfish (Schneier, 1994) foi criado pelo especialista em segurança Bruce Schneier, que tinha intuito de criar um algoritmo de criptografia rápido e uma alternativa de algoritmo criptográfico livre de qualquer patente. É consideravalmente mais rápido que o DES - que

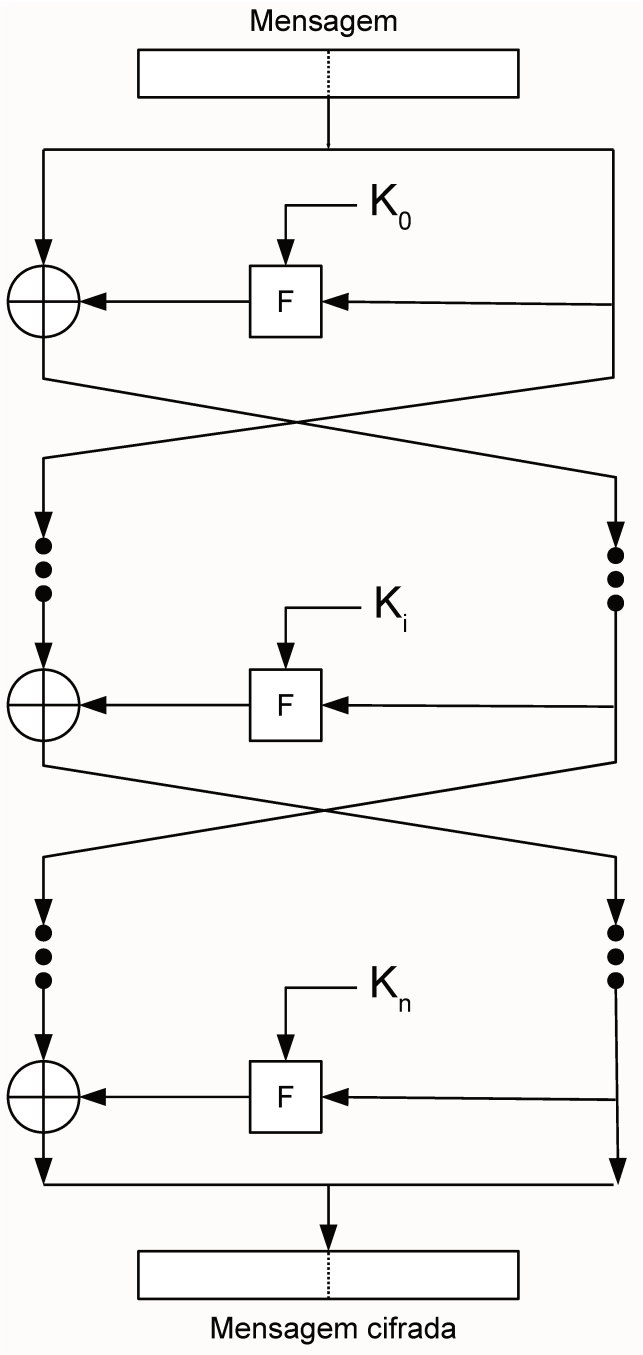

Fig. 2.8: Cifrador de Feistel.

na época era o algoritmo mais popular - e, assim como ele, cifra blocos com tamanho de 64 bits.

É um algoritmo considerado seguro pois, assim como ocorre com o AES, não há técnicas de criptoanálise eficazes contra ele atualmente (RSA Security, 2009a). Inclusive, é o algoritmo padrão do programa OpenVPN, que utilizaremos em nosso trabalho para criptografar os dados. O tamanho

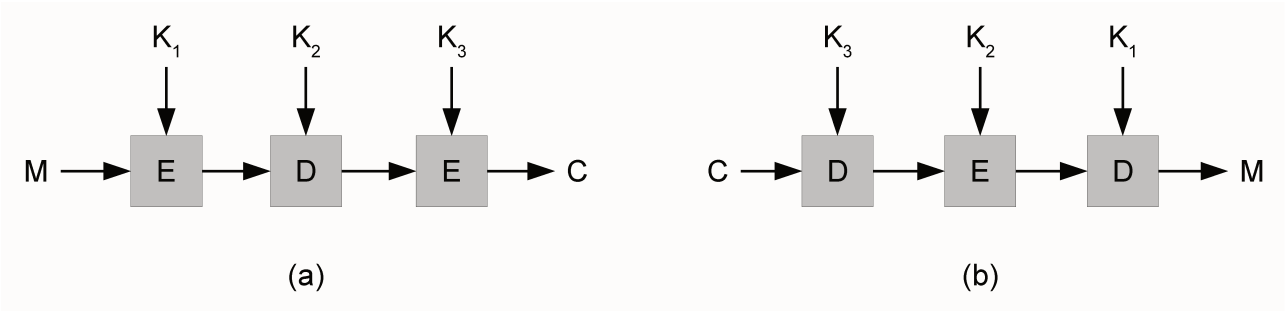

Fig. 2.9: Criptografia do Triple DES, com o DES sendo aplicado por 3 vezes. (a) Cifragem. (b) Decifragem.

de chave suportado pelo Blowfish corresponde a todos múltiplos de 8 no intervalo de 32 a 448 bits, mostrando-se um algoritmo flexível neste sentido.

Assim como o Triple DES, o Blowfish é baseado na estrutura de criptografia de Feistel (Feistel, 1973). A Figura 2.8 ilustra o cifrador de Feistel, e uma explicação detalhada do seu funcionamento pode ser encontrado na Seção 2.3.2, sobre o Triple DES.

2.3.4 Modos de operação

Os algoritmos de criptografia que cifram em blocos são basicamente uma cifra de substituição monoalfabética: o bloco de texto que chegar ao cifrador vai sempre resultar no mesmo bloco de texto cifrado se a chave secreta for sempre a mesma (Tanenbaum, 2003). Esta é uma propriedade que um usuário mal-intencionado pode usar para alterar o conteúdo de alguma mensagem.

Para contornar esta característica indesejada, os algoritmos cifradores em blocos podem usar diferentes modos de operação. Alguns deles não ajudam na segurança, como é o caso do modo de operação ECB (*Eletronic Code Book*), que apenas transforma o bloco de texto simples em bloco de texto cifrado, conforme mencionado no parágrafo anterior. Já outros modos, como o CBC (*Cipher Block Chaining*), adicionam métodos que evitam esta vulnerabilidade. O CBC é o modo de operação que será utilizado em nosso trabalho, e sua forma de operação está detalhada nesta seção.

Além do CBC e ECB, podemos citar os modos de operação CFB (*Cipher Feedback Block*) e OFB (*Output Feedback Block*). Umas das idéias iniciais do trabalho era usar mais de um modo de operação além do CBC, provavelmente o CFB. O modo CFB oferece a oportunidade de um algoritmo de criptografia cifrador de blocos, como o AES, virar um cifrador de fluxo (*stream cipher*) (Luo, 2008), característica esta encontrada nativamente em algoritmos cifradores de fluxo, como o RC4 (Kaukonen and Thayer, 1999). Embora alguns autores afirmem que um algoritmo cifrador de bloco em modo CFB normalmente não seja tão rápido quanto um algoritmo cifrador de fluxo (RSA Security, 2009b), gostaríamos de tentar visualizar esta diferença na prática. Entretanto, esbarramos em dificuldades

técnicas que estão descritas na Seção 4.2.3, impedindo a inclusão de testes desta natureza em nossos resultados.

Modo CBC

Em nosso trabalho, iremos usar apenas o modo de operação CBC (*Cipher Block Chaining* - Encadeamento de blocos de cifras) com os algoritmos de criptografia. Um exemplo de sua operação pode ser vista na Figura 2.10. No modo CBC, cada bloco de texto simples recebe uma operção lógica XOR (OR exclusivo) do bloco do texto cifrado anterior e então é cifrado. Um vetor de inicialização VI é usado como semente para dar início ao processo, e este bloco é transmitido (em texto simples) junto com o texto cifrado (Tanenbaum, 2003).

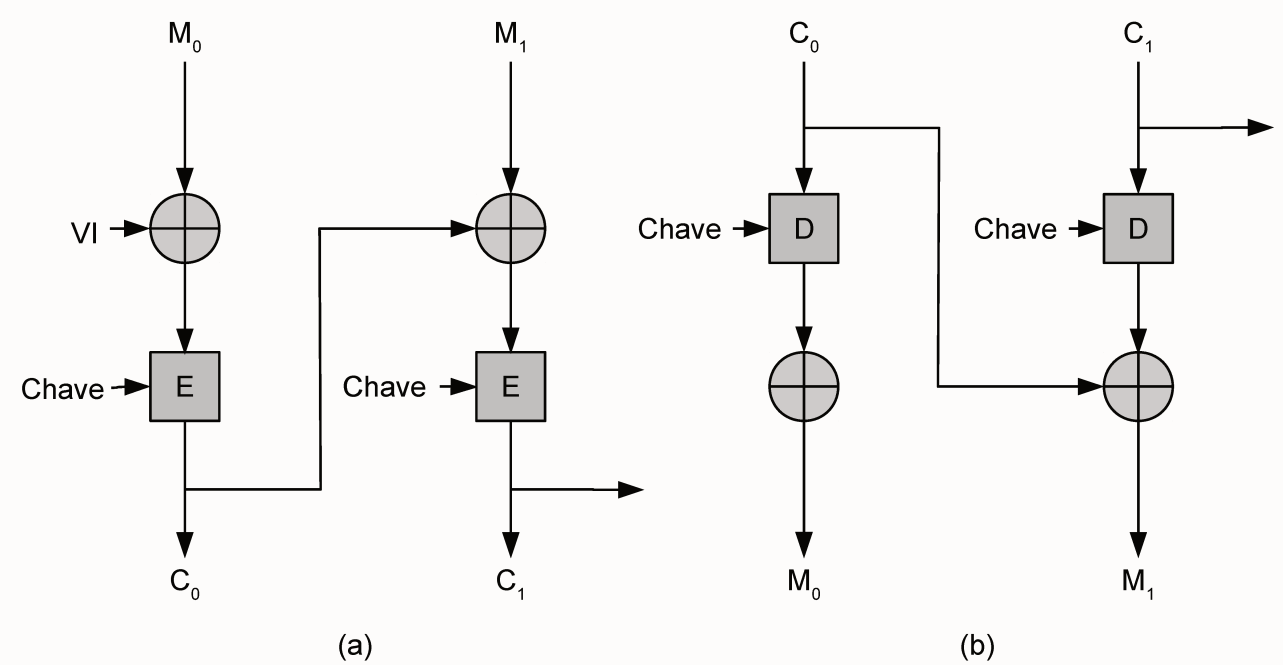

Fig. 2.10: Modo de operação CBC. (a) Cifragem. (b) Decifragem

Para o processo de cifragem, o processo começa calculando C_0 = E (M_0 XOR VI). Depois é calculado C_1 = E(M_1 XOR C_0) e assim por diante.

No processo de decifragem temos o inverso. Para obter M_0 em texto simples (decifrado) fazemos M_0 = VI XOR D(C_0), depois M_1 = C_0 XOR D(C_1) e assim por diante.

A principal vantagem da utilização do modo CBC é que o mesmo bloco de texto simples nunca resultará no mesmo bloco de texto cifrado, dificultando a criptoanálise (Tanenbaum, 2003).

2.4 Transport Layer Security (TLS)

O TLS (*Transport Layer Security*) (Dierks and Rescorla, 2008) é um protocolo de segurança criado pelo IETF (*The Internet Engineering Task Force*), derivado do protocolo SSL (*Secure Socket Layer*) (Freier et al., 1996) versão 3, desenvolvido pela Netscape no ano de 1994. Atualmente o TLS encontra-se na sua versão 1.2.

Os principais objetivos do TLS são:

- Segurança criptográfica: o TLS deve ser usado para estabelecer uma comunicação segura entre duas partes;
- Interoperabilidade: programadores independentes devem ser capazes de desenvolver aplicações utilizando TLS que podem trocar parâmetros criptográficos sem conhecimento do código do outro.
- Extensibilidade: o TLS procura fornecer uma estrutura na qual novos métodos de chave pública e criptografia possam ser incorporados conforme o necessário. Isto cumpre com 2 outros objetivos: evitar a necessidade de criar um novo protocolo (com o risco de criar vulnerabilidades) e evitar a necessidade de implementar uma nova biblioteca de segurança.
- Eficiência relativa: operações criptográficas tendem a usar intensivamente o processador, principalmente em operações com chave pública. Por esta razão, o procolo TLS incorpora uma sessão opcional de cache para reduzir o número de conexões que necessitam serem estabelecidas desde o seu início. Adicionalmente, a atividade na rede também é reduzida.

O TLS atua entre a camada de aplicação e a camada de transporte. A Figura 2.11 ilustra a posição do TLS nas camada TCP/IP. Na figura, podemos observar que o TLS é composto por 2 camadas: *Handshake Protocol* e *Record Protocol*. O *Handshake Protocol* é responsável pela troca de parâmetros envolvidos para estabelecer a comunicação segura e o *Record Protocol* é o local onde os dados são cifrados e decifrados.

O *Handshake Protocol* é composto por outros 3 protocolos:

- *Handshake*: realiza a troca de parâmetros de segurança, tais como versão do protocolo, algoritmo criptográfico, autenticação, e técnicas de cifragem com chave pública para gerar as chaves secretas;
- *Change Cipher Spec*: sinalização de transições nas estratégias de cifragem. Em outras palavras, armazena as informações correspondentes aos métodos que estão sendo utilizados no momento para realizar a cifragem e decifragem dos dados;

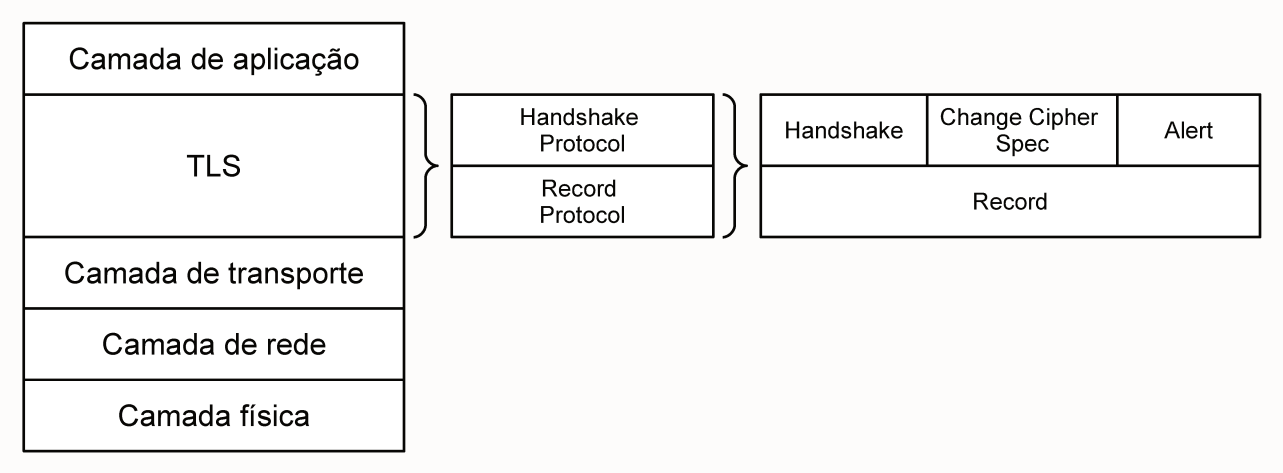

Fig. 2.11: Representação do protocolo TLS nas camadas TCP/IP.

- *Alert*: responsável por transmitir as mensagens de alerta com relação as informações que estão sendo transmitidas na comunicação;

Em nosso trabalho o protocolo TLS é usado pelo OpenVPN, sendo este responsável por criar uma rede privada virtual cifrada.

2.5 Qualidade de voz

Quando enviamos dados de voz através de redes IP, existem diversos fatores que afetam a qualidade da voz percebida pelo usuário. Entre os fatores que podem degradar a qualidade de uma comunicação VoIP, podemos citar: perda de pacotes, atraso, variação do atraso[3], compressão da voz, transdutores (microfones e alto-falantes), algoritmos de cancelamento de eco e detecção de atividade de voz nas extremidades da comunicação (remetente e destinatário) (Barbieri et al., 2002).

Nas próximas seções vamos demonstrar como a qualidade de voz pode ser medida por meio dos métodos existentes e mais bem aceitos tanto no meio comercial quanto no acadêmico.

2.5.1 E-Model

O E-Model (ou Modelo E) é uma métrica do ITU-T G.107 (Bergstra and Middelburg, 2003) para medir a qualidade de voz percebida pelo usuário em uma transmissão de telefone. O E-Model leva em

[3]A variação do atraso é conhecida também como *jitter*. Este termo, contudo, pode causar confusão pois nem sempre ele é usado como sinônimo de variação no atraso do pacote (Demichelis and Chimento, 2002). Neste trabalho evitaremos o termo "*jitter*" sempre que possível

conta um grande número de deficiências provenientes de chamadas telefônicas, em particular a deficiência ocasionada por dispositivos de codificação e atraso de uma via[4], como também de problemas clássicos de telefonia, tais como perdas, ruídos e eco.

Este modelo foi criado com base em vários testes subjetivos com uma grande gama de parâmetros de transmissão. A saída primária é um valor de avaliação da qualidade escalar conhecido como *Transmission Rating Factor R*, também conhecido por *R Factor* ou "Fator R". O valor de "R" pode ser transformado em outras medidas de qualidade, como no MOS (detalhes na Seção 2.5.2).

O algortimo de cálculo do E-model leva em conta a combinação dos efeitos destes parâmetros na conexão. O cálculo do R pode ser visto na Fórmula 2.1:

$$R = R_0 - I_s - I_d - I_{e,eff} + A \tag{2.1}$$

Onde (ITU-T, 2008b):

- R_0: fator que expressa a taxa básica de sinal-ruído;
- I_s: fator que representa todas as perdas que ocorrem simultâneamente com o sinal de voz;
- I_d: fator somatório de todos as perdas devidas ao atraso e eco. Na prática trata-se apenas do atraso envolvido na transmissão;
- $I_{e,eff}$: fator de perdas relacionadas a codificador da voz. Estes valores são fixos de acordo com o *codec* usado, e podem ser consultados na recomendação ITU-T G.113 Apêndice I (ITU-T, 2001);
- A: é o fator de vantagem. Algumas comunicações, como as realizadas via satélite, tem muito mais perda na qualidade do sinal da voz devido a problemas intrínsecos da tecnologia. Então leva-se em conta que o usuário poder tolerar alguma queda na qualidade da comunicação nestes casos específicos, pois sua percepção da má qualidade é atenuada por entender a dificuldade técnica envolvida em condições mais adversas da comunicação. Perceba que, diferente dos outros parâmetros, o fator A soma ao R_0 ao invés de substraí-lo;

Embora tenhamos estes parâmetros, o IxChariot, responsável por fazer a análise da qualidade de voz em nossos testes, usa uma equação modificada para realizar esta medição da qualidade da voz, que pode ser vista na Fórmula 2.2. Para simular o tráfego VoIP, ele gera fluxos de pacotes RTP (*Real-time Transport Protocol*), e de tempo em tempo ele coleta as informações de atraso, variação do atraso, perda de pacotes e número de pacotes consecutivos perdidos para determinar a percepção de qualidade da comunicação VoIP do ponto de vista do usuário (Ixia, 2004).

[4] *One-way delay*

Fator R	MOS	Satisfação do usuário
93	4.2	Máximo obtido pelo *codec* G.711
90-100	4.34-5.0	Muito satisfeito
80-90	4.03-4.34	Satisfeito
70-80	3.60-4.03	Alguns usuários insatisfeitos
60-70	3.10-3.60	Muitos usuários insatisfeitos
50-60	2.58-3.10	Quase todos os usuários insatisfeitos
0-50	1.0-2.58	Não recomendado

Tab. 2.4: Relação entre o MOS, Fator R e nível de satisfação do usuário

$$R = R_0 - I_d - I_e \tag{2.2}$$

A explicação dos parâmetros usados ocorre da mesma forma que a relacionada na Fórmula 2.1, mudando apenas o nome da variável correspondente ao fator de perdas, que de $I_{e,eff}$ passa para I_e.

2.5.2 Mean Opinion Score (MOS)

O MOS (*Mean Opinion Score*) é um padrão amplamente aceito para avaliar a percepção de qualidade de voz. Os testes de MOS são apresentados no ITU-T P.800 (ITU-T, 1996), no qual os usuários avaliam a qualidade de uma conversa dando notas em uma escala de 1 (ruim) até 5 (excelente). Contudo, estes testes subjetivos de MOS consumiam muito tempo, eram caros e não permitiam medições em tempo real (Carvalho et al., 2005).

Passado algum tempo, alguns métodos foram desenvolvidos para medir o MOS objetivamente. O ITU-T G.107 (Bergstra and Middelburg, 2003) define, conforme explicado na seção 2.5.1, o E-Model, e neste documento é descrito uma forma de converter o resultado do E-Model (o Fator R) para um valor na escala do MOS. A Fórmula 2.3 demonstra como este cálculo pode ser realizado.

$$\begin{aligned} &\text{Para } R < 0: && MOS = 1 \\ &\text{Para } 0 < R < 10: && MOS = 1 + 0.035R + R(R-60)(100-R)7\cdot 10^{-6} \\ &\text{Para } R > 100: && MOS = 5 \end{aligned} \tag{2.3}$$

A relação entre o valor de MOS, o Fator R e a satisfação do usuário com relação a qualidade da voz pode ser vista na Tabela 2.4 (Bergstra and Middelburg, 2003) (VoIP Troubleshooter LLC, 2009). Esta tabela também traz os valores esperados quando o *codec* G.711 é utilizado para codificar a voz.

O IxChariot fornece uma avaliação máxima de MOS no valor de 4.38 ao usar o *codec* G.711. De fato, este valor de MOS máximo para o codec G.711 varia na literatura, uma vez que os desenvolvedores ou mesmo pesquisadores decidem por usar este ou aquele parâmetro da fórmula do E-Model,

produzindo geralmente resultados diferente, porém próximos. Há também a sensibilidade envolvida ao analisar a qualidade da voz dependendo do método utilizado, pois até mesmo o gênero da voz (masculina ou feminina) podem gerar resultados diferentes de MOS (Dai, 2000). Como exemplo destas diferenças nas medições de qualidade, no PSQM (*Perceptual Speech Quality Measure*) é dito que o valor típico de MOS para o *codec* G.711 é 4.5 (Dai, 2000) enquanto Hersent et al. (Hersent et al., 2002) afirma que este valor é 4.2.

De qualquer forma, independente se o valor é 4.5 ou 4.2, qualquer valor de MOS acima de 4 pode ser considerado um valor ótimo, pois o usuário nestas condições encontra-se satisfeito com a qualidade da comunicação. A preocupação com a qualidade começa quando os valores de MOS ficam abaixo de 3.6, que embora não seja ainda um impeditivo para a comunicação, já poderá incomodar alguns usuários.

2.6 Pesquisas recentes relacionadas

Nesta seção foram relacionados os trabalhos recentes que aproximam da nossa proposta, e que foram usados de alguma forma para consulta, criação dos cenários de rede, escolha das ferramentas usadas para os testes, etc.

Com respeito a trabalhos em qualidade de voz, Barbosa (Barbosa et al., 2006) realizou experimentos similares aos do nosso trabalho com relação ao uso de diferentes cenários de rede e avaliação da qualidade das chamadas. Ele usou alguns parâmetros para criação do ambiente de rede de modo a constatar a influência de cada um deles nos resultados. Neste ponto nosso trabalho difere, pois nossos parâmetros eram baseados em medições de cenários reais. Barbosa também não fez o controle explícito da variação do atraso, fato que ele salientou em seu trabalho. Outra diferença é o uso por ele da ferramenta NIST.net (Carson and Santay, 2003) para emular os problemas na rede e largura de banda enquanto nós utilizamos o *Traffic Control* e NetEm para estas tarefas.

Vozňák (Vozňák, 2008) avaliou o impacto da criptografia na qualidade da comunicação VoIP, com um conjunto de ferramentas similar ao do nosso trabalho, usando OpenVPN e o IxChariot. Contudo, seu foco foi nas alterações de consumo de banda da rede quando o *codec* e o algoritmo de criptografia eram alterados. Nappa (Nappa, 2007) fez algo similar, porém sem dar ênfase a largura de banda consumida nos resultados finais. Nappa também usou OpenVPN e IxChariot, com o programa iPerf (NLANR/DAST, 2010) para gerar tráfego na rede, característica esta necessária em alguns dos seus testes.

Markopoulou (Markopoulou et al., 2003) realizou testes para verificar a qualidade de comunicações VoIP em *backbones* de Internet. Nestes testes, ele verificava as anomalias que afetavam diferentes provedores dos Estados Unidos, sendo possível observar o atraso, variação do atraso, taxa de

perda de pacotes e congestionamentos que ocorriam. De seu trabalho aproveitamos estes dados para criar alguns de nossos cenários com parâmetros próximos aos reais.

Snyder (Snyder, 2006) propôs alguns cenários baseados em testes de medição. Estes cenários apresentavam diferentes problemas e limitações, levantados com base em medições das condições de redes presentes em hotéis, pontos de acesso Wi-Fi, e outras localizações. Através destes cenários, ele realizou testes de desempenho com diversos equipamentos que criavam VPNs e verificou o comportamento de cada um deles em cada situação imposta.

Capítulo 3

Programas utilizados

Para a realização da proposta do trabalho, foram usadas diversas ferramentas e programas em conjunto. Cada escolha de programa foi baseada em critérios técnicos, documentação disponível e também de experiência prévia de uso, já que a familiaridade com a mesma auxilia no bom uso de seus recursos e na resolução de possíveis problemas. Assim sendo, nesta seção apresentaremos uma breve descrição do VirtualBox, OpenVPN, IxChariot e do Traffic Control.

3.1 VirtualBox

O VirtualBox (VirtualBox, 2008) é um programa virtualizador multiplataforma para máquinas x86[1] criado pela Sun. Uma máquina virtual compartilha os recursos do computador através do sistema operacional (hospedeiro), executando um outro sistema operacional através da máquina virtual (convidado). Este compartilhamento é permitido e gerenciado pelo VirtualBox, que pode até mesmo executar várias máquinas virtuais simultaneamente, fazendo com que elas possam não somente se comunicar entre si, mas também com o hospedeiro.

Como o VirtualBox abstrai o sistema operacional local para a máquina virtual, a máquina virtual pode conter qualquer tipo de sistema operacional suportado. Ou seja, é possível instalar o VirtualBox no Windows XP Professional e executar uma máquina virtual contendo qualquer versão atual do Linux, e trabalhar com os 2 sistemas operacionais ao mesmo tempo, como se a máquina virtual com o Linux fosse um aplicativo do Windows.

O uso do VirtualBox em nosso trabalho facilita na criação e gerenciamento da rede para os testes propostos. Sem ele, precisaríamos ter fisicamente um outro computador disponível apenas para criar esta rede e executar os testes pretendidos.

[1] Conjunto de instruções da arquitetura x86, baseado no processador Intel 8086 (Cornea et al., 2002)

Para realizar a simulação, foi criada uma máquina virtual de 3 Gigabytes de tamanho com 512 Megabytes de memória RAM disponíveis, com o sistema operacional Ubuntu Linux 8.04 instalado. A máquina virtual será responsável por conter o servidor da rede OpenVPN.

A comunicação entre a máquina virtual e a máquina hospedeira foi realizada por meio de uma interface virtual criada pelo próprio VirtualBox no computador hospedeiro. Aos olhos do sistema convidado, ele "acredita" estar conectado fisicamente ao hospedeiro por meio de um cabo de rede (Sun Microsystems, 2009) e usa esta interface para se comunicar com o mesmo.

Porém, a comunicação não ocorrerá diretamente através desta interface virtual criada pelo VirtualBox. Como desejamos criar uma VPN usando o OpenVPN, é preciso usar uma interface virtual criada pelo mesmo, que é capaz de cifrar e decifrar os dados transmitidos pela comunicação VoIP.

A opção pela máquina virtual, ao invés de 2 computadores físicos, foi uma escolha técnica e prática. Técnica porque temos o controle total da rede, sem necessidade de cabeamento que não só é vulnerável a interferências externas como também poderia causar um acréscimo, mesmo que ínfimo, de atraso. E escolha prática porque podemos gerenciar todos os processos envolvidos no mesmo computador.

3.2 OpenVPN

O OpenVPN (OpenVPN, 2007) é um programa que tem a capacidade de criar uma VPN (*Virtual Private Network* - Rede Privada Virtual) cifrada. A vantagem de uma VPN é auxiliar na divisão lógica da rede, possibilitando que um grupo específico de pessoas possa se comunicar de forma segura. Esta característica é muito apreciada no meio comercial, pois uma pessoa de qualquer lugar com acesso à Internet pode acessar a VPN de sua empresa. Outro exemplo é a possibilidade de unir transparentemente a comunicação VoIP entre diferentes escritórios de uma empresa, seja onde eles estiverem.

Quanto a confidencialidade, o OpenVPN suporta uma vasta gama de algoritmos de criptografia, uma vez que o mesmo usa para esta tarefa a biblioteca de outro programa, o OpenSSL (OpenSSL, 2007). O OpenSSL é uma implementação de código aberto do protocolo SSL (*Secure Socket Layer*) e TLS (*Transport Layer Security*), suportando diversos algoritmos simétricos, assimétricos e de *hash*. Com estes recursos disponíveis, o OpenVPN é capaz de prover segurança a todos os dados que trafegam através da VPN gerenciada por ele, dificultando que terceiros possam interceptar as informações transmitidas.

Com o uso do OpenVPN, podem ocorrer atrasos por conta da criptografia realizada pelo mesmo, pois cada pacote precisa ser cifrado no remetente e decifrado no destinatário. Um atraso expressivo pode ser percebido pelo receptor, degradando a qualidade da comunicação. Outro problema é que ao

cifrar um pacote, este aumenta de tamanho, fazendo com que a transmissão necessite de mais banda disponível se comparada a uma transmissão não cifrada do mesmo volume de dados.

Portanto, o OpenVPN pode influenciar negativamente na transmissão, sobretudo no VoIP, o que é um efeito colateral inevitável para garantir a confidencialidade dos dados. É necessário então se antecipar a estes problemas, certificando-se se haverá banda de rede suficiente e se o atraso calculado na rede será aumentado com a introdução de criptografia a ponto de impossibilitar uma chamada VoIP com qualidade satisfatória. O nosso trabalho irá procurar verificar sob quais condições de rede a transmissão VoIP cifrada seja prejudicada no cenário proposto, e se alguns dos algoritmos, mesmo em condições desfavoráveis, terão uma performance melhor que os demais e se irão garantir a qualidade da transmissão VoIP.

A configuração do OpenVPN é simples. Um dos clientes deve funcionar como o servidor OpenVPN da comunicação. O cliente precisa apontar nas configurações do OpenVPN o endereço IP válido de localização do outro cliente que atua como servidor. O OpenVPN então cria uma interface virtual de rede e é através desta interface virtual que a comunicação ocorre com outros clientes OpenVPN. Qualquer aplicação instalada no sistema operacional corrente que faça uso da rede pode usar esta interface virtual para se comunicar com outro cliente OpenVPN de forma transparente e segura.

Optamos pelo uso do OpenVPN também pelo fato de já ter sido usado em outros trabalhos que envolviam testes com VoIP (Vozňák, 2008) (Vozňák et al., 2008).

3.3 IxChariot

O IxChariot (Ixia, 2008) é uma ferramenta comercial, disponibilizada de forma gratuita por um determinado período (*trial*), para testar a performance de uma rede e de seus componentes, auxiliando na identificação de problemas através de relatórios detalhados com informações da largura de banda utilizada, pacotes perdidos, bytes enviados e recebidos, etc. Uma parte destas informações podem ser apresentadas em forma de gráfico, o que facilita o estudo dos dados da rede levantados pelo programa. Esta ferramenta mostrou-se adequada a nossa proposta após observarmos alguns trabalhos que realizavam medições da qualidade de chamadas VoIP em diferentes situações (Vozňák et al., 2008) (Vozňák, 2008).

Em nossos testes, usamos apenas o componente de VoIP da ferramenta. A Figura 3.1 mostra a tela do programa IxChariot após o teste de uma chamada VoIP. Desta forma, é possível observar o MOS (*Mean Opinion Score*) de cada chamada, atraso, perda de pacotes, variação do atraso (*jitter*), Fator R entre outras informações pertinentes.

O programa é composto pelo *Console*, responsável por criar e definir como serão os testes, e pelos *Endpoints* que recebem as instruções do *Console* para realizar os testes. O *Console* do IxChariot

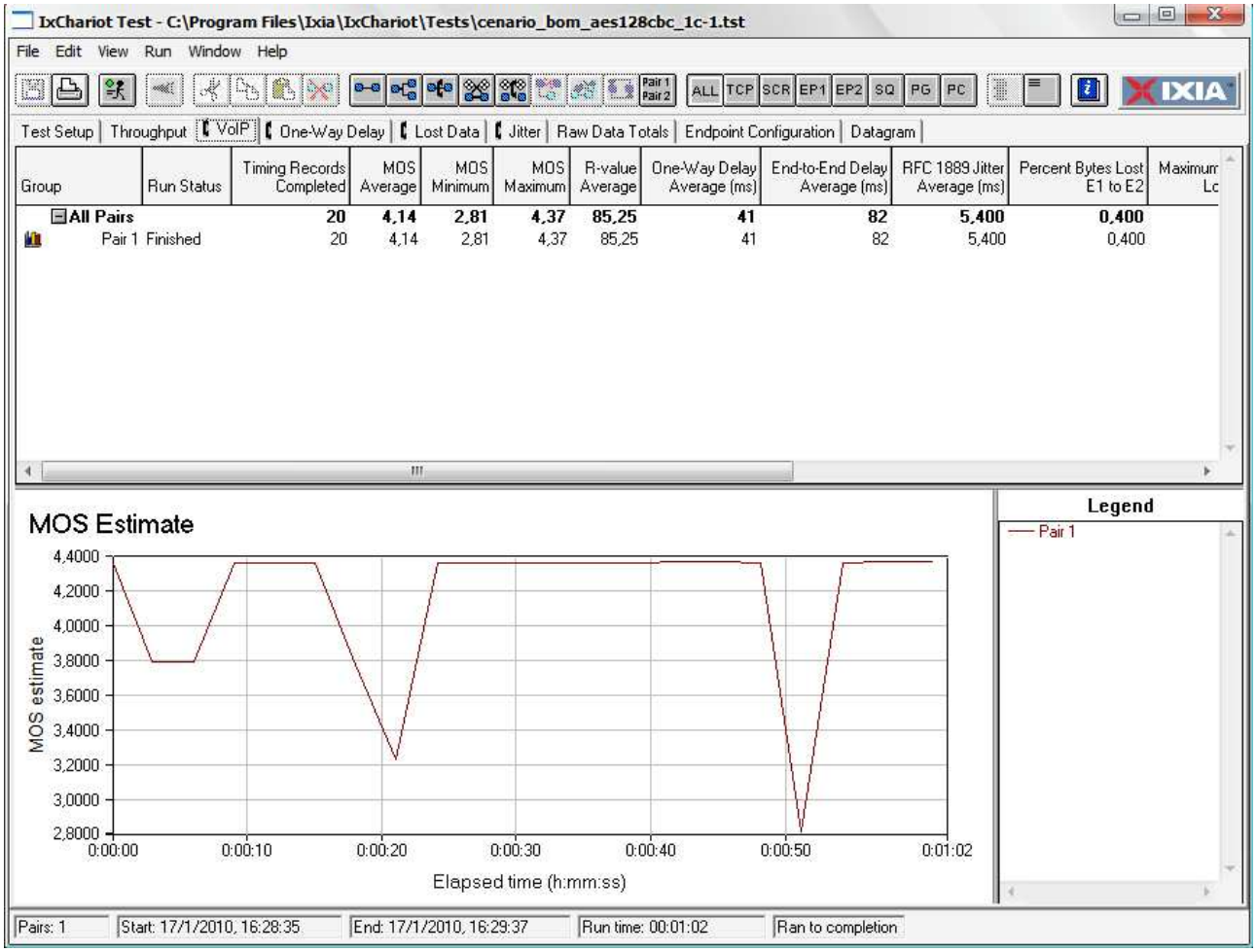

Fig. 3.1: Tela do programa IxChariot

apenas suporta o sistema operacional Windows, porém os *Endpoints* estão disponíveis para vários sistemas operacionais, dentre eles o próprio Windows, Unix, Linux, etc.

Em nosso ambiente de testes, instalamos um *Endpoint* na máquina Windows e outro em uma máquina virtual contendo o sistema operacional Ubuntu Linux. O *Console* foi instalado na mesma máquina que continha o sistema operacional Windows. A Figura 4.1 ilustra a disposição destes programas no ambiente de testes utilizado.

O uso do componente de VoIP do IxChariot ocorre da seguinte maneira. No *Console* criamos uma chamada VoIP informando os números IP das 2 máquinas que tem o *Endpoint* instalado. Ao criar esta chamada é possível redefinir vários parâmetros para a chamada VoIP, entre eles o *codec* usado, *buffer*, etc. O tempo da chamada que será realizada pode ser definida no menu *Run*, opção *Run Options*.

3.4 Traffic Control

O sistema operacional Linux oferece uma grande variedade de ferramentas para controle de tráfego. Em nosso trabalho iremos usar a ferramenta "tc", uma abreviação para *Traffic Control*, que permite ao usuário um controle granular das filas e mecanismos de enfileiramento da interface de rede. O "tc" faz parte do pacote de ferramentas de nome *iproute2*, pacote este que abriga uma suíte de utilitários de linha de comando que manipulam as estruturas do núcleo[2] do Linux para configuração da rede IP em uma estação (Brown, 2006).

Há diversas razões práticas para considerar o uso do controle de tráfego em redes. A lista a seguir mostra alguns exemplos de problemas comuns que podem ser resolvidos através do controle de tráfego (Brown, 2006):

- Limitar a largura de banda da rede para uma determinada taxa de transferência;
- Limitar a largura de banda da rede de um usuário, serviço ou cliente;
- Permitir a distribuição equitativa de banda de rede ociosa;
- Garantir que um determinado tipo de tráfego seja descartado;

Em nosso trabalho iremos usar as funções do "tc" providas pelos componentes NetEm e *Token Bucket Filter* (TBF).

[2] kernel

3.4.1 NetEm

O NetEm é um reforço às ferramentas de controle de tráfego no Linux, permitindo que o usuário possa adicionar atraso, perda de pacotes entre outros problemas nos pacotes que trafegam pela interface de rede. Ele foi concebido com base nas funções já existentes de QoS (*Quality Of Service* - Qualidade de Serviço) e DiffServ (*Differentiated Services* - Serviços Diferenciados) presentes no Linux (Hemminger, 2005).

O NetEm foi fundamental para este trabalho. É através dele que iremos criar *scripts* com linhas de comando para a emular os problemas de rede presentes nos cenários que podem ser vistos na Seção 4.3.

Vale ressaltar que no decorrer do trabalho por várias vezes usaremos o verbo "emular" ao invés de "simular" ao falar da inserção de anomalias na rede. Segundo Hemminger (Hemminger, 2005), uma simulação é um ambiente completamente virtual; uma aproximação mais sintética na qual envolve a criação de um modelo de protocolos de rede e inserção de dados sintéticos neste modelo. Simulação é muito usada no desenvolvimento de um protocolo desde o seu início pois seu funcionamento é melhor reproduzível e não é influenciado por detalhes de tempo de uma rede real. Emulação é a reprodução de características de um ambiente em um ambiente diferente, porém sem a criação de um ambiente virtual para que isto ocorra.

3.4.2 Token Bucket Filter

O TBF (*Token Bucket Filter*) é uma simples disciplina de filas (*Queueing Disciplines* - "Qdisc") que permite a passagem dos pacotes pela interface de rede que chegam respeitando uma taxa limite de transferência pré-definida, a qual não pode exceder (Brown, 2006).

Basicamente, o TBF atrasa os pacotes que chegam na interface de rede, e para que possa realizar esta tarefa ele dispõe de um *buffer*, onde controla os pacotes que devem ser transmitidos.

O processo detalhado pode ser acompanhado na Figura 3.2. A implementação consiste em um *buffer*, constantemente preenchido por alguns pedaços virtuais de informação, chamado de *tokens*, em uma taxa determinada. O mais importante parâmetro do *buffer* é o seu tamanho, que corresponde ao número de tokens que podem ser guardados (LARTC Project, 2010).

Cada *token* coleta um pacote de dados, que deve ser transmitido, da fila de pacotes e o *token* é removido do seu *buffer*. Temos então 3 possíveis cenários:

- Os dados que chegam no TBF estão em uma taxa **igual** a taxa de *tokens* que entram. Neste caso cada pacote entrante tem o seu *token* correspondente e é transmitido sem atrasos.

- Os dados que chegam no TBF estão em uma taxa **menor** que taxa de *tokens* que entram. Somente uma parte dos *tokens* são removidas da saída de cada pacote de dados, então os tokens

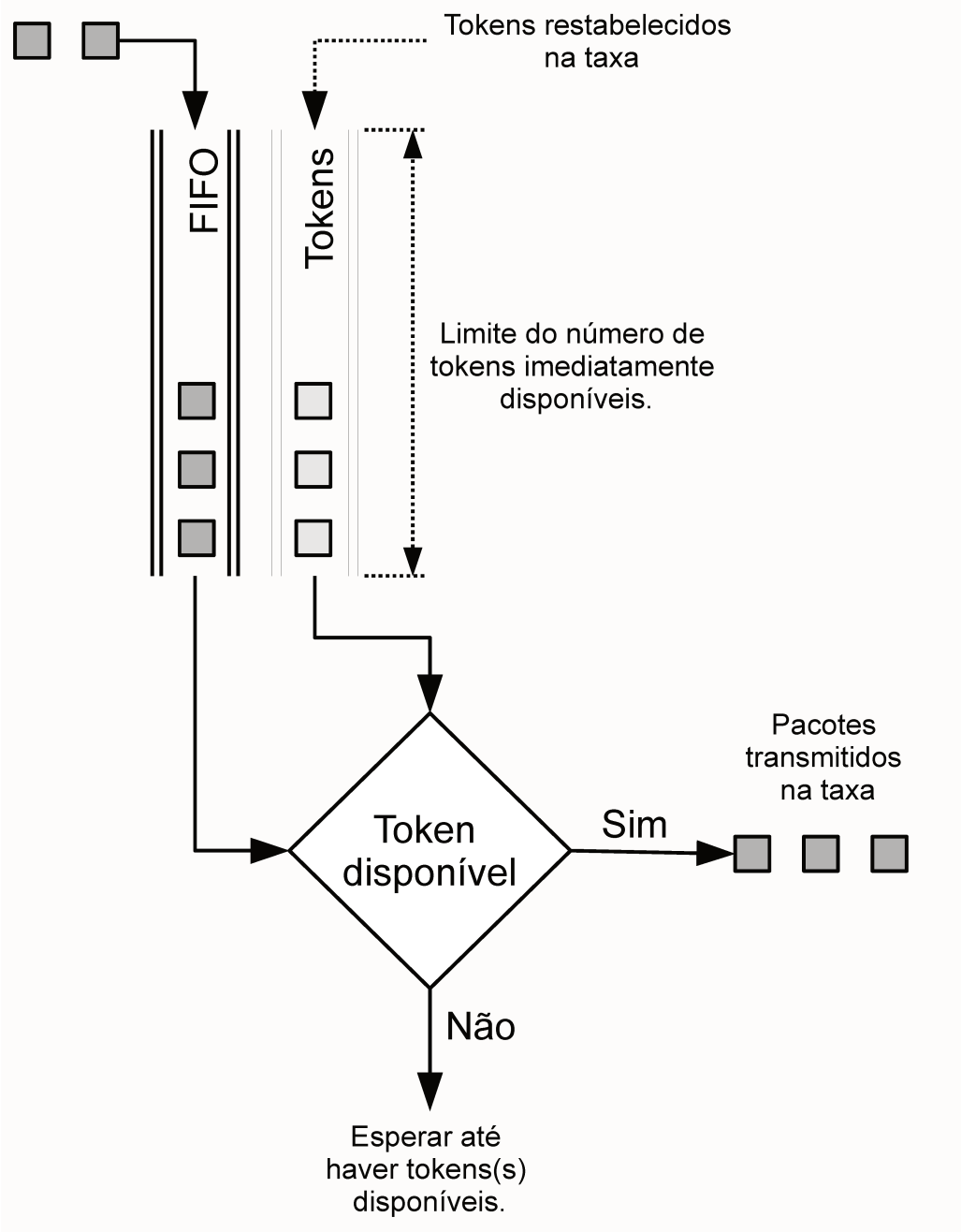

Fig. 3.2: Exemplo de algoritmo do *Token Bucket Filter*.

se acumulam até atingirem o tamanho do *buffer*.

- Os dados que chegam no TBF em uma taxa **maior** que a taxa de *tokens* que entram. Isto significa que o *buffer* logo ficará vazio. Se os dados continuarem a chegar, eles começarão a ser descartados.

O último cenário é particularmente muito importante, pois é nele que ocorrerá o controle efetivo da taxa de transferência de dados.

Ou seja, o TBF será o responsável em nosso trabalho por limitar a largura de banda da rede. Um exemplo prático de como isto pode ser feito pode ser visto na Seção 4.2.4.

Capítulo 4

Avaliação da qualidade das chamadas nos cenários de rede

Neste capítulo iremos mostrar a principal contribuição do trabalho, explicando as configurações realizadas nas ferramentas que compuseram os testes, a descrição dos cenários de rede nos quais os testes foram executados e finalmente os resultados obtidos com a devida análise das informações.

4.1 Disposição das ferramentas no ambiente

No Capítulo 3, detalhamos todos as ferramentas necessárias para o trabalho. Explicamos previamente a função e disposição delas no ambiente de testes, mas para clarificar esta questão julgamos necessária a criação de uma figura que englobasse todas estas ferramentas. A Figura 4.1 ilustra a organização usada.

De forma a resumir o entendimento da função de cada uma delas, a Tabela 4.1 exibe uma pequena e objetiva descrição de cada ferramenta.

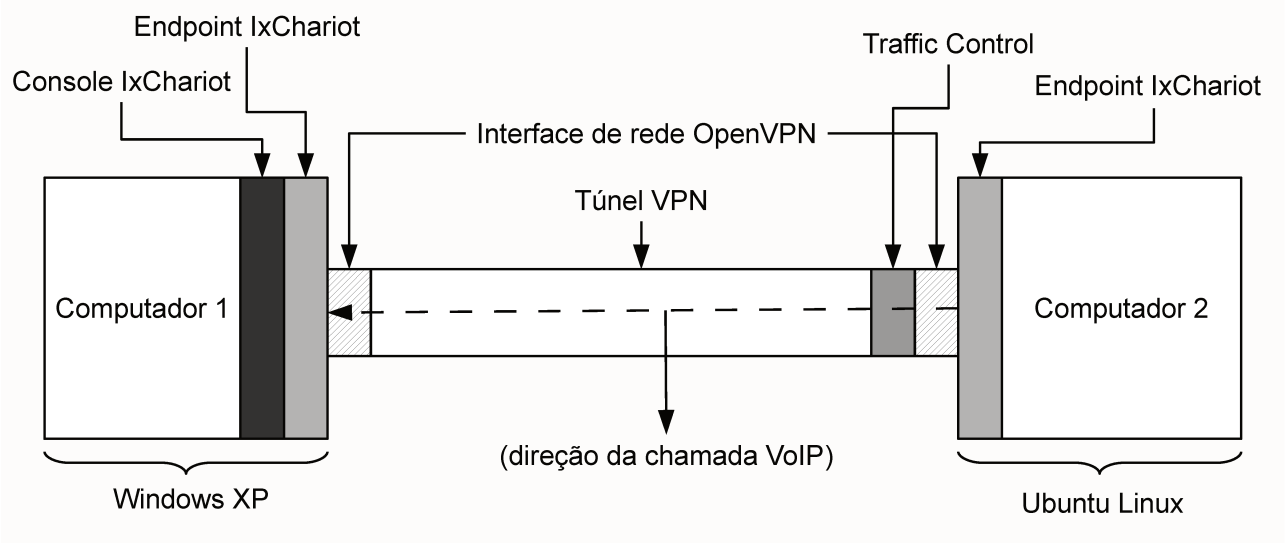

Fig. 4.1: Disposição dos programas em nossos testes entre dois computadores

Ferramenta	**Função**
VirtualBox	Criar a máquina virtual para comunicação com a máquina real. Foi um recurso usado para termos um computador virtual afim de facilitar a realização dos testes, dispensando o uso de outro computador.
OpenVPN	Criar a rede privada virtual. Responsável pela criptografia da comunicação VoIP.
IxChariot	Criar a chamada VoIP entre os *Endpoints* IxChariot e realiza a avaliação da mesma. O *Console* IxChariot coordena a chamada, passando os parâmetros necessários aos *Consoles* para que possam criar as chamadas, e ao final das mesmas analisa e exibe os dados.
Traffic Control	Responsável por introduzir as anomalias de rede e limitar a largura de banda da rede em cada cenário proposto.

Tab. 4.1: Resumo de todas as ferramentas usadas neste trabalho.

4.2 Configuração do ambiente

Para que os testes sejam executados, é natural que haja um preparo prévio do ambiente. Nesta seção apresentamos as configurações que foram necessárias para cada ferramenta utilizada.

Podemos dar destaque para a configuração do *Traffic Control*, apresentada na Seção 3.4, pois além

de darmos detalhes de como a configuração foi realizada, também disponibilizamos alguns testes que realizamos, além dos obstáculos encontrados ao usar a ferramenta e as soluções encontradas. Estes testes não fazem parte do escopo principal do trabalho. Porém, eles foram necessários para que pudéssemos tomar algumas decisões com respeito as configurações adotadas.

4.2.1 Configuração do VirtualBox

Para configurar o VirtualBox, foi criada uma máquina virtual (convidada) de 3 GB de tamanho, espaço em disco suficiente para instalar o sistema operacional, com disponibilidade de usar até 512 MB de memória RAM da máquina real (hospedeira). Nesta máquina virtual, instalamos o sistema operacional Ubuntu Linux.

Usamos a opção da interface de rede do hospedeiro[1] conectada a uma interface virtual, para que a comunicação entre a máquina real e a máquina virtual pudesse ocorrer. Esta interface é criada pelo próprio VirtualBox para atender a este tipo de opção de conexão de rede.

A configuração de rede para as 2 máquinas se comunicarem foi feita como usualmente é realizada para conectar 2 computadores em uma rede local, definindo uma máscara de rede comum e um número de IP local.

4.2.2 Configuração do IxChariot

Para configuração do IxChariot precisamos antes instalar o programa de *Endpoint* na máquina convidada, que executa em nossos testes o sistema operacional Ubuntu Linux, e na máquina hospedeira que executa o sistema operacional Windows XP.

Após a instalação, no programa de *Console* do IxChariot criamos a chamada VoIP, definindo como origem da chamada a máquina convidada e como destino a máquina hospedeira. Ou seja, a chamada ocorrerá no sentido da máquina com Linux para a com o Windows. Esta disposição foi adotada pois as regras do *Traffic Control* são aplicadas no tráfego de saída da interface de rede criada pelo OpenVPN no sistema operacional Linux.

Dentre as opções da chamada VoIP disponíveis no IxChariot, usamos as opções definidas como padrão para realizar a chamada VoIP, com exceção da opção *jitter buffer*[2], pois dependendo do cenário de rede seu tamanho é aumentado de forma a refletir as mudanças que um administrador de uma rede VoIP provavelmente faria no aplicativo VoIP usado pelos usuários devida as condições da rede. As chamadas usam o *codec* G.711 μ-law para codificação da voz devido a pouca interferência que exerce no atraso e na qualidade da chamada (detalhes na Seção 2.1.4).

[1] Conhecida também como *host-only*

[2] Opção do IxChariot que permite a configuração do tamanho do *buffer* para a variação de atraso dos pacotes

4.2.3 Configuração do OpenVPN

O OpenVPN foi compilado para o sistema operacional Linux, enquanto a versão para Windows foi obtida diretamente da página oficial do programa na Internet (OpenVPN, 2007). Usamos a versão 2.0 rc19[3] nos dois sistemas operacionais. A idéia inicial era compilar o OpenVPN para os 2 sistemas operacionais, pois tinhamos o intuito de usar outro modo de cifra de bloco, como o CFB, ao invés de somente o CBC.

O objetivo em usar 2 modos diferentes de operação era descobrir se existem diferenças significativas em usar o modo CBC ou CFB. Em transmissões nas quais os pacotes são ligeiramente menores, como em uma comunicação VoIP, a tendência é que um algoritmo cifrador em blocos, usando o modo de operação CFB, seja superior em desempenho se comparado com o modo de operação CBC (Elbayoumy and Shepherd, 2007).

Porém, nossos testes foram frustrados. No OpenVPN, o algoritmo AES em modo CFB apresentou problemas em manter conexão VPN cifrada ativa. Os erros encontrados remetem a um problema relacionado ao tamanho do pacote criado neste modo de operação. Na lista de discussão de desenvolvedores do OpenVPN (Yellin, 2009) encontramos outras pessoas relatando este mesmo erro, considerado como um defeito do OpenVPN. A solução dada, provisória e não testada, foi comentar uma linha de código do OpenVPN e recompilá-lo. Assim, teoricamente seria possível usar o modo CFB em conjunto com alguns algoritmos criptográficos para cifrar os dados.

Como tinhamos nos testes uma máquina com Windows e outra com Linux, precisávamos realizar uma compilação diferente do OpenVPN em cada ambiente para que a habilitação do modo CFB pudesse ser realizada. Primeiramente fizemos a mudança sugerida no código com sucesso no Linux, mas nos deparamos com a dificuldade de compilar o OpenVPN em ambiente Windows. Após tentar a compilação usando o programa MinGW[4] (MinGW team, 2009), e depois com a plataforma de desenvolvimento Microsoft Visual C++ (Microsoft Corporation, 2009), não obtivemos sucesso devido a dependências de bibliotecas e outros problemas encontrados. Em partes, esta dificuldade tem origem na documentação desatualizada para realizar a compilação do OpenVPN para Windows. Chegamos a consultar listas de discussões com o propósito de obter informações atuais que descrevessem os meios de realizar esta compilação e até conseguimos progredir mais em busca deste objetivo, mas não foi possível finalizá-la.

Voltando a configuração propriamente dita, o OpenVPN exige que apenas um arquivo seja configurado para que ele possa ser executado e usado. Neste arquivo são inseridas as informações do endereço IP de destino, de qual algoritmo de criptografia será usado, número máximo de clientes co-

[3] "rc" significa *release candidate*. O número "19" corresponde a décima nona versão disponibilizada oficialmente pelos desenvolvedores do OpenVPN como candidata a lançamento da versão 2.0 final.

[4] O MinGW (*Minimalistic GNU for Windows*) é um conjunto de ferramentas gratuitas que permitem a compilação de programas para a plataforma Windows.

nectados ao mesmo tempo, entre outros. Detalhes deste arquivo podem ser vistos na seção de Anexos deste trabalho.

Das opções de configuração disponíveis que merecem destaque podemos citar as relacionadas com a autenticação e criptografia. Para a autenticação usamos certificados previamente criados para cliente e servidor. É através destes certificados (modo SSL/TLS) que a chave secreta é compartilhada entre os participantes. A criação destes certificados é feita através do OpenSSL.

Para o OpenVPN ser usado com certificados é necessário criar, antes de tudo, um certificado denominado "certificado raiz", o certificado da entidade autenticadora. Este certificado raiz irá assinar os certificados que serão gerados para os clientes e para o próprio servidor, assim apenas os certificados assinados por este certificado raiz terão permissão de acessar a rede OpenVPN.

O próximo passo é gerar um certificado e uma chave privada para cada cliente e o servidor. O servidor e todos os clientes que participam de uma rede OpenVPN usarão o mesmo certificado raiz em suas configurações. É importante também gerar os parâmetros do algoritmo Diffie-Hellman (Diffie and Hellman, 1976) para uso no servidor, para que a troca de chaves possa ocorrer de forma segura. O processo para gerar estas chaves e certificados são amplamente descritas na documentação do OpenSSL e do OpenVPN, por isto não entraremos em detalhes quanto a isto. Podemos adiantar que o OpenVPN traz um conjunto de *scripts* que facilitam todo este trabalho de geração de chaves e certificados.

A escolha do algoritmo de criptografia usado é definido pela opção "cipher" em cada arquivo de configuração dos clientes. Esta opção, ao contrário do que o nome possa sugerir, não trata-se apenas da escolha do algoritmo de criptografia. Seria mais próximo a algo como uma "suíte criptográfica", pois além do algoritmo de criptografia também realizamos a escolha do modo de operação (detalhes na Seção 2.3.4) e, conforme o algoritmo, o tamanho da chave.

Dependendo da versão do OpenVPN e da forma como foi compilado, é possível que haja suporte a diferentes tipos de "cypher" entre os clientes e o servidor. Nestes caso, irá prevalecer a opção definida pelo servidor. Caso o cliente não suporte a opção de "cypher" definida pelo servidor, será usada a opção padrão do OpenVPN, sendo atualmente o algoritmo Blowfish com chave de 128 bits em modo CBC (*Cipher-Block Chaining*).

As opções de "cypher" disponíveis no OpenVPN podem ser obtidos através do seguinte comando em ambiente Linux:

```
openvpn --show-ciphers
```

No Windows, se a pasta do OpenVPN estiver dentre as variáveis de ambiente, o comando é quase o mesmo:

```
openvpn.exe --show-ciphers
```

Uma saída típica para este comando seria:

```
The following ciphers and cipher modes are available for use with
   OpenVPN.  Each cipher shown below may be used as a parameter to
   the --cipher option.  The default key size is shown as well as
   whether or not it can be changed with the --keysize directive.
   Using a CBC mode is recommended.

DES-CBC 64 bit default key (fixed)
RC2-CBC 128 bit default key (variable)
DES-EDE-CBC 128 bit default key (fixed)
DES-EDE3-CBC 192 bit default key (fixed)
DESX-CBC 192 bit default key (fixed)
BF-CBC 128 bit default key (variable)
RC2-40-CBC 40 bit default key (variable)
CAST5-CBC 128 bit default key (variable)
RC2-64-CBC 64 bit default key (variable)
AES-128-CBC 128 bit default key (fixed)
AES-192-CBC 192 bit default key (fixed)
AES-256-CBC 256 bit default key (fixed)
CAMELLIA-128-CBC 128 bit default key (fixed)
CAMELLIA-192-CBC 192 bit default key (fixed)
CAMELLIA-256-CBC 256 bit default key (fixed)
```

Nas opções com chave de tamanho definida como *variable* (variável), como ocorre com o Blowfish, podemos definir o tamanho da chave utilizando usando a opção “key”, seguido do tamanho de chave desejado. No caso do algoritmo criptográfico Blowfish, se não definirmos o tamanho da chave explicitamente, automaticamente será usada uma com tamanho de 128 bits. Se desejarmos especificar uma chave de outro tamanho, teremos que respeitar os critérios do algoritmo escolhido para realizar esta escolha, e usando mais uma vez o Blowfish como exemplo, com ele só é possível escolher tamanhos de chave múltiplos de 8, tais como 128, 136, 256, etc... até 448. Os algoritmos com chave de tamanho *fixed* (fixo) já determinam na própria opção o tamanho da chave que será usado.

Os arquivos de configuração do OpenVPN usados em nosso servidor e cliente podem ser visualizados no Apêndice A.1.

Escolha dos algoritmos e tamanho de chaves

Para os testes realizados neste trabalho utilizaremos com o algoritmo de criptografia AES uma chave com tamanho de 128 bits. Com o algoritmos Blowfish será usada uma chave com 128 bits de tamanho, que é o tamanho de chave padrão do algoritmo. Com o algoritmo Triple DES usaremos uma chave de 168 bits.

Para os três algoritmos de criptografia, o modo de operação escolhido foi o CBC (*Cipher-block chaining*). Este é o modo recomendado para ser usado com o algoritmo AES, Blowfish e Triple DES, uma vez que ele é o mais bem compreendido e definido para algoritmos simétricos (Frankel et al., 2003). O CBC também é o modo de operação recomendado pelos desenvolvedores do OpenVPN para todos os algoritmos de criptografia que são suportados pelo mesmo.

4.2.4 Configuração do Traffic Control

Para emulação de tráfego e problemas na rede será usada a ferramenta *Traffic Control*, mais especificamente um módulo dela denominado NetEm (Kuznetsov). Este módulo interage diretamente com o núcleo do Linux, sendo capaz de emular a perda de pacotes, duplicação, corrupção, atraso e colocar os pacotes fora de sequência. Ela pode ser acessada através do comando *tc*, que faz parte do pacote *iproute2*.

Com o uso do NetEm, os diversos cenários descritos no Capítulo 4.3 serão criados, aplicando os diferentes problemas na rede em diferentes intensidades.

Para facilitar a tarefa de inserir estes problemas, criamos *shell scripts* em *bash*[5]. Estes *scripts* tem o objetivo de gerar os cenários por meio dos comandos do *Traffic Control*, e por consequência do NetEm, responsáveis por inserir os problemas na rede e periodicamente agravar alguns dos problemas, de modo a emular os congestionamentos que alguns cenários tem como requisito.

Como exemplo de uso do NetEm, para criar um cenário com 100 milissegundos de atraso, com perda de 10% de pacotes, corrupção de 2% dos pacotes, aplicado na interface de rede nomeada de "eth0"[6], usamos o comando:

```
tc qdisc add dev eth0 root handle 1: netem delay 100ms loss 10%
   duplicate 10% corrupt 2%
```

É possível observar que neste comando usamos o *Traffic Control* ("tc") e depois invocamos como parâmetro o NetEm ("netem"), e em seguida passamos os parâmetros para inserção de anomalias na rede.

[5]O *bash* é um interpretador de comandos para sistemas operacionais tipo Unix (*Unix-like*).

[6]No sistema operacional Linux e em sistemas operacionais derivados do Unix as interfaces de rede Ethernet recebem por padrão o nome de ethX, sendo "X" o número atribuído à interface

Uma vez criado o cenário através do parâmetro "add", será necessário realizar mudanças em tempo real das condições da rede para que possamos emular o congestionamento presente em alguns dos cenários. Para isto, usamos o parâmetro "change" ao invés de "add" no comando. Assim, para aumentar o atraso da rede para 1000 milissegundos, o seguinte comando deve ser aplicado:

```
tc qdisc change dev eth0 root handle 1: netem delay 1000ms
```

Deste modo, alteramos o atraso da interface rede eth0 de 100 milissegundos para 1000 milissegundos. As demais regras permanecem inalteradas, a menos que sejam substituídas explicitamente por outro comando ou a interface seja removida do controle do *Traffic Control* através do parâmetro "remove".

A seguir, iremos apresentar alguns fatores que atuam de forma a prejudicar a comunicação em uma rede, sobretudo quando estamos transmitindo tráfego VoIP, uma vez que o mesmo é mais sensível a determinados fatores, como o atraso. Comentaremos como cada fator foi inserido em nossos testes, reportando e explicando eventuais problemas e dúvidas que surgiram e como elas foram resolvidas.

Atraso e variação do atraso

O atraso em uma rede é o tempo que um pacote leva para sair da origem e chegar ao seu destino. Este atraso pode estar relacionado a mudanças no roteamento, congestionamento na rede, pacotes aguardando na fila do roteador para ser transmitido, entre outros fatores (ITU-T, 2008a).

Para inserir o atraso e a variação do atraso (*jitter*) em nossa rede do ambiente de testes, foi preciso ter alguns cuidados. Em uma rede real, os atrasos são distribuídos de forma não-uniforme, sendo necessário usar um modo de distribuição de atraso que possa emular esta situação. Optamos então por usar a Distribuição Normal, conhecida também como Distribuição Gaussiana, através do parâmetro "distribuition normal" do NetEm. A Distribuição Gaussiana é comumente usada para descrever a variação de atraso de pacotes[7] na rede (Kuznetsov).

Quanto ativada, a distribuição do atraso ocorria como esperado. Porém, a taxa de perda de pacotes mostrava-se fora do normal, gerando em determinados testes uma perda de até 30% dos pacotes, distorcendo os resultados da qualidade da comunicação VoIP, que alcançava valores incompatíveis com os parâmetros por nós definidos nos cenários. A princípio acreditamos que o problema estaria no próprio NetEm, mas notamos que a perda de pacotes reportada por ele, através de suas próprias estatísticas, não apresentava este número de 30%. Concluimos neste ponto que o causador desta incompatibilidade não era o NetEm. Voltamos então nossa atenção para a ferramenta que criava o tráfego VoIP, o IxChariot. Foram feitos alguns testes com uma ferramenta alternativa, o *siprtp* do projeto PJSIP (Prijono, 2009), e notamos uma taxa de perda de pacotes similar a reportada pelo

[7] Em inglês é usado o termo PDV (*Packet Delay Variation*) nos estudos que envolvem a variação do atraso dos pacotes que trafegam em uma rede.

IxChariot. Nestas condições, concluimos que o problema não estava na ferramenta que gerava o tráfego VoIP.

Ainda assim, tentamos aumentar o *jitter buffer* do IxChariot. Sem sucesso, usamos uma outra ferramenta, o iPerf (NLANR/DAST, 2010), para nos ajudar a diagnosticar o problema. O iPerf é capaz de criar um tráfego de pacotes UDP e reportar os problemas encontrados durante a comunicação entre o cliente e servidor da própria aplicação.

Ao realizar os testes com o iPerf foram mostrados resultados similares ao IxChariot e *siprtp*, pois este também descartava grande parte dos pacotes. No entanto o aplicativo informava que estes pacotes eram descartados por estarem fora de sequência. Com esta informação, suspeitamos que ao inserir a variação de atraso na comunicação os pacotes que deveriam chegar antes de outros, respeitando a sequência de envio, acabavam chegando muito depois por receberem um atraso maior, e vice-versa. A razão para este problema ocorrer é que o NetEm enfileira os pacotes por tempo para serem enviados, assim os pacotes com menor atraso eram enviados antes dos pacotes com maior atraso, não respeitando o número de sequência. Este efeito colateral era sentido em nossos testes com comunicação VoIP, que ao verificar um pacote fora de sequência acabava por reportá-lo como perdido.

Para resolver este problema o *Traffic Control* dispõe de um "qdisc" (*Queueing Disciplines*, ou Disciplinas de Filas) que mantém a variação de atraso aplicado aos pacotes sem mudar a sequência de envio deles. Sendo *eth0* o nome da interface que receberá a regra, o seguinte comando é aplicado para isto:

```
tc qdisc add dev eth0 parent 1:1 pfifo limit 1000
```

Sendo *pfifo* a disciplina de escalonamento responsável por manter os pacotes na sequência correta. Esta é uma das disciplinas de enfileiramento de pacotes do tipo FIFO (*First In, First Out*) disponíveis, uma alternativa ao "qdisc" padrão usando no Linux, o *pfifo_fast* (Brown, 2006). O *pfifo_fast*, embora baseado no *pfifo*, também fornece algum grau de priorização do tráfego.

Todo "qdisc" do tipo FIFO deve ter um tamanho limite de *buffer*, para evitar problemas de sobrecarga[8] caso o "qdisc" não consiga enviá-los tão rápido quanto os recebe. Para isto, o parâmetro *limit* define quantos pacotes são possíveis de serem armazenados. Este limite pode ser também especificado em número de bytes.

Como exemplo, se quisermos gerar um atraso de 120 milissegundos na rede com uma variação de atraso de 40 milissegundos, temos que criar um atraso na rede que tenha uma variação de 100 milissegundos até 140 milissegundos, ou seja, teríamos uma variação de ±20 milissegundos. Isto pode ser feito com o seguinte comando, usando a Distribuição Normal:

[8] *buffer overflow*

```
tc qdisc add dev eth0 root handle 1: netem delay 120ms 20ms
   distribution normal
```

Perda de pacotes

Em uma rede de computadores, é comum que ocorra a perda de pacotes por diversos motivos, tais como: degradação do sinal, congestionamento e falhas gerais na rede. Ao gerar a perda de pacotes, novamente precisamos ter atenção para que ocorra uma emulação correta. Por padrão, o NetEm realiza uma perda de pacotes simplesmente percentual, descartando os pacotes conforme a percentagem desejada de forma aleatória.

Contudo, este método de perda de pacotes pode não ser o mais indicado para emular um ambiente de rede real. Muitos estudos analíticos na literatura concluem que a perda de pacotes ocorre em rajadas. Entretanto é importante considerar cuidadosamente as hipóteses feitas sobre o modelo de tráfego de uma rede , já que em alguns testes de medição de tráfego os resultados tendem a indicar que as perdas de pacotes não ocorrem em rajadas (Bolot and Vega-García, 1997) (Boutremans et al., 2002).

Porém, pode existir a necessidade de emular uma perda de pacotes que ocorra em rajadas, ou seja, recriar momentos em que ocorre uma perda maior de pacotes, intercalando com momentos de calmaria na rede. Na avaliação de uma chamada VoIP isto faz uma grande diferença na qualidade da comunicação.

Para ilustrar esta diferença, considere um cenário de rede na qual ocorre uma perda de 4% de pacotes. A perda de um pacote a cada 25 pacotes não é algo grave e não degradaria a qualidade da chamada VoIP. Porém se 1000 pacotes forem enviados, a perda de 40 pacotes consecutivos (4% do total) faria com que uma parte da chamada ficasse totalmente perdida, embora o restante dele não tivesse qualquer problema. Aos ouvidos do usuário uma perda sequencial de pacotes é muito pior que uma perda distribuída.

A versão atual do NetEm traz o recurso de correlação, com o objetivo de que a variável aleatória, usada como critério para determinar quais pacotes serão descartados, faça com que os pacotes que serão dados como perdidos fiquem próximos um do outro. Para exemplificar, em um conjunto de 100 pacotes transmitidos, uma escolha aleatória de 5 pacotes realizará uma escolha esparsa de pacotes dentro do conjunto: pacote 6, pacote 20, pacote 50, pacote 66 e pacote 75 seriam os escolhidos, por exemplo. Ao usar a correlação, é possível aumentar as chances de escolher pacotes próximos: pacote 25, pacote 30, pacote 34, pacote 41 e pacote 43. Ao escolher pacotes próximos para serem descartados é possível emular o efeito de rajada almejado.

No entanto, esta correlação não só funciona de forma pouco precisa no NetEm (Salsano et al., 2009), como também somente a correlação não é suficiente para termos resultados precisos ao emu-

lar a perda de pacotes. Na literatura, existem modelos matemáticos simples para recriar de forma adequada as rajadas de perdas de pacotes[9], são eles: o modelo de Bernoulli (Yajnik et al., 1999), Gilbert (Gilbert, 1960) e Gilbert-Elliot (Elliot, 1963).

Observando este problema no NetEm, um grupo de desenvolvedores implementou uma nova versão da ferramenta, provisoriamente chamada de NetEm2 (Salsano et al., 2009), que disponibiliza estes modelos para emulação de perda de pacotes. A idéia é inserir estas melhorias no NetEm atual, processo que já está em discussão. Atualmente o NetEm2 está disponível como uma atualização[10] para determinadas versões de núcleo do Linux.

Embora pudéssemos usar o NetEm2, preferimos continuar utilizando o NetEm, pois temos certeza que o NetEm atual não nos trará qualquer problema, uma vez que ele está incluído oficialmente nas versões estáveis do núcleo do Linux e desta forma temos a garantia de seu funcionamento pleno. Em virtude do NetEm, na versão atual, não funcionar adequadamente ao usar a variável de correlação, conforme explicado anteriormente, optamos pela distribuição uniforme de perda de pacotes, a qual o NetEm realiza de forma precisa.

Pacotes fora de sequência

Em uma transmissão de dados, os pacotes são enviados pelo remetente em uma sequência definida ao destinatário, que fica encarregado de ler a informação que está sendo passada nesta ordem estabelecida, de modo que possa compreender o conteúdo da informação. Porém, é possível que ocorra algum problema no percurso de um dos pacotes e ele acabe chegando ao destino mais cedo do que deveria, ou seja, chegando muito antes do pacote subsequente a ele. Temos então o que chamamos de um pacote fora de sequência ou fora de ordem.

A solução usada pelos programas para contornar este problema é armazenar em um *buffer* os pacotes que chegam. O *buffer* é uma área de armazenamento de dados temporária, usada por muitos programas para armazenar determinados dados que necessitam da chegada de outros para que a informação desejada fique completa e possa ser apresentada ao usuário.

Aplicações que operam em tempo real, como o VoIP, não podem esperar muito para a chegada de um pacote fora de sequência, pois isto resultará em atrasos que podem prejudicar a comunicação. Assim, este tipo de problema pode resultar em perdas de pacotes. Como exemplo disto, considere no NetEm uma regra que determine que 1 a cada 100 pacotes fique fora de sequência, e os demais pacotes fiquem com um atraso de 100 milissegundos. Isto pode ser feito de 2 maneiras atualmente no NetEm:

[9] *burst losses*
[10] *patch*

```
tc qdisc add dev eth0 root handle 1: netem delay 100ms reorder
   100% gap 1
```

Ou deste modo:

```
tc qdisc add dev eth0 root handle 1: netem delay 100ms reorder 1%
```

Da maneira como o NetEm realiza a emulação de pacotes fora de sequência, o pacote fora de sequência é enviado imediatamente, sem o atraso de 100 milissegundos, e assim ele inicia uma sequência descontínua, pois os outros pacotes esperarão 100 milissegundos para serem enviados. Se tivermos uma taxa de transferência de 100 pacotes por segundo (10 pacotes a cada 100 milissegundos), aproximadamente 10 pacotes serão enviados até a sequência dos pacotes ser reestabelecida, e os próximos 90 pacotes estarão na sequência correta. Neste cenário, temos assim, 10 pacotes em 100 pacotes fora de sequência (Bäckström, 2009). Dependendo da configuração utilizada no programa VoIP que recebe estes pacotes fora de sequência, estes pacotes podem ser descartados se o tempo limite deles no *buffer* da ferramenta se esgotar. Notamos este comportamento em nossos testes com o IxChariot.

Este problema pode ser solucionado ajustando o *buffer* do IxChariot. Se nos exemplos anteriores usassemos um *buffer* de 100 milissegundos, teríamos pouca ou nenhuma perda de pacote. Se usassemos um *buffer* ligeiramente menor, como 80 milissegundos, os pacotes fora de sequência seriam já todos descartados.

Em testes experimentais, se utilizarmos o *buffer* do *jitter* com tempo de 40 milissegundos com a regra de NetEm definindo um atraso de 100 milissegundos e 1% dos pacotes fora de sequência, obtemos um MOS de 3.05. No entanto se alterarmos o *buffer* do *jitter* para 100 milissegundos temos um MOS de 4.00, pois a quantidade de pacotes descartados no próprio *buffer* é drasticamente menor.

A explicação para este evento ocorre da seguinte maneira. Espeficicamente neste cenário temos uma taxa de 50 pacotes por segundo, ou seja, 5 pacotes são enviados a cada 100 milissegundos. Assim precisamos de um *jitter* total de 100 milissegundos. Se o pacote não puder esperar 100 milissegundos, ou seja, se o *buffer* for menor que isto, o pacote é descartado dentro do próprio *buffer*. Detalhes sobre este teste podem ser observados na Tabela 4.2, onde temos a relação do *buffer* utilizado e os datagramas perdidos e fora de sequência, em um cenário com o NetEm definindo um atraso de 100 milissegundos e 1% de pacotes fora de sequência.

Com estas informações, podemos apresentar os impactos do tamanho do *buffer* em uma conversação VoIP que apresenta pacotes fora de sequência, usando o MOS como indicador de qualidade da conversação. Acompanhe a Tabela 4.2, onde temos resultados referentes a um cenário com o NetEm definindo um atraso de 100 milissegundos e 0.1% de pacotes fora de sequência. Pode-se observar que um *buffer* de 40, 60 e 80 milissegundos apresentam os resultados de MOS bem próximos, pois eles

Tamanho do *jitter buffer* (em ms)	Datagramas perdidos	Datagramas fora de sequência	MOS
40	130	330	3.05
60	124	295	3.16
80	124	314	3.09
100	23	402	4.00

Tab. 4.2: Relação do tamanho do *buffer* do *jitter* e pacotes fora de sequência após o envio de 3000 datagramas, em um cenário com 100 milissegundos de atraso e 1% de pacotes fora de sequência

Tamanho do *jitter buffer* (em ms)	Datagramas perdidos	Datagramas fora de sequência	MOS
40	38	92	3.97
100	2	32	4.3

Tab. 4.3: Relação do tamanho do *buffer* de *jitter* e pacotes fora de sequência após o enviode 3000 datagramas, em um cenário com 100 milissegundos de atraso e 0.1% de pacotes fora de sequência

não tem o tamanho necessário, que no caso corresponde ao atraso de 100 milissegundos, para que possa obter todos os pacotes e ordená-los, sem precisar descartá-los.

O efeito prático de usarmos no NetEm uma taxa de 1% de pacotes fora de sequência resulta em 10% de pacotes fora de sequência. Assim, usamos uma taxa de 0.1%, que resulta nas medições em 1% de pacotes fora de sequência, conforme esperado.

Para evitar que os pacotes sejam descartados no *buffer*, devido a espera pelos pacotes fora da sequência, o *buffer* de *jitter* pode ser aumentado. Porém, não é comum a preocupação em aumentar o *buffer* de *jitter* por causa de pacotes fora de sequência. O *buffer* de *jitter* é usualmente ajustado para suprir a variação de atraso entre os pacotes, conforme descrito na Seção 4.2.4. Outro bom motivo é a prudência ao aumentar o *buffer*, pois atrasos acima de 100 milissegundos já podem ser notados pelo usuário, e valores acima de 150 milissegundos começam a prejudicar a qualidade da conversa.

Por isto, em nossos testes, o *buffer* de *jitter* foi ajustado conforme a variação do atraso de cada cenário. Detalhes sobre a definição do *buffer* da variação de atraso podem ser vistos no Capítulo 4.3.

Largura de banda da rede

Originalmente, as interfaces de redes do computador usado nos testes podem alcançar até 1Gbps na transferência de dados. Porém, para os cenários de teste propostos neste trabalho, em alguns momentos precisamos diminuir esta lagura de banda significativamente antes de inserir o tráfego VoIP, de modo a criar condições específicas e avaliar o comportamento do tráfego neste ambiente.

Para limitar a largura da banda da interface de rede, usamos o *Token Bucket Filter* (TBF). O TBF é uma disciplina de gerenciamento de fila para a ferramenta *Traffic Control*. Basicamente, ela é

responsável por atrasar os pacotes de forma a emular larguras da banda menores.

Para explicar como o TBF pode ser usado, vamos usar a seguinte regra do *Traffic Control* como exemplo:

```
tc qdisc add dev tap0 parent 1:1 handle 10: tbf rate 0.5mbit \
   textit{buffer} 1600 limit 3000
```

Sobre os parâmetros mostrados no comando:

- parent: o valor deste parâmetro está relacionado com a hierarquia de regras do *Traffic Control* no Linux;
- handle: o valor deste parâmetro está relacionado com a hierarquia de regras do *Traffic Control* no Linux;
- dev: o valor corresponde a interface de rede que terá o limite de banda limitado. No caso, a interface de rede em questão é a "tap0";
- tbf: é a disciplina no momento usada, responsável por realizar a limitação artificial da largura de banda da rede;
- rate: define a largura da banda. No caso, definida para 0.5 Mbits.
- buffer: número de pacotes que estão imediatamente disponíveis. Quanto maior a largura da banda ("rate") maior deve ser o *buffer*, caso contrário a largura de banda desejada não será alcançada;
- limit: número máximo de pacotes que podem ser alocados no "buffer". Se o tráfego que passar pela interface de rede definida for muito grande, é possível que o valor do parâmetro "buffer" precise ser aumentado, caso contrário os pacotes no *buffer* começarão a ser descartados;

É possível acompanhar pelo NetEm algumas estatísticas de seu funcionamento, verificando o número de pacotes e *bytes* que passaram através dele, número de pacotes que foram descartados e os que tiveram que ficar na fila. Isto pode ser feito através do comando:

```
tc -s -d qdisc ls
```

Que irá exibir algo parecido como saída no terminal:

```
qdisc netem 1: dev tap0 root limit 1000 delay 45.0ms 10.0ms loss
   0.25% duplicate 1%
  Sent 8245879 bytes 39597 pkt (dropped 86, overlimits 0 requeues
      0)
  rate 0bit 0pps backlog 0b 0p requeues 0
```

Esta saída mostra que a interface de tap0, com as regras “limit 1000 delay 45.0ms 10.0ms loss 0.2% duplicate 1%” enviou 8245879 bytes, correspondendo a 39597 pacotes. Destes pacotes, 86 foram descartados devido as regras NetEm aplicadas.

4.3 Cenários de rede

Os cenários de rede serão usados para ressaltar as diferenças de desempenho entre cada algoritmo de criptografia em diferentes condições de rede, além de auxiliar a avaliar as diferenças entre uma comunicação cifrada e outra normal. Estes cenários foram criados com base no trabalho de Markopoulou (Markopoulou et al., 2003) e em testes de medição de qualidade de chamadas VoIP em ambientes com diferentes tipos de problemas, realizados por Snyder (Snyder, 2006).

Os cenários terão diferentes larguras de banda disponíveis, além de outros atributos como perda de pacotes, atrasos, variação de atraso, pacotes fora de ordem e duplicação de pacotes. Cada um destes parâmetros irá variar conforme o cenário que desejamos obter. Alguns destes cenários terão a emulação de congestionamento, onde determinados parâmetros da rede serão degradados por um curto período e depois retornarão ao estado anterior.

Estas anomalias de rede serão criadas através da ferramenta NetEm (Kuznetsov), disponível para Linux através do pacote de ferramentas *iproute2*. A limitação da banda será realizada pelo programa “tc” (*Traffic Control*), que faz parte do pacote *iproute2*. Como o sistema operacional que realizará a geração das anomalias será o Linux, a emulação dos congestionamentos será realizada através de um *script* de comandos usando o interpretador *bash*. Este *script* irá introduzir e alterar as anomalias da rede em momentos pré-definidos.

Ao todo, serão sete os cenários criados. Inicialmente criamos 4 cenários que foram divididos em “S1”, “S2”, “S3” e “S4“, baseado nos testes de Snyder (Snyder, 2006), cada um com diferentes características. Os cenários de Snyder, para simplificar a nomenclatura, começam com a letra “S” seguido de um número para que sejam distinguidos. Os cenários de Snyder são os seguintes:

- Cenário S1: A banda será limitada em 0.1 Mbits, com 60 milissegundos de atraso, perda de 2% dos pacotes, 1% de pacotes fora de ordem, 1% de pacotes duplicados, 20 milissegundos de variaçao de atraso, e um congestionamento a cada 20 segundos de 30% de pacotes perdidos e 1.000 milissegundos de atraso com duração de 3 segundos;

- Cenário S2: A banda será limitada em 0.5 Mbits, com 60 milissegundos de atraso, perda de 2% dos pacotes, 1% de pacotes fora de ordem, 1% de pacotes duplicados, 20 milissegundos de variaçao de atraso, e um congestionamento a cada 20 segundos de 30% de pacotes perdidos e 1.000 milissegundos de atraso com duração de 3 segundos;

- Cenário S3: A banda será limitada em 0.5 Mbits, com 45 milissegundos de atraso, perda de 0.25% dos pacotes, 1% de pacotes fora de ordem, 1% de pacotes duplicados, 10 milissegundos de variaçao de atraso, e sem congestionamento;

- Cenário S4: neste cenário não será introduzido qualquer tipo de problema na rede. A banda será de 100 Mbits sem atrasos, perda de pacotes, falhas ou congestionamentos.

Os outros 2 cenários foram criados com base no artigo de Markopoulou et al., e os seus nomes começam pelo letra "M" seguido de um número para efeito de distinção:

- Cenário M1: baseado nos dados do provedor $P2$ do artigo de Markopoulou et al.. A banda será limitada em 1.544 Mbits, pois a largura de banda dos enlaces usados por Markopoulou correspondiam a tecnologia de largura de banda T1. O Cenário M1 descrito tem de 37 a 50 milissegundos de atraso, e os congestionamentos tem de 37 a 120 milissegundos de atraso. Markopoulou et al. não especificam a perda de pacotes, mas é mencionado em sua pesquisa que não mais que 0.26% pacotes na rede eram perdidos em qualquer dos caminhos, então iremos adotar este valor. Temos então um cenário que será criado com 43 milissegundos de atraso e variação de atraso de 12 milissegundos, 0.26% de pacotes perdidos e 2 congestionamentos no período total de 500 segundos, com duração de 100 segundos cada e com um atraso de 78 milissegundos e 82 milissegundos de variação de atraso. Neste cenários houve a necessidade de aumentar o *buffer* do *jitter* do IxChariot, ajustado para 100 milissegundos.

- Cenário M2: baseado nos dados do provedor $P4$ do artigo de Markopoulou et al.. A banda será limitada em 1.544 Mbits, com 40 milissegundos de atraso, perda de 0.25% dos pacotes e um congestionamento a cada 65 segundos, com duração de 3 segundos cada, com atraso de 275 milissegundos e variaçao de atraso de 50 milissegundos. Teremos então um cenário com 40 milissegundos de atraso, 0.25% de pacotes perdidos e variação de atraso de 50 milissegundos, e congestionamento com duração de 3 segundos, a cada 65 segundos, de 275 milissegundos.

Por fim, sugerimos um cenário distinto dos demais, o Cenário B1:

- Cenário B1: este cenário não tem problemas na rede e tem uma banda de 100 Mbits. A diferença neste cenário com relação aos demais está no número de chamadas VoIP simultâneas que iremos realizar, que é de 150 chamadas. Ele foi pensado para que um dos cenários testados exigisse mais do processamento do computador, então com 150 chamadas simultâneas era possível fazer com que o processador precisasse cifrar/decifrar um grande volumes de dados. O objetivo é verificar se alguns dos algoritmos se sobressai dos demais e qual a diferença na qualidade das chamadas cifradas para as não cifradas.

Para facilitar a leitura e comparação entre os cenários, na Tabela 4.4 temos todos os cenários listados e suas configurações.

Algumas observações precisam ser feitas acerca dos cenários de Snyder. Na descrição dos seus cenários, que usamos como referência, não é mencionada a duração de cada congestionamento, apenas o intervalo que eles ocorrem (Snyder, 2006). Assim, baseado nos trabalhos de (Markopoulou et al., 2002), definimos um valor de 3 segundos de duração para cada congestionamento, usando os valores do provedor P4, onde os congestionamentos ocorriam a cada 60-70 segundos.

O cenário M1 também precisou de ajustes extras. Para nos basearmos nos valores de Markopoulou tivemos que adequar o IxChariot ao cenário. Como Markopoulou mencionou que os picos de congestionamento duravam vários minutos e ocorriam com frequência, o administrador da rede teria que aumentar o *buffer* do *jitter* do aplicativo VoIP dos usuários, caso contrário nos períodos de pico a comunicação VoIP seria totalmente inviável, uma vez que nestas ocasiões o MOS alcançava valores mínimos, perto de 1, pois o *buffer* do *jitter* de apenas 40 milissegundos (valor padrão do IxChariot) não era capaz de lidar com o atraso dos pacotes, e assim vários pacotes eram descartados no próprio *buffer*.

Cenário	Banda (em Mbits)	Atraso (em ms)	Variação do Atraso (em ms)	Perdidos (em %)	Fora de Sequência (em %)	Duplicados (em %)	Congestionamento	Nº chamadas
S4	100	0	0	0	0	0	Não	1
S3	0.5	45	10	0.25	1	1	Não	1
S2	0.5	60	20	2	1	1	Sim	1
S1	0.1	60	20	2	1	1	Sim	1
M1	1.544	43	12	0.26	0	0	Sim	1
M2	1.544	40	0	0.25	0	0	Sim	1
B1	100	0	0	0	0	0	Não	150

Tab. 4.4: Resumo das características de cada cenário

4.3.1 Os cenários e resultados de Snyder

Os resultados dos testes de Snyder (Snyder, 2006) podem ser vistos na Figura 4.2. Estes testes de Snyder foram realizados com dez equipamentos que implementam VPN por SSL. Em seus testes, ele usou o programa Shunra Virtual Enterprise para criar os problemas na rede e o programa GL Communications para medir a qualidade da chamada.

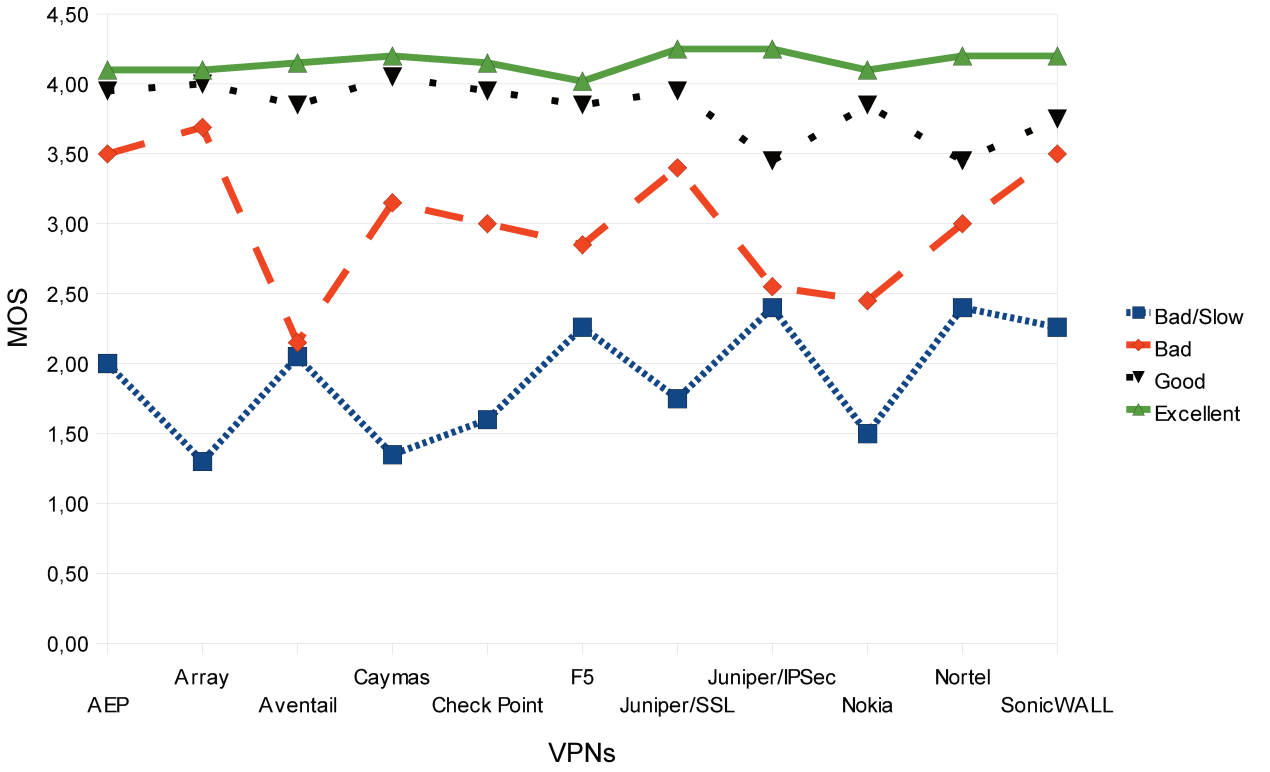

Fig. 4.2: Resultados de qualidade de chamada VoIP nos testes de Snyder, com diferentes cenários de rede.

Em seu trabalho, é possível observar uma grande diferença de resultados de MOS entre os equipamentos, que usam diferentes implementações e tecnologias para realizar a criptografia das chamadas. Ele examinou 4 diferentes cenários, começando com uma rede de 100 Mbps e poucos milissegundos de latência, chegando até uma rede com qualidade ruim, de 100 Kbps, com 60 milissegundos de latência e outros problemas. Estes cenários foram chamados de “unimpaired”, “good”, “bad” e “bad/slow”.

Embora usemos em nosso trabalho diferentes programas para realizar as mesmas tarefas, esperamos usar seus resultados na Seção 4.4 como uma referência para determinar se os cenários que criamos terão resultados próximos aos de Snyder e assim validar a efetividade dos mesmos ao retratarem as mesmas condições.

4.4 Testes e resultados

Cada algoritmo de criptografia selecionado para o trabalho, conforme descrito no Capítulo 2.3, foi confrontado com os cenários de rede descritos no Capítulo 4.3. Contando com os testes que

envolvem a ausência de criptografia nos dados, teremos ao todo 24 resultados: 6 cenários de testes com 3 diferentes algoritmos de criptografia mais a ausência de criptografia.

Ao final de cada chamada, os valores referentes a qualidade da comunicação VoIP eram obtidos através do MOS (*Mean Opinion Score*). Cada teste foi executado 12 vezes (12 chamadas VoIP), e o pior e melhor resultado de cada bateria de testes foram descartados. O objetivo testa ação é diminuir a possibilidade de um resultado destoar dos demais.

Com os 10 resultados de MOS restantes foi calculada a média aritmética (Fórmula 4.2), o desvio padrão (Fórmula 4.3) e o coeficiente de variação percentual (Fórmula 4.1) para chegarmos aos resultados finais do cenário junto ao algoritmo criptográfico.

$$c_v = \frac{\sigma}{\overline{x}} \cdot 100 \tag{4.1}$$

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.2}$$

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{4.3}$$

A média aritmética é representada por $\overline{x}$, o desvio padrão por σ e o coeficiente de variação percentual terá a notação c_v. O desvio padrão não será usado diretamente como base para qualquer análise dos nossos resultados, mas ele é usado no cálculo do coeficiente de variação.

Para cada cenário, organizamos os resultados em 2 tabelas: na primeira tabela teremos os dez valores de MOS (*Mean Opinion Score*) resultantes de cada teste para cada algoritmo e na segunda tabela apresentamos os resultados de média aritmética, desvio padrão e coeficiente de variação de cada algoritmo criptográfico.

Para os nomes dos algoritmos exibidos nestas tabelas, preferimos usar os nomes dados a eles na opção de "cipher" do OpenVPN:

1. BF-CBC: algoritmo de criptografia Blowfish, utilizando o modo de operação CBC com chave de 128 bits de tamanho;

2. AES-128-CBC: algoritmo de criptografia AES, utilizando o modo de operação CBC com chave de 128 bits de tamanho;

3. DES-EDE3-CBC: algoritmo de criptografia Triple DES, utilizando o modo de operação CBC com chave de 192 bits de tamanho;

4. NONE: ausência de criptografia, ou seja, a transmissão dos dados não será criptograda;

Com os cenários que exigem mais tempo entre os congestionamentos, precisamos de um tempo de chamada maior que 2 minutos, pois um tempo menor que este não é suficiente para retratar todo o comportamento do cenário. Para estes casos, delimitamos o limite de 4 minutos por chamada, pois em algumas pesquisas realizadas (Ong et al., 2008) (Marktest, 2003), este parece ser o valor aproximado de duração das chamadas, seja de clientes residenciais ou de ambientes privados, como empresas. Poderíamos extender este tempo também, porém avaliar chamadas com 10 minutos ou mais fogem um pouco da duração típica de uma conversa telefônica, e por consequência tiraria um pouco o foco do trabalho de recriar situações tão reais quanto possíveis.

A Seção 4.4.1 mostra alguns resultados obtidos da performance bruta de alguns algoritmos criptográficos. Os testes foram executados usando o OpenSSL, utilizando dois processadores Athlon de diferentes velocidades.

A Seção 4.4.9 trata da discussão geral dos cenários, onde vamos comparar um cenário com outro em busca de padrões de comportamento entre os cenários e algoritmos, acrescentar informações importantes a possíveis dúvidas e expor as dúvidas e problemas encontrados.

Quanto a avaliação do nível de satisfação do usuário ao realizar uma chamada VoIP em determinado cenário, a Seção 4.4.9 apresenta esta e outras informações relacionadas.

A Seção 4.4.9 avalia a média e variação dos resultados de MOS que cada algoritmo obteve ao longo de todos os testes nos cenários.

Por fim, recomendamos a leitura sequêncial dos cenários, pois muitos deles detalham informações que são comuns a outros cenários, e para a leitura como um todo de nosso trabalho não ficar repetitiva optamos por detalhar as observações mais pertinentes conforme elas aparecessem. Contudo, sempre que possível iremos indicar o cenário onde determinada observação, compartilhada com outro cenário, se encontra.

4.4.1 Performance dos algoritmos cifrando blocos de dados

Devido as diferenças de arquitetura entre os algoritmos criptográficos, existem diferenças significantes de performance entre eles que podem ser observadas em testes de força bruta. Sendo assim, realizamos alguns testes prévios e nesta seção apresentamos os resultados obtidos. A importância destes testes para o nosso trabalho reside na posterior comparação dos mesmos com os resultados dos testes propostos, usando as ferramentas OpenVPN, Traffic Control e IxChariot. Assim, podemos confrontar a performance dos algoritmos nestes testes de cifragem de dados com o desempenho final obtido pelos mesmos em nossos cenários.

A Tabela 4.5 mostra um comparativo de algoritmos criptográficos, disponibilizado através do comando *openssl speed* em um computador equipado com o processador AMD Athlon 64 3000+. Este comparativo relaciona a velocidade de cada algoritmo criptográfico ao cifrar diferentes tamanhos de

Tipo	Tamanho do bloco de dados				
	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
RC4	188114.87k	227333.56k	237757.84k	239903.20k	241175.79k
DES CBC	42256.56k	44694.71k	45323.76k	45130.47k	45158.75k
3DES	15181.85k	15382.14k	15473.14k	15537.12k	15515.15k
Blowfish CBC	65848.08k	71450.88k	73471.22k	74269.75k	74201.25k
AES-128 CBC	48640.51k	76543.16k	86177.23k	92996.50k	93568.69k

Tab. 4.5: Velocidade relativa dos algoritmos mostrada pelo OpenSSL, em um AMD Athlon 64 3000+ com 512MB de memória RAM executando Ubuntu Linux 8.04.

Tipo	Tamanho do bloco de dados				
	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
RC4	316251.09k	446492.57k	476507.73k	459422.82k	481235.42k
DES CBC	86284.27k	87993.67k	87742.73k	64148.82k	85663.33k
3DES	28688.01k	30752.77k	31349.92k	29730.49k	30750.86k
Blowfish CBC	132048.67k	145040.93k	151692.58k	149810.15k	127930.33k
AES-128 CBC	127076.89k	165454.37k	185885.97k	188084.08k	186590.91k

Tab. 4.6: Velocidade relativa dos algoritmos mostrada pelo OpenSSL, em um AMD Athlon X2 5200+ com 512MB de memória RAM executando Ubuntu Linux 8.04.

blocos de dados em um período de apenas 3 segundos. Pode-se notar que o algoritmo RC4, um cifrador de fluxo, é capaz de cifrar muito mais dados que os demais, que são algoritmos cifradores de bloco. A Tabela 4.6 mostra o mesmo comparativo realizado em um computador equipado com AMD Athlon X2 5200+. Pelos dados obtidos, nota-se que um processador com performance significativamente superior pode ser até mais de 50% veloz na cifragem de dados.

Seria interessante realizar o mesmo teste incluindo o AES ou Blowfish em modo CFB (detalhes na seção 2.3.1), porém a versão atual estável do OpenSSL (0.9.8g), no período em que realizamos os testes, não fornece estas opções para o teste de velocidade.

4.4.2 Cenário S4

Como mencionado anteriormente, temos no Cenário S4 as condições ideais de rede, sem qualquer anomalia e banda limitada pela interface de rede (de 100 Mbits). Conforme esperávamos, todos os algoritmos de criptografia apresentaram o MOS (*Mean Opinion Score*) máximo possível de ser medido pelo IxChariot, de 4,38. A Tabela 4.7 mostra os resultados de MOS de cada algoritmo e a Tabela 4.8 exibe os valores da média aritmética, desvio padrão e coeficiente de variação percentual para o Cenário S4.

Algoritmo	Testes									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38
BF-CBC	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38
DES-EDE3-CBC	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38
NONE	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38	4,38

Tab. 4.7: Relação de resultados de MOS dos algoritmos de criptografia no Cenário S4.

Algoritmo	$\overline{x}$	σ	c_v
AES-128-CBC	4,38	0	0
BF-CBC	4,38	0	0
DES-EDE3-CBC	4,38	0	0
NONE	4,38	0	0

Tab. 4.8: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário S4.

Como é possível observar, os testes neste cenário não geraram resultados conclusivos. Os algoritmos criptográficos não interferiram na qualidade das chamadas, alcançando os maiores resultados possíveis de MOS. Chegamos a medir valores de 4,37 em algumas das chamadas, mas com o critério de exclusão do menor valor eles não foram contabilizadoss.

4.4.3 Cenário S3

O Cenário S3 é baseado no cenário "good" de Snyder. Com o Cenário S3 já é possível ver claramente a influência dos problemas de rede que inserimos na qualidade das chamadas VoIP. Temos um total de 10 resultados válidos de MOS para cada algoritmo criptográfico avaliado no Cenário S3, que podem ser visualizados na Tabela 4.9.

Algoritmo	Testes									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	4,14	4,15	4,12	4,07	4,14	3,98	3,97	4,00	4,11	3,98
BF-CBC	4,05	4,04	4,10	4,10	4,14	4,06	4,17	4,03	4,09	4,03
DES-EDE3-CBC	4,02	4,02	4,16	4,02	4,04	4,14	4,14	4,14	4,17	4,03
NONE	4,03	4,14	4,10	4,03	4,05	4,10	4,19	4,05	4,08	4,18

Tab. 4.9: Relação de resultados de MOS dos algoritmos de criptografia no Cenário S3.

Com base nos resultados de MOS, obtivemos os valores da média aritmética, desvio padrão e coeficiente de variação percentual para o Cenário S3 que estão representados na Tabela 4.10.

Algoritmo	$\overline{x}$	σ	$\mathbf{c_v}$
AES-128-CBC	4,066	0,07545	1,86
BF-CBC	4,081	0,04771	1,17
DES-EDE3-CBC	4,088	0,06630	1,62
NONE	4,095	0,05874	1,43

Tab. 4.10: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário S3.

Comparando com os resultados obtidos por Snyder no cenário "good", apresentados na Figura 4.2, temos resultados muito parecidos de MOS no Cenário S3. A maior parte dos valores obtidos por Snyder neste cenário ficaram em torno de valor 4,0, corroborando os nossos resultados.

Dos dados obtidos, temos como melhor resultado neste cenário o teste com a ausência de criptografia, que obteve o MOS de 4,095. Em seguida temos o Triple DES e logo após o Blowfish. Por último, temos o algoritmo AES.

Estes resultados são curiosos se formos compará-los com os resultados de desempenho destes algoritmos criptográficos disponibilizado pelo OpenSSL, que pode ser visto na Tabela 4.5. Enquanto o teste do OpenSSL indicaria que o algoritmo mais rápido seria o Blowfish, nossos testes indicaram o Triple DES como o mais rápido, sendo que nos testes do OpenSSL o Triple DES é um dos mais lentos. Contudo, em nossos testes a diferença é muito pequena.

Podemos afirmar que a diferença de desempenho entre os algoritmos foi, na prática, perto de nula. Um usuário não conseguiria perceber a diferença entre o uso do algoritmo de criptografia AES, que apresentou o pior resultado com um MOS de 4,066, e a ausência de criptografia que apresentou o melhor resultado com um valor de 4,095.

4.4.4 Cenário S2

O Cenário S2 apresenta diversos problemas na rede, e era esperado que resultados de MOS baixos fossem obtidos nos testes com este cenário. Os valores de MOS que obtivemos na bateria de testes podem ser visualizados na Tabela 4.11.

Algoritmo	Testes									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	2,30	2,30	2,46	2,34	2,34	2,45	2,35	2,44	2,41	2,31
BF-CBC	2,45	2,36	2,42	2,49	2,45	2,35	2,48	2,42	2,45	2,39
DES-EDE3-CBC	2,41	2,53	2,37	2,45	2,41	2,45	2,38	2,37	2,48	2,47
NONE	2,40	2,43	2,39	2,47	2,39	2,43	2,39	2,58	2,47	2,49

Tab. 4.11: Relação de resultados de MOS dos algoritmos de criptografia no Cenário S2.

Algoritmo	$\overline{x}$	σ	c_v
AES-128-CBC	2,370	0,06377	2,69
BF-CBC	2,426	0,04742	1,95
DES-EDE3-CBC	2,432	0,05308	2,18
NONE	2,444	0,06059	2,48

Tab. 4.12: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário S2.

Comparando com os resultados obtidos por Snyder em seu cenário “bad”, temos resultados pouco parecidos de MOS no Cenário S2. A maior parte dos resultados obtidos por Snyder neste cenário ficaram em torno do valor 3,0, enquanto em nossos testes este valor ficou em 2,4, uma diferença expressiva.

Dos resultados calculados em nosso teste e exibidos na Tabela 4.12 podemos verificar que a ausência de criptografia mostrou o melhor desempenho. Dentre os algoritmos de criptografia temos o Triple DES com o melhor desempenho, seguido por Blowfish e, por último, o AES. As colocações obtidas por cada algoritmo são exatamente as mesmas do Cenário S3, porém tivemos uma variação maior nos resultados de MOS dos algoritmos neste cenário se comparados aos do Cenário S3, provavelmente originado dos congestionamentos que o Cenário S2 aplica durante a conversação.

E, novamente, não temos uma diferença de desempenho entre os algoritmos criptográficos que possa ser definida como expressiva.

4.4.5 Cenário S1

O Cenário S1 é quase idêntico ao Cenário S2, com exceção da largura de banda da rede; enquanto o Cenário S2 tem uma banda de 500 kbps, o Cenário S1 tem uma banda de apenas 100 kbps. Os valores de MOS que obtivemos na bateria de testes e a média aritmética, desvio padrão e coeficiente de variação percentual de cada algoritmo podem ser visualizados na Tabela 4.13 e 4.14, respectivamente.

Algoritmo	Testes									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	2,17	2,26	2,21	2,07	2,10	2,16	2,22	2,15	2,18	2,17
BF-CBC	2,18	2,30	2,27	2,29	2,28	2,11	2,19	2,22	2,16	2,29
DES-EDE3-CBC	2,17	2,17	2,28	2,29	2,22	2,15	2,15	2,30	2,40	2,18
NONE	2,35	2,23	2,28	2,16	2,23	2,21	2,13	2,24	2,15	2,16

Tab. 4.13: Relação de resultados de MOS dos algoritmos de criptografia no Cenário S1.

Algoritmo	$\overline{x}$	σ	c_v
AES-128-CBC	2,169	0,05547	2,56
BF-CBC	2,229	0,06641	2,98
DES-EDE3-CBC	2,231	0,08333	3,73
NONE	2,214	0,06753	3,05

Tab. 4.14: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário S1.

Fazendo a comparação com os resultados obtidos por Snyder em seu cenário “bad/slow”, temos resultados pouco parecidos de MOS no Cenário S1, mas diferente do que ocorreu no Cenário S2, nosso valores de MOS ficaram superiores ao invés de inferiores aos de Snyder. A maior parte dos resultados obtidos por ele neste cenário ficaram em torno do valor de 2,0, enquanto em nossos testes este valor ficou em torno de 2,22.

Aqui tivemos uma mudança na classificação de desempenho dos algoritmos. Enquanto no Cenário S3 e no Cenário S2 temos a ausência de criptografia como o que apresenta maior MOS, seguido do Triple DES, Blowfish e AES, no Cenário S1 observamos que a ausência de criptografia ficou na terceira posição, com o Triple DES em primeiro e o Blowfish em segundo. O AES, assim como ocorreu no Cenário S3 e no Cenário S2 permaneceu como o algoritmo de criptografia de menor MOS, ficando em último lugar.

Estes resultados de classificação, comparados aos obtidos nos cenários anteriores, reforçam a dificuldade em dizer que determinado algoritmo criptográfico é superior ao outro quanto aos resultados de qualidade para uma chamada VoIP.

Neste cenário tivemos uma variação de resultados de MOS de cerca de 3% para cada algoritmo de criptografia, superior aos dos cenários S3 e S2, mostrando que este cenário é o mais instável do que os apresentados anteriormente. Sobre a variação, podemos destacar o Triple DES, que apresentou a maior variabilidade em seus resultados e o AES, que foi o algoritmo de criptografia que menos teve variação nos seus valores de MOS.

4.4.6 Cenário M1

Os resultados de MOS do Cenário M1 são apresentados na Tabela 4.15 e os cálculos de média aritmética, desvio padrão e coeficiente de variação percentual são mostrados na Tabela 4.16.

Algoritmo	**Testes**									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	3,81	3,79	3,82	3,85	3,74	3,75	3,80	3,88	3,81	3,82
BF-CBC	3,79	3,82	3,76	3,82	3,76	3,83	3,76	3,75	3,80	3,76
DES-EDE3-CBC	3,78	3,83	3,78	3,81	3,80	3,83	3,80	3,79	3,85	3,77
NONE	3,86	3,74	3,85	3,85	3,78	3,79	3,71	3,80	3,84	3,82

Tab. 4.15: Relação de resultados de MOS dos algoritmos de criptografia no Cenário M1.

Algoritmo	$\overline{\mathbf{x}}$	σ	$\mathbf{c_v}$
AES-128-CBC	3,807	0,01094	1,09
BF-CBC	3,785	0,00810	0,81
DES-EDE3-CBC	3,804	0,00681	0,68
NONE	3,804	0,01318	1,32

Tab. 4.16: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário M1.

Neste cenário, diferente dos anteriores, temos o AES como o algoritmo de criptografia com melhor desempenho, obtendo um MOS de 3,807. Um resultado interessante, devido ao fato do AES ter sido o último colocado nos Cenários S3, S2 e S1.

Em seguida, com resultados bem próximos, temos o Triple DES e a ausência de criptografia com o mesmo valor de MOS: 3,804. Por último temos o Blowfish, um pouco mais distante com o MOS de 3,785. Outro ponto destaque é a variação dos resultados, que fica em torno de 1%, significativamente menores do que nos cenários S1, S2 e S3.

Markopoulou et al. avaliou a qualidade da comunicação VoIP como boa para este cenário, lembrando que usamos os dados do Provedor P2 de sua pesquisa. Em nosso trabalho, usando a Tabela 2.4 como referência, temos uma qualidade de chamada que poderia deixar a maioria dos usuários satisfeitos, corroborando com os seus testes.

4.4.7 Cenário M2

Comparado com o Cenário M1, o Cenário M2 apresenta resultados interessantes, pois houve uma grande similaridade de entre estes 2 cenários. É possível visualizar os resultados do Cenário M2 nas Tabelas 4.17 e 4.18.

Algoritmo	**Testes**									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	3,99	3,98	4,01	4,05	4,02	4,01	3,97	3,98	4,03	3,98
BF-CBC	4,00	3,97	3,99	3,99	4,03	4,02	4,01	3,94	4,01	3,98
DES-EDE3-CBC	3,99	4,00	3,97	3,98	4,03	4,01	4,03	3,96	3,99	4,02
NONE	3,93	3,98	3,97	3,94	4,00	3,98	3,99	3,99	4,05	4,05

Tab. 4.17: Relação de resultados de MOS dos algoritmos de criptografia no Cenário M2.

Algoritmo	$\overline{\mathbf{x}}$	σ	$\mathbf{c_v}$
AES-128-CBC	4,002	0,00654	0,65
BF-CBC	3,994	0,00659	0,66
DES-EDE3-CBC	3,998	0,00610	0,61
NONE	3,988	0,00988	0,99

Tab. 4.18: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário M2.

Neste cenário temos novamente o AES como o algoritmo de criptografia com melhor desempenho, obtendo um MOS de 4,002, em seguida o Triple DES e a ausência de criptografia com o mesmo valor de MOS, de 3,988, e por último o Blowfish com o MOS de 3,994. Ou seja, não só tivemos a mesma classificação de desempenho dos algoritmos criptográficos observadas no Cenário M1 como também tivemos o algoritmo Triple DES com o mesmo MOS que a ausência de criptografia, fato este que ocorreu nos cenários M1 e M2.

Quanto a variação, podemos constatar que o coeficiente de variação de resultados de MOS para cada algoritmo de criptografia não chegou a 1% no pior caso, registrado pela ausência de criptografia. Ou seja, tivemos uma baixa variação.

Markopoulou et al. avaliou a qualidade da comunicação VoIP como ruim para este cenário, lembrando que usamos os dados do Provedor P4 de seu trabalho. Isto ocorreu pois nos instantes de congestionamento, que duravam pouco, mas eram frequêntes, havia forte degradação da qualidade da comunicação VoIP. A Figura 4.3 mostra um gráfico com a distribuição da variação do MOS durante

uma das chamadas VoIP neste cenário. Podemos notar três quedas no resultado do MOS. O fato é que fazer a média aritmética dos valores de MOS podem "ocultar" estes momentos, pois no restante do tempo a comunicação ocorria em qualidade muito boa. Sendo assim, observando nossos números finais, poderiámos qualificar este cenário no mínimo como "bom", mas na prática muitos usuários ficariam insatisfeitos com a qualidade da chamada.

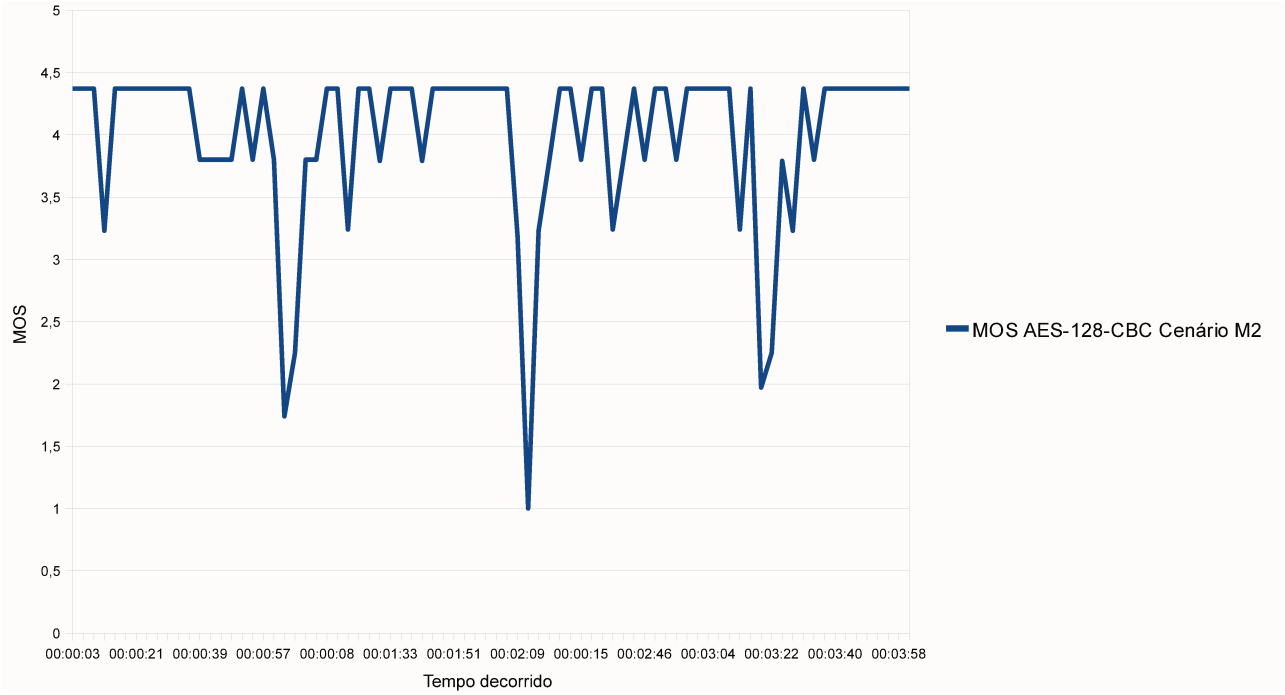

Fig. 4.3: Distribuição dos resultados de MOS em uma chamada cifrada com algoritmo AES no cenário M2

Já podemos notar a partir deste cenário e do M1 que os cenários baseados no trabalho de Markopoulou et al. mostraram uma variabilidade muito menor nos resultados se compararmos aos cenários propostos por Snyder. Podemos observar que o coeficiente de variação de resultados de MOS para cada algoritmo de criptografia fica em torno de 1%. Este comportamento é, provavelmente, causado pela maior instabilidade dos cenários de Snyder, que tem maiores variações de atraso, perda de pacotes maiores e tem ainda a introdução dos problemas de pacotes fora de sequência e pacotes duplicados.

4.4.8 Cenário B1

Como o algoritmo de criptografia de um programa usado para criação de redes privadas depende, sobretudo, da capacidade do *hardware* (Pena and Evans, 2000) para ter um bom desempenho, foi

proposto o Cenário B1, que nada mais é que um ambiente de força-bruta, em que o objetivo principal é demonstrar a diferença entre estes algoritmos de criptografia, ou que pudesse diferenciar chamadas cifradas das não cifradas.

Neste teste foram realizadas 150 chamadas VoIP simultâneas sem inserir qualquer tipo de problema na rede. O objetivo era verificar se o MOS médio de todas estas chamadas ao testar com cada algoritmo mostrava alguma diferença entre eles.

O número de 150 chamadas foi definido de acordo com o *hardware* utilizado. Observamos em testes experimentais que, ao chegar no número de 150 chamadas simultâneas, os resultados de MOS começavam a se alterar de forma significativa para todos os algoritmos de criptografia. Foi este, então, o critério adotado para a escolha deste número de chamadas. Vale ressaltar que este número, naturalmente, varia conforme o poder computacional do *hardware* utilizado.

Assim como ocorreu com os outros cenários, executamos 10 testes. Os resultados de MOS destes testes podem ser vistos na Tabela 4.19 e e 4.20.

Algoritmo	**Testes**									
	1	2	3	4	5	6	7	8	9	10
AES-128-CBC	3,09	2,98	3,59	3,84	3,95	3,95	3,98	3,73	3,88	3,86
BF-CBC	3,73	3,85	3,76	3,67	3,90	3,8	3,8	3,72	3,94	3,86
DES-EDE3-CBC	3,76	3,92	3,78	3,91	3,57	3,69	3,63	3,58	3,53	3,83
NONE	3,87	3,85	3,88	3,83	3,97	3,90	3,87	3,90	4,08	3,92

Tab. 4.19: Relação de resultados de MOS dos algoritmos de criptografia no Cenário B1.

Algoritmo	$\overline{\mathbf{x}}$	σ	$\mathbf{c_v}$
AES-128-CBC	3,685	0,36244	9,84
BF-CBC	3,803	0,08525	2,24
DES-EDE3-CBC	3,720	0,14166	3,81
NONE	3,907	0,07212	1,85

Tab. 4.20: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos no Cenário B1.

Nota-se que a diferença entre os algoritmos criptográficos e a ausência de criptografia fica evidente, diferente dos cenários anteriores. Devido ao atraso gerado pela criptografia, os resultados de MOS dos algoritmos ficaram atrás dos testes onde nenhuma criptografia foi aplicada.

Contudo, todos os valores de média obtidos ao final dos testes de cada algoritmo e da ausência de criptografia ficam com MOS entre 3,6 e 4,0, que segundo a Tabela 2.4, deixariam alguns usuários

insatisfeitos com a qualidade. Teoricamente, o usuário ainda não conseguiria distinguir a qualidade de uma chamada criptografada com Triple DES (o pior resultado) e outra sem criptografia (o melhor resultado).

Um aspecto interessante destes resultados é que eles indicam que existe uma tendência de que um algoritmo de criptografia distancie do outro conforme o número de chamadas aumente. Isto sugere que o MOS entre os algoritmos pode variar o suficiente para que a diferença de qualidade entre eles comece a ser percebida pelo usuário. Neste caso, a escolha do algoritmo de criptografia, ou da própria decisão de aplicar criptografia, influencie de forma significativa na satisfação do usuário.

É importante ressaltar que este cenário é diferente dos demais, pois: as características da rede não foram alteradas em qualquer momento, não é baseado, a princípio, em um cenário real e mais de uma chamada é realizada simultaneamente. Por estes motivos, não incluimos seus resultados em alguns momentos ao avaliar o comportamento geral dos cenários, assunto discutido na seção 4.4.9.

4.4.9 Discussão dos resultados

Após as baterias de testes, se faz necessária uma seção dedicada a discutir e avaliar os resultados que obtivemos. Nas seções anteriores, chegamos a levantar alguns detalhes e questionamentos pontuais, mas pretendemos neste momento discutir de uma forma geral os resultados, englobando todo o contexto do nosso trabalho e comparando um cenário com outro. Contudo, conforme explicado na seção 4.4.8, sobre o cenário B1, não compararemos diretamente seus resultados com os outros cenários no decorrer desta seção, porém serão feitas algumas considerações a respeito deste cenário ao final da desta seção.

Após finalizar os testes, é possível observar que os cenários S1 e S2 tiveram resultados finais de MOS (*Mean Opinion Score*) próximos, variando de 2,2 a 2,4, e o Cenário S3 apresentava valores superiores e com níveis de satisfação de usuário muito melhores. Nestes três cenários verificamos que o algoritmo criptográfico AES apresentou o mais baixo desempenho dentre os algoritmos.

A Tabela 4.21 traz a média aritmética de cada algoritmo criptográfico em cada cenário testado, e entre parênteses a posição de classificação de desempenho do mesmo no cenário. Na última coluna temos a média aritmética ($\overline{x}$) do MOS do algoritmo ao longo dos cenários testados. Com este valor, podemos classificar os algoritmos de acordo com a qualidade de MOS apresentada nas chamadas cifradas pelos mesmos. Não incluímos os resultados do Cenário S4, pois eram pouco relevantes ao demonstrar as diferenças entre os algoritmos, uma vez que todos tiveram o mesmo resultado final de MOS, que correspondia ao valor máximo possível.

Algoritmo	**Cenários**					
	S3	S2	S1	M1	M2	$\overline{x}$
AES-128-CBC	4,066 (4)	2,370 (4)	2,169 (4)	3,807 (1)	4,002 (1)	3,283 (4)
BF-CBC	4,081 (3)	2,426 (3)	2,229 (2)	3,785 (3)	3,994 (3)	3,303 (3)
DES-EDE3-CBC	4,088 (2)	2,432 (2)	2,231 (1)	3,804 (2)	3,998 (2)	3,311 (1)
NONE	4,095 (1)	2,444 (1)	2,214 (3)	3,804 (2)	3,988 (2)	3.309 (2)

Tab. 4.21: Resultados de média aritmética de MOS dos algoritmos criptográficos nos cenários.

Assim, através desta média aritmética, é possível observar que o DES-EDE3-CBC apresentou o melhor desempenho de um modo geral, enquanto o AES-128-CBC apresentou o pior. Contudo, não é possível dizer que determinado algoritmo tem efetivamente um desempenho superior com os resultados obtidos, que variaram tão pouco de um algoritmo para outro. A diferença de qualidade para chamada VoIP usando o algoritmo DES-EDE3-CBC (3,311) e outra com o AES-128-CBC (3,283), aos ouvidos do usuário, é imperceptível.

Embora pareça óbvio que a ausência de criptografia deveria ter se saído melhor, é preciso ter cuidado ao realizar este tipo de afirmação. Os testes de Snyder mostraram que uma VPN usando o protocolo SSL/TLS poderia aumentar a qualidade da comunicação, pois a conexão TCP criada pela VPN melhorava a qualidade das chamadas reordenando e retransmitindo os pacotes. Isto não se encaixa em nosso caso, pois nossa VPN opera sobre o protocolo UDP, e não TCP.

Contudo, esta melhoria no trabalho de Snyder é relativa pois Gast explica que este tipo de melhoria vem as custas de uso de mais banda de rede[11], pois a VPN precisa detectar o erro na transmissão e retransmitir o pacote sem criar atrasos. Entretanto se a largura da banda de rede for de apenas 100 kbps isto não é totalmente possível, ao menos usando o *codec* G.711 pois o mesmo consome cerca de 80 kbps de banda com o cabeçalho UDP/IP e RTP. Já com 500 kbps, os resultados de Snyder foram superiores devido a possibilidade de realizar estas retransmissões sem qualquer problema.

Outro fato importante, ainda ligado ao parágrafo anterior, é a respeito dos resultados de MOS do Cenário S2 e S1, que podem ser observados na Figura 4.4. Conforme mencionado na Seção 4.4.5, a única diferença entre eles é a banda de rede menor do Cenário S1, com apenas 100 kbps. Esta banda de rede menor prejudicou os resultados de Snyder devida as retransmissões de pacotes que não eram possíveis de realizar com apenas 100 kbps, e assim como Snyder também tivemos resultados de queda de MOS no Cenário S1, que tinha as mesmas condições. Contudo não temos este comportamento de retransmissões de pacotes devido ao tipo de túnel VPN usado por nós, o UDP. A Figura 4.5 mostra a taxa de transferência de uma chamada no Cenário S2 e Cenário S1 com o algoritmo Blowfish em

[11]Gast usa o termo *headroom* para a sobra de banda da rede utilizada para realizar as retransmissões de pacotes, e a tradução mais próxima do termo seria "reserva dinâmica"

nosso ambiente, onde podemos notar que a taxa de transferência em ambos cenários são basicamente iguais, não alcançando o limite de 100 kbps em nenhum momento. Ou seja, mesmo não tendo o comportamento de retransmissões que modificaram os resultados de MOS de Snyder em um cenário similar ao cenário S1, também tivemos queda de MOS, mostrando que o aumento na largura de banda da rede pode influenciar positivamente nos resultados, mesmo que a princípio a banda de rede pareça suficiente.

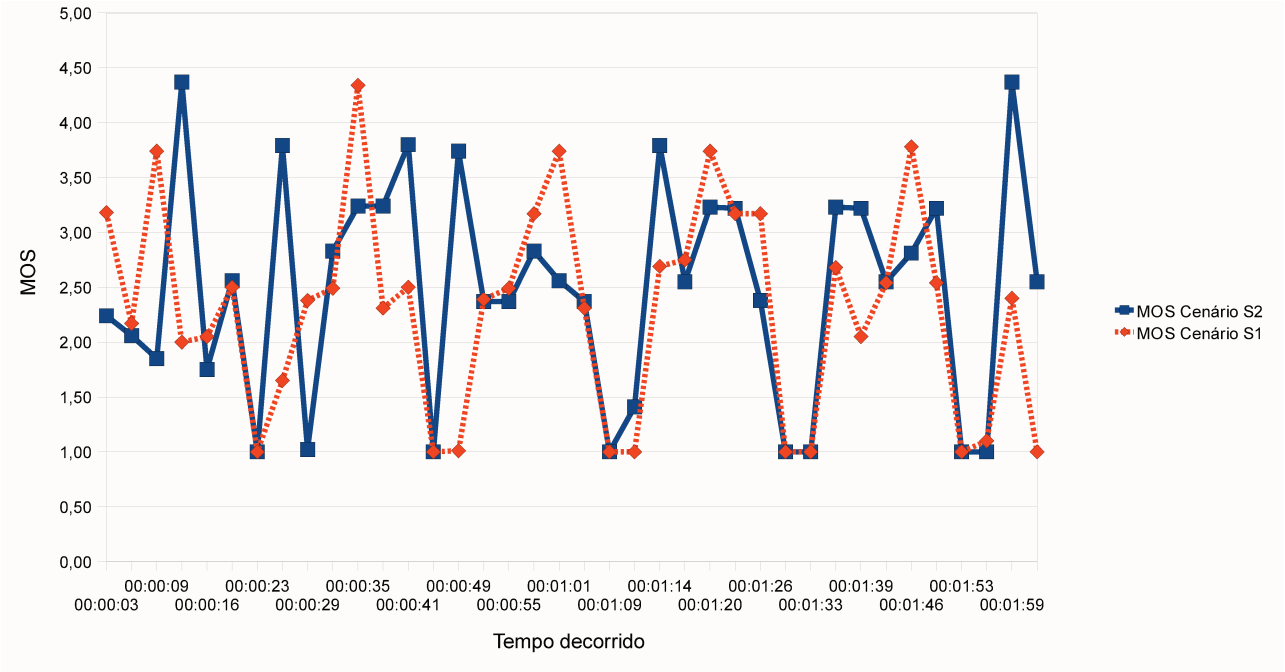

Fig. 4.4: Valores de MOS obtidos de uma chamada com 2 minutos de duração no Cenário S1 e outra no Cenário S2 com o algoritmo Blowfish.

Analisando os resultados, observamos que a média dos atrasos dos resultados do Cenário S2 eram significativamente menores que a do Cenário S1. Em busca de explicações, notamos que o atraso máximo medido durante uma chamada em S1 era sempre consideravelmente maior que no Cenário S2 em todas as chamadas realizadas. Por consequência, a média dos atrasos era afetada, e ficava consideravalmente maior em S1. Como exemplo para demonstrar este comportamento, as Figuras 4.6 e 4.7 mostram a variação em milissegundos do atraso máximo medido e média do atraso, respectivamente, de uma chamada VoIP criptografada com o algoritmo criptográfico Blowfish nos cenários S2 e S1 em 10 chamadas.

Buscando compreender estes resultados de atraso, analisamos uma das chamadas cifradas com o algoritmo Blowfish no Cenário S2 e outra no Cenário S1 e comparamos os atrasos fim-a-fim medidos ao longo dos 2 minutos de duração da chamada. O resultado desta comparação pode ser vista na Figura 4.8. Podemos notar que o Cenário S1 não foi capaz de manter o atraso no nível proposto,

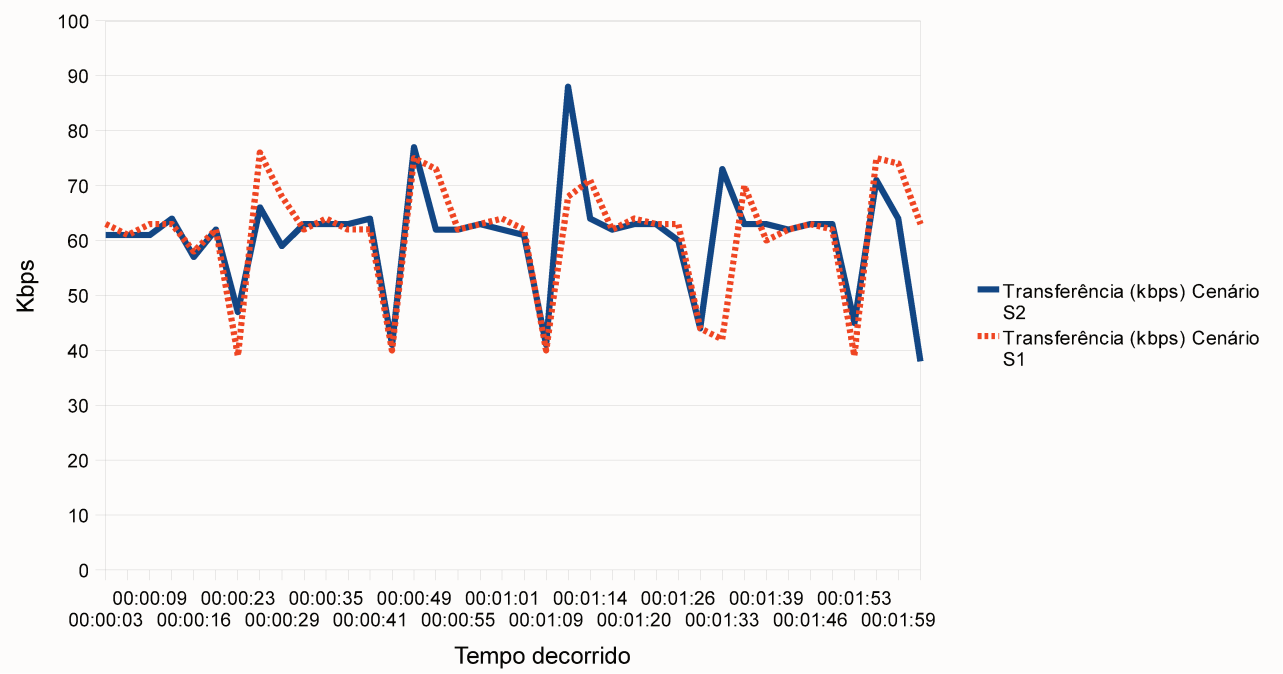

Fig. 4.5: Taxa de transferência de uma chamada com 2 minutos de duração no Cenário S1 e outra no Cenário S2 com o algoritmo Blowfish.

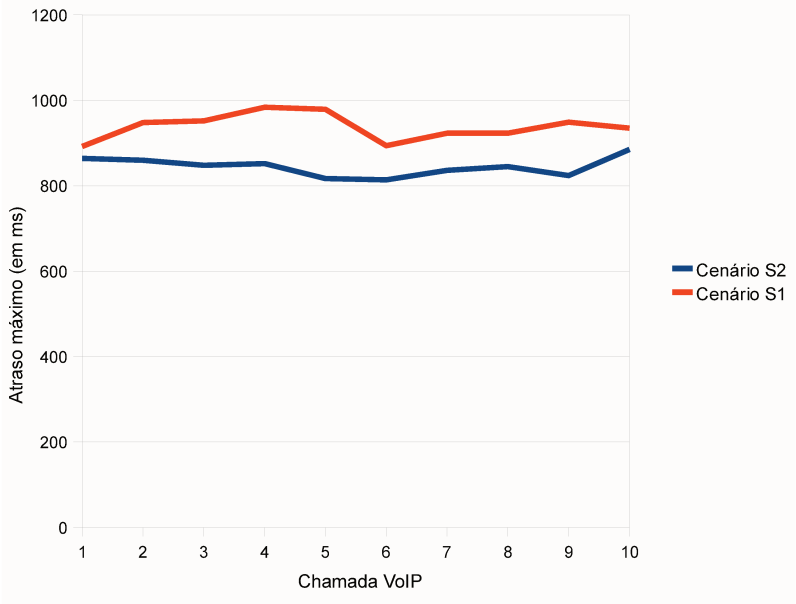

Fig. 4.6: Atraso máximo em milissegundos medido de uma chamada VoIP criptografada com o algoritmo Blowfish nos cenários S2 e S1 em 10 chamadas

em torno de 60 milissegundos, e verificamos comportamentos parecidos em todas as outras chamadas realizadas. Outro fato interessante é que os picos de atraso, gerados pelos congestionamentos,

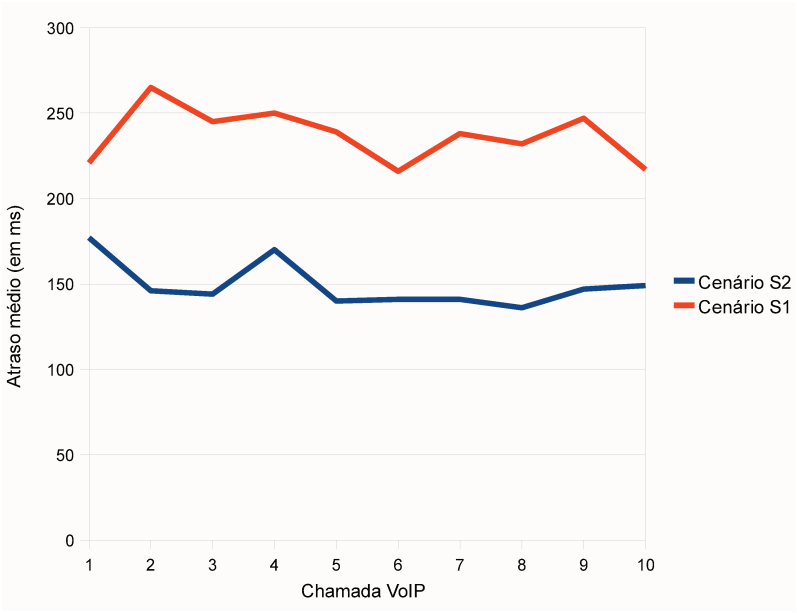

Fig. 4.7: Média dos atrasos em milissegundos, medido de uma chamada VoIP criptografada com o algoritmo Blowfish nos cenários S2 e S1 em 10 chamadas

costumavam serem maiores e com maior duração no Cenário S1 do que em S2.

Nos cenários baseados nos trabalhos de Markopoulou et al. tivemos inversão dos resultados em relação aos obtidos por Snyder. Verificamos que o AES, último colocado nos cenários de Snyder, apresentou o melhor desempenho nos testes de Markopoulou et al.. Isto nos leva a crer que a diferença entre os 2 conjuntos de testes ora prejudicou e ora favoreceu o AES. Os cenários de Snyder apresentavam uma perda de pacotes maior, inserção de pacotes fora-de-sequência e picos de atraso maiores resultantes dos congestionamentos agressivos de curta duração que ocorria em pequenos intervalos de tempo. Por sua vez, os cenários de Markopoulou et al. apresentavam perdas de pacotes signicativamente menores, sem inserção de pacotes fora-de-sequência e picos de atraso menores durante os congestionamentos. Porém, estes congestionamentos tinham maior tempo de duração.

Outra característica que diferencia os 2 conjuntos de testes é a variação nos resultados de MOS obtidos, ocasionados pela característica de instabilidade do cenário, definida pela quantidade de anomalias inseridas e pela gravidade das mesmas. Um atraso de 150 milissegundos tem uma gravidade maior que um atraso de 50 milissegundos, por exemplo, então é esperado que o cenário que tenha o atraso maior apresente um MOS mais baixo.

Os coeficientes de variação dos algoritmos em cada cenário podem ser visualizados na Tabela 4.22. Os cenários de Snyder apresentaram uma variação maior nos resultados de MOS nos algoritmos criptográficos. Em alguns casos a variação chegou a 3,73%, que ocorre com o algoritmo Triple DES no Cenário S1. Os cenários de Markopoulou tiveram uma variação consideralmente menor, sendo a

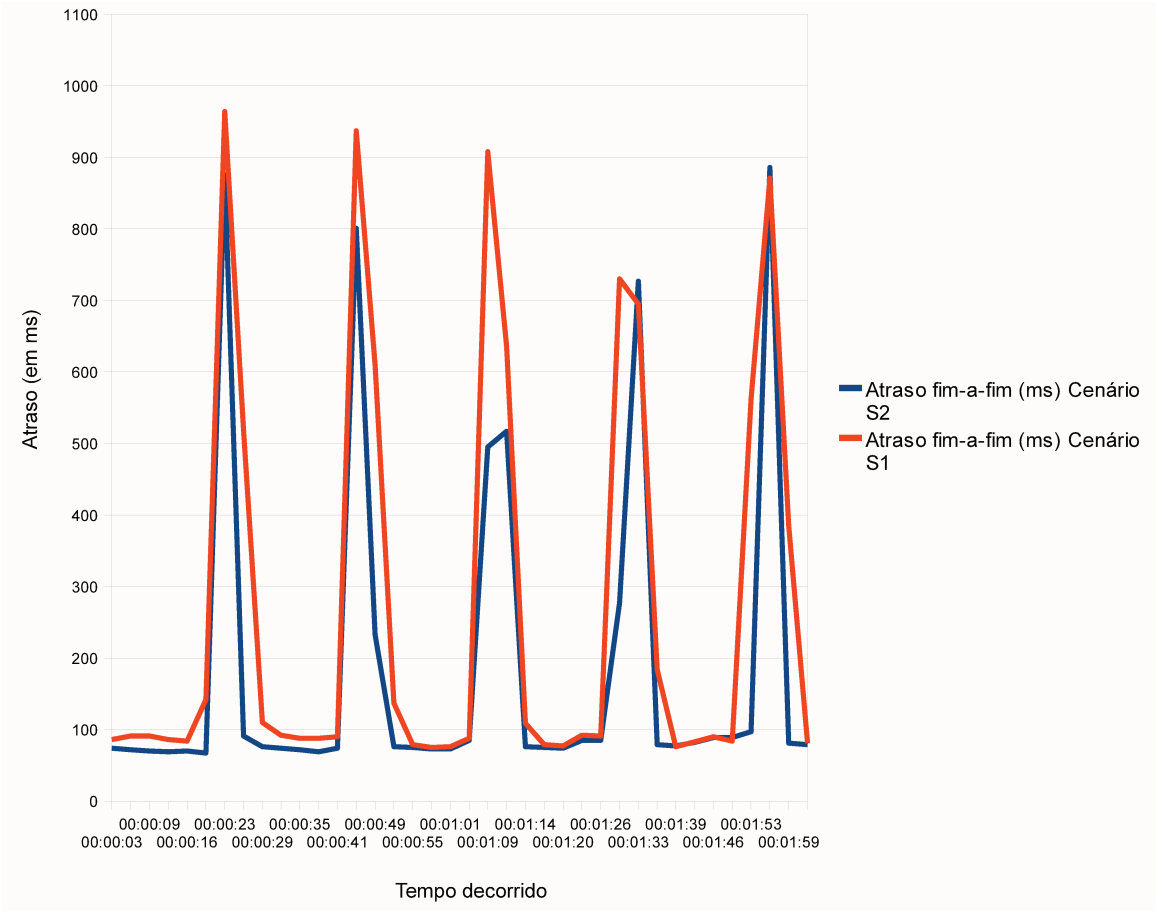

Fig. 4.8: Comparativo de atraso fim-a-fim, em milissegundos, entre o Cenário S2 e S1 ao longo de uma chamada VoIP de 2 minutos de duração criptografada com o algoritmo Blowfish

maior variação foi de 1,32% obtida pela ausência de criptografia (*none*).

Algoritmo	Cenário S3	Cenário S2	Cenário S1	Cenário M1	Cenário M2
AES-128-CBC	1,86 (1)	2,69 (1)	2,56 (4)	1,09 (4)	0,65 (3)
BF-CBC	1,17 (4)	1,95 (4)	2,98 (3)	0,81 (2)	0,66 (2)
DES-EDE3-CBC	1,62 (2)	2,18 (3)	3,73 (1)	0,68 (3)	0,61 (4)
NONE	1,43 (3)	2,48 (2)	3,05 (2)	1,32 (1)	0,99 (1)

Tab. 4.22: Resultados do coeficiente de variação dos algoritmos criptográficos nos cenários.

Os cenários de Markopoulou et al. variaram menos devido a semelhança da configuração adotada entre os cenários os quais foram criados baseados em sua pesquisa.

Sobre o Cenário B1, fica evidente em seus resultados a diferença de MOS das chamadas cifradas comparada com as sem cifragem, algo que não ocorreu nos outros cenários. Devido ao Cenário B1

exigir muito processamento, é possível afirmar que esta foi a razão para termos resultados de MOS com diferenças expressivas se compararmos com os resultados dos outros cenários. Com relação aos algoritmos de criptografia, o Blowfish se saiu significamente melhor que o AES e o Triple DES, que tiveram resultados próximos de MOS.

Influência dos cenários nos algoritmos

Com os resultados obtidos em nosso trabalho, é possível observar como cada cenário de rede que testamos influenciou nos resultados da qualidade da chamada VoIP. Podemos assim verificar qual cenário causou maior variação de resultados de MOS.

Esta análise pode ser feita de forma similar a que foi realizada com cada algoritmo nos respectivos cenários. Inicialmente é necessário fazer uma média aritmética utilizando as médias aritméticas do MOS de cada algoritmo criptográfico ao final de cada cenário, cálculo ilustrado pela Fórmula 4.4.

$$\overline{x}_{cenario} = \frac{1}{4}(\overline{x}_{aes} + \overline{x}_{bf} + \overline{x}_{3des} + \overline{x}_{none}) \tag{4.4}$$

Posteriormente é calculado o desvio padrão usando estas mesmas médias aritméticas dos algoritmos para, em seguida, obter o coeficiente de variação percentual.

A Tabela 4.23 mostra estes valores e os níveis de satisfação (Bergstra and Middelburg, 2003). O Cenário M2 foi o cenário mais estável de todos, pois os algoritmos criptográficos obtiveram valores de MOS sempre bem próximos, com pouca variação. Em seguida temos os cenários S3 e M1, que obtiveram uma variação de 0,27% e 0,30%, respectivamente. Finalmente temos os cenários S2 e S1 como os que apresentaram maior instabilidade nos valores de MOS dos algoritmos, apresentando valores de 1,36% e 1,3% respectivamente.

Cenário	$\overline{\mathrm{x}}$	σ	$\mathbf{c_v}$	**Satisfação do usuário**
S4	4,38	0	0,0	Muito satisfeito
S3	4,083	0,01240	0,30	Satisfeito
S2	2,418	0,03286	1,36	Não recomendado
S1	2,211	0,02885	1,30	Não recomendado
M1	3,800	0,01010	0,27	Alguns usuários insatisfeitos
M2	3,996	0,00597	0,15	Alguns usuários insatisfeitos
B1	3,779	0,02614	2,61	Alguns usuários insatisfeitos

Tab. 4.23: Média aritmética, desvio padrão, coeficiente de variação percentual e satisfação esperada do usuário independente do algoritmo criptográfico no cenário.

Esta análise é interessante do ponto de vista de elaboração de novos cenários. Se o objetivo fosse

diversificar os testes, esta característica de instabilidade pode se tornar um ponto interessante a ser observado na criação de novos cenários para testes.

Acreditamos que o ideal para os testes seja utilizar cenários que possam gerar resultados em vários níveis de satisfação de qualidade da chamada com uma variação alta. A Tabela 4.23 mostra os níveis de satisfação obtidos em nossos cenários de acordo com as informações da Tabela 2.4.

Variação de desempenho dos algoritmos

Nos resultados dos testes também podemos analisar qual dos algoritmos criptográficos mostrou maior variação nos seus resultados. Podemos, desta forma, determinar o algoritmo mais "instável" nas condições impostas. Os valores apresentados não incluem os resultados do cenário B1, pelo motivos explicados na seção 4.4.8.

Com base nas informações da Tabela 4.21 podemos chegar aos valores de média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos dos cenários S1, S2, S3, S4, M1 e M2, exibidos na Tabela 4.24.

Algoritmo	$\overline{\mathbf{x}}$	σ	$\mathbf{c_v}$
AES-128-CBC	3,283	0,933	28,41
BF-CBC	3,303	0,900	27,24
DES-EDE3-CBC	3,311	0,902	27,26
NONE	3,309	0,904	27,33

Tab. 4.24: Média aritmética, desvio padrão e coeficiente de variação percentual dos algoritmos criptográficos em todos os cenários, exceto o cenário B1.

Concluimos então que o AES-128-CBC se mostrou o algoritmo com maior variação de resultados durante todos os testes, variando os resultados de MOS para as chamadas em 28,41%, podendo ser considerado o algoritmo mais suscetível as anomalias de rede inseridas em nossos testes. Contudo, os resultados obtidos com os algoritmos de criptografia ficaram todos muito próximos e, assim, não é seguro apontar que este ou aquele algoritmo sofre mais/menos variação que os demais. Contudo, a influência dos cenários nos resultados de MOS é nítida, pois a variação é resultado da disparidade entre os mesmos, uma vez que temos cenários que apresentaram valores de média aritmética elevados, como é o caso do Cenário M1, e outros com valores baixos, como ocorreu com o Cenário S2.

Outro ponto que deve ser destacado é que o algoritmo NONE não mostrou qualquer diferença que fosse perceptível para os demais quanto a variação de seus resultados de MOS.

Capítulo 5

Conclusão e trabalhos futuros

Este trabalho teve como objetivo verificar o impacto da criptografia em comunicações VoIP estabelecidas em diferentes cenários de rede. Estes cenários foram criados usando recursos do programa Traffic Control do Linux e shell scripts, que foram utilizados para acionar as seqüências de comandos que emulavam estes ambientes. A criptografia foi realizada pelo OpenVPN, que criava uma interface de rede virtual onde os dados trafegados por meio dela eram todos cifrados. As chamadas VoIP eram criadas e avaliadas pelo programa IxChariot.

Foi possível observar que o impacto da criptografia na qualidade da chamada é perceptível apenas quando várias chamadas simultâneas são realizadas, pois neste cenário há um grande volume de dados e, conseqüentemente, exige-se muito processamento para a cifragem e decifragem dos dados. Neste caso, é possível observar diferenças no MOS entre chamadas cifradas das não-cifradas, assim como diferenças de desempenho entre os algoritmos de criptografia.

Nos cenários com apenas uma chamada por vez, onde vários problemas de rede foram introduzidos, como atraso, variação de atraso e perda de pacotes, não havia diferenças observáveis entre uma chamada VoIP cifrada e uma não cifrada. O principal problema para a chamada VoIP nestes casos era ocasionada pelos problemas inseridos na rede. Nestas condições, devemos observar outros critérios para a escolha do algoritmo de criptografia, como seu histórico de vulnerabilidades. Outra contribuição deste trabalho que podemos destacar foi o levantamento bibliográfico sobre o assunto, citando pesquisas, testes e artigos que envolvem equipamentos VPN, OpenVPN, controle de tráfego e VoIP.

A criação dos cenários via implementação de shell scripts foi outra contribuição interessante, mostrando como é possível organizar as regras do Traffic Control e TBF em uma seqüência de comandos para este propósito, recriando um ambiente de rede base e emulando congestionamentos na mesma.

Sobre as ferramentas e métodos que usamos para criar o ambiente de testes, podemos concluir que elas foram adequadas à proposta do trabalho. Usando os resultados de Snyder (Snyder, 2006)

como base para verificar a efetividade de nossos cenários, pôde-se observar que os resultados que obtivemos eram próximos aos obtidos por Snyder. No entanto, seria interessante uma alternativa gratuita ao IxChariot para realizar os testes, na qual também tivesse a capacidade de gerar as chamadas e informar o MOS, detalhes sobre o atraso, perda de pacotes, etc. Algumas ferramentas gratuitas estão caminhando neste sentido, como o PJSIP (Prijono, 2009), que já está sendo preparado para trazer suporte ao RTCP-XR (Friedman et al., 2003), trazendo estatísticas detalhadas e suporte a métricas de monitoramento da comunicação VoIP, e assim informar a cada momento da conversação o MOS da chamada. Atualmente há apenas um suporte parcial que pode ser ativado na compilação, e informa apenas algumas das estatísticas disponíveis. Uma ferramenta desta natureza, que pode ser acessada por meio de comandos e com emissão de seus resultados no próprio terminal, exigiria menos intervenção manual e traria mais agilidade aos testes, vantagens estas que não foram possíveis de observar em nossos métodos.

É possível concluir que a emulação de um ambiente tem a vantagem de proximidade do cenário real de uma rede, mas é mais trabalhosa e propensa a pequenos problemas devido ao grande número de ferramentas usadas em conjunto. A simulação, por ocorrer em ambientes totalmente controlados e exigir normalmente apenas o próprio simulador, pode ser facilmente automatizada e gerar menos contratempos.

Sobre os trabalhos futuros, notamos ao final do trabalho que seria interessante ter realizado os mesmos testes com várias chamadas VoIP simultâneas, a exemplo do que fizemos no Cenário B1, aumentando o uso do processador do computador, para verificar se em outros cenários poderíamos notar uma diferença na qualidade das chamadas VoIP entre os algoritmos de criptografia. Estes testes poderiam ocorrer também com diferentes algoritmos de criptografia e/ou diferentes tamanhos de chave, mas para manter o escopo deste trabalho não foram concretizadas nesta pesquisa.

Alguns cenários de rede que usamos impactavam mais na qualidade da chamada que outros. Uma outra sugestão para trabalhos futuros é isolar, ou mesmo combinar, determinados problemas que inserimos em cada cenário de rede e verificar como cada um deles impacta na qualidade da chamada VoIP.

Referências Bibliográficas

Humberto Abdelnur, Vincent Cridlig, Radu State, and Olivier Festor. VoIP Security Assessment: Methods and Tools. In Henning Schulzrinne Saverio Niccolini and Radu State, editors, *The 1st IEEE workshop on VoIP Management and Security: VoIP MaSe*, pages 29–34, Vancouver Canada, 2006. IEEE/IFIP, IEEE Communications Society. doi: 10.1109/VOIPMS.2006.1638119. URL http://dx.doi.org/10.1109/VOIPMS.2006.1638119.

J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. RFC 3830 (Proposed Standard), August 2004. URL http://www.ietf.org/rfc/rfc3830.txt. Updated by RFC 4738.

Roberto Barbieri, Danilo Bruschi, and Emilia Rosti. Voice over IPsec: Analysis and solutions. *Computer Security Applications Conference, Annual*, 0:261+, 2002. ISSN 1063-9527. doi: 10.1109/CSAC.2002.1176297. URL http://dx.doi.org/10.1109/CSAC.2002.1176297.

Rodrigo Barbosa, Arthur Callado, Carlos A. Kamienski, Stênio Fernandes, Dênio Mariz, Judith Kelner, and Djamel Sadok. Avaliação do desempenho de aplicações VoIP P2P. In *XXIV Simpósio Brasileiro de Redes de Computadores*, Junho 2006. URL http://www.lbd.dcc.ufmg.br:8080/colecoes/sbrc/2006/st8_3.pdf.

J. A. Bergstra and C. A. Middelburg. ITU-T Recommendation G.107 : The E-Model, a computational model for use in transmission planning. Technical report, ITU-T, 2003.

Jean-Chrysostome Bolot and Andrés Vega-García. The case for FEC-Based error control for packet audio in the Internet. *ACM Multimedia Systems*, 1997.

Catherine Boutremans, Gianluca Iannaccone, and Christophe Diot. Impact of link failures on VoIP performance. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 63–71, New York, NY, USA, 2002. ACM. ISBN 1-58113-512-2. doi: http://doi.acm.org/10.1145/507670.507680.

Martin A. Brown. *Traffic Control HOWTO*. The Linux Documentation Project, October 2006. URL http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html. Acessado em 16/03/2008.

Ingmar Bäckström. Performance measurement of IP networks using the two-way active measurement protocol. Master's thesis, School of Computer Science and Engineering Royal - Institute Of Technology, April 2009.

J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. URL http://www.ietf.org/rfc/rfc4880.txt. Updated by RFC 5581.

Feng Cao and Saadat Malik. Vulnerability analysis and best practices for adopting IP telephony in critical infrastructure sectors. *IEEE Communications Magazine*, 44:138–145, 2006.

Mark Carson and Darrin Santay. Nist net: a linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, 2003. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/956993.957007.

Leandro Carvalho, Edjair Mota, Regeane Aguiar, Ana F. Lima, Jose Neuman de Souza, and Anderson Barreto. An E-model implementation for speech quality evaluation in VoIP systems. In *ISCC '05: Proceedings of the 10th IEEE Symposium on Computers and Communications*, pages 933–938, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2373-0. doi: http://dx.doi.org/10.1109/ISCC.2005.23.

Marius Cornea, Ping Tak Peter Tang, and John Harrison. *Scientific Computing on Itanium-based Systems*. Intel Press, June 2002.

Renshou Dai. A technical white paper on Sage's PSQM test. August 2000. URL http://sageinst.com/downloads/925/psqmwp8_00.pdf.

C. Demichelis and P. Chimento. IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). RFC 3393 (Proposed Standard), nov 2002. URL http://www.ietf.org/rfc/rfc3393.txt.

Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. URL http://tools.ietf.org/rfc/rfc5246.txt.

Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

Ashraf D. Elbayoumy and Simon J. Shepherd. Stream or block cipher for securing VoIP? *International Journal of Network Security*, 5(2):128 – 133, September 2007. URL http://ijns.femto.com.tw/contents/ijns-v5-n2/ijns-2007-v5-n2-p128-133.pdf.

E. O. Elliot. Estimates of error rates for codes on burst-noise channels. *Bell Systems Technical Journal*, 42:1977–1997, September 1963.

EMVCo. *EMV Integrated Circuit Card Specifications for Payment Systems - Book 2 - Security and Key Management*. EMVCo, 4.2 edition, June 2008. URL http://www.emvco.com/specifications.aspx?id=155.

Horst Feistel. Cryptography and computer privacy. *j-SCI-AMER*, 228(5):15–23, May 1973. ISSN 0036-8733.

S. Frankel, R. Glenn, and S. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602, September 2003. URL http://www.ietf.org/rfc/rfc3602.txt.

Alan O. Freier, Paul C. Kocher, and Philip L. Karlton. The SSL Protocol Version 3.0, 1996. URL http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt. Acessado em 05/07/2007.

T. Friedman, R. Caceres, and A. Clark. RTP Control Protocol Extended Reports (RTCP XR). RFC 3611 (Proposed Standard), 2003. URL http://www.ietf.org/rfc/rfc3611.txt.

Matthew Gast. Strangely, ssl vpns can help voip call quality, March 2006. URL http://www.oreillynet.com/etel/blog/2006/03/strangely_ssl_vpns_can_help_vo.html. Acessado em 21/01/2008.

E. N. Gilbert. Capacity of a burst-noise channel. *Bell Systems Technical Journal*, 39:1253–1265, September 1960.

Prateek Gupta and Vitaly Shmatikov. Security analysis of Voice-over-IP protocols. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 49–63, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2819-8. doi: http://dx.doi.org/10.1109/CSF.2007.31.

Stephen Hemminger. Network Emulation with netEm. In *Proceedings of the 6th Australia's National Linux Conference (LCA'05)*, April 2005. URL http://developer.osdl.org/shemminger/netem/LCA2005_paper.pdf. Acessado em 07/12/2008.

Oliver Hersent, David Guide, and Jean-Pierre Petit. *Telefonia IP*. Addison Wesley, trad. 1 ed. edition, 2002.

ITU-T. ITU-T P.800. Methods for subjective determination of transmission quality - Series P: telephone transmission quality; methods for objective and subjective assessment of quality, August 1996.

ITU-T. G.113 Appendix I: Provisional planning values for the equipment impairment factor Ie. Technical report, ITU-T, 2001. URL http://www.itu.int/rec/T-REC-G.113.

ITU-T. ITU-T Recommendation G.8261: Timing and synchronization aspects in packet networks. Technical report, International Telecommunication Union, April 2008a. URL http://www.itu.int/rec/T-REC-G.8261/en.

ITU-T. E-model Tutorial, 2008b. URL http://www.itu.int/ITU-T/studygroups/com12/emodelv1/tut.htm.

Ixia. Assessing VoIP call quality using the E-model, 2004. URL http://www.ixiacom.com/library/white_papers/display?skey=voip_quality#a.

Ixia. IxChariot, 2008. URL http://www.ixchariot.com/. Acessado em 06/05/2008.

K. Kaukonen and R. Thayer. A stream cipher encryption algorithm Arcfour. Internet-Draft (work in progress), Dezembro 1999. URL http://www.mozilla.org/projects/security/pki/nss/draft-kaukonen-cipher-arcfour-03.txt.

Stephen Kent and Karen Seo. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 4301, December 2005. URL http://www.ietf.org/rfc/rfc4301.txt.

Neal Koblitz. Elliptic curve cryptosystems. *j-MATH-COMPUT*, 48(177):203–209, jan 1987. ISSN 0025-5718.

Alexey N. Kuznetsov. Netem. URL http://www.linuxfoundation.org/en/Net:Netem. Acessado em 07/12/2008.

LARTC Project. Linux advanced routing & traffic control, 2010. URL http://www.lartc.org/. Acessado em 25/04/2010.

John E Lincoln. Hosted VoIP market research review, 2009. URL http://www.freedomiq.com/blog/hosted-voip-market-research-review/comment-page-1/. 20/07/2010.

Lan Luo. A note to modes of block cipher as stream cipher without information loss. *International Journal Of Computational Cognition*, 6(3):48 – 50, 2008. URL http://www.yangsky.com/ijcc/pdf/ijcc634.pdf.

Athina P. Markopoulou, Fouad A. Tobagi, and Mansour J. Karam. Assessment of VoIP quality over internet backbones. In *IEEE Infocom*, pages 150–159, 2002.

Athina P. Markopoulou, Fouad A. Tobagi, and Mansour J. Karam. Assessing the quality of voice communications over internet backbones. *IEEE/ACM Trans. Netw.*, 11(5):747–760, 2003. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2003.818179.

Marktest. Rede fixa: duração das chamadas, 2003. URL http://www.marktest.com/wap/a/n/id~32c.aspx.

Microsoft Corporation. Microsoft visual C++, Outubro 2009. http://msdn.microsoft.com/en-us/visualc/default.aspx.

V. S. Miller. Uses of elliptic curves in cryptography. pages 417–426, 1986.

MinGW team. MinGW, Outubro 2009. http://www.mingw.org/.

Antonio Nappa. Analysis and implementation of secure and unsecure Voice Over IP environment and performance comparison using openser, 2007.

National Institute of Standards and Technology. *FIPS PUB 186: DIGITAL SIGNATURE STANDARD (DSS)*. NIST, Maio 1994. URL http://www.itl.nist.gov/fipspubs/fip186.htm.

National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. NIST, Outubro 1999. URL http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

National Institute of Standards and Technology. *Recommendation for Key Management – Part 1: General (Revised)*. NIST, Março 2007. URL http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.

NLANR/DAST. Iperf - the TCP/UDP bandwidth measurement tool, 2010. URL http://iperf.sourceforge.net/. Acessado em 07/07/2008.

National Institute of Standards and Technology. Specification for the advanced encryption standard (AES). Federal Information Processing Standards Publication 197, 2001. URL http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. Acessado em 06/09/2007.

Rob SG Ong, Johan Post, Harry van Rooij, and Jan de Haan. Call-duration and triage decisions in out of hours cooperatives with and without the use of an expert system. February 2008. URL http://ukpmc.ac.uk/articlerender.cgi?artid=1274480.

OpenSSL. OpenSSL, 2007. URL http://www.openssl.org. Acessado em 12/08/2007.

OpenVPN. OpenVPN, 2007. URL http://openvpn.net/. Acessado em 12/08/2007.

C.J.C. Pena and J. Evans. Performance evaluation of software virtual private networks (VPN). *Local Computer Networks, Annual IEEE Conference on*, 0:522, 2000. ISSN 0742-1303. doi: http://doi.ieeecomputersociety.org/10.1109/LCN.2000.891094.

Benny Prijono. PJSIP, 2009. URL http://www.pjsip.org. Acessado em 01/10/2008.

R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359340.359342.

Matt J. B. Robshaw. Stream Ciphers Technical Report TR-701. Technical report, RSA Laboratories, 1995. URL http://security.ece.orst.edu/koc/ece575/rsalabs/tr-701.pdf. Acessado em 06/10/2009.

RSA Laboratories. RSA Laboratories' frequently asked questions about today's cryptography, version 4.1, 2000. URL http://www.rsa.com/rsalabs/node.asp?id=2152. Acessado em 03/06/2009.

RSA Security. What are some other block ciphers?, 2009a. URL http://www.rsa.com/rsalabs/node.asp?id=2254. Acessado em 06/10/2009.

RSA Security. What is a stream cipher?, 2009b. URL http://www.rsa.com/rsalabs/node.asp?id=2174. Acessado em 03/10/2009.

Stefano Salsano, Fabio Ludovici, and Alessandro Ordine. *Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel*. University of Rome 'Tor Vergata', 2.0 edition, October 2009. URL http://netgroup.uniroma2.it/twiki/pub/Main/NetEm2/TR-loss-netem.pdf).

Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pages 191–204, London, UK, 1994. Springer-Verlag. ISBN 3-540-58108-1.

H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003. URL http://www.ietf.org/rfc/rfc3550.txt. Updated by RFCs 5506, 5761.

Joel Snyder. Test shows voip call quality can improve with ssl vpn links, September 2006. URL http://www.networkworld.com/reviews/2006/022006-ssl-voip-test.html. Acessado em 10/08/2007.

William Stallings. *Cryptography and Network Security: Principles and Practices*. Prentice Hall, 4 ed. edition, 2005.

Inc. Sun Microsystems. *Sun VirtualBox: User Manual*, 2009. URL http://download.virtualbox.org/virtualbox/3.0.12/UserManual.pdf. Acessado em 26/05/2009.

Andrew S. Tanenbaum. *Redes de Computadores*. Elsevier, Rio de Janeiro, trad. 4 ed. edition, 2003.

Nadeem Unuth. Reasons for choosing Voice over IP - VoIP advantages, 2010. URL http://voip.about.com/od/voipbasics/a/ReasonsForVoIP.htm. 20/07/2010.

VirtualBox. VirtualBox, 2008. URL http://www.virtualbox.org. Acessado em 05/03/2008.

VoIP Troubleshooter LLC. Measuring Voice Quality, 2009. URL http://www.voiptroubleshooter.com/basics/mosr.html. Acessado em 06/12/2009.

Ellen M. Voorhees and Lori P. Buckland, editors. *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, volume Special Publication 800-67, 2004. National Institute of Standards and Technology (NIST).

Miroslav Vozňák. Impact of network security on speech quality. Technical report, CESNET, 2008. URL http://www.cesnet.cz/doc/techzpravy/2008/impact-of-network-security-on-speech-quality/impact-of-network-security-on-speech-quality.pdf.

Miroslav Vozňák, Alessandro Rozza, and Antonio Nappa. Performance comparision of secure and insecure VoIP environments, 2008.

Wireshark. Wireshark, 2008. URL http://www.wireshark.org/. Acessado em 08/12/2008.

Maya Yajnik, Sue Moon, Jim Kurose, and Don Towsley. Measurement and modelling of the temporal dependence in packet loss. *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 1:345–352, March 1999. doi: 10.1109/INFCOM.1999.749301.

Frank Yellin. Unable to use -OFB or -CFB ciphers in OpenVPN, May 2009. URL http://www.pubbs.net/200905/openvpn/125967-openvpn-devel-unable-to-use-ofb-or-cfb-ciphers-in-openvpn.html. Acessado em 11/08/2009.

Apêndice A

Apêndices

A.1 *Scripts* de criação dos cenários

Cenário S3

```
#!/bin/bash

INTERFACE=tap0

echo "### Cenario S3 ###"

echo "Aplica regra basica"
#regra certa (jitter de 10ms)
tc qdisc add dev $INTERFACE root handle 1: netem delay 45ms 5ms
   distribution normal loss 0.25% duplicate 1% reorder 0.1%

echo "Limitar banda"
tc qdisc add dev $INTERFACE parent 1:1 handle 10: tbf rate 0.5mbit
   buffer 1600 limit 3000

echo "pfifo"
tc qdisc add dev $INTERFACE parent 10:1 handle 20: pfifo
limit 1000

exit
```

Cenário P2

```
#!/bin/bash

INTERFACE=tap0

function congestiona() {
  echo "CONGESTIONA: adiciona delay de 78ms e perda de pacotes por
     100s"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 78ms 41
     ms distribution normal loss 0.26%
  sleep 100; #100s
}

function descongestiona() {
  echo "DESCONGESTIONA: volta as regras basicas"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 43ms 6
     ms distribution normal loss 0.26%
  sleep 140; #140s
}

echo "### Cenario P2 ###"

echo "Aplica regra basica"
tc qdisc add dev $INTERFACE root handle 1: netem delay 43ms 6ms
   distribution normal loss 0.26%

echo "Limitar banda"
tc qdisc add dev $INTERFACE parent 1:1 handle 10: tbf rate 1.544
   mbit buffer 1600 limit 3000

echo "pfifo"
tc qdisc add dev $INTERFACE parent 10:1 handle 20: pfifo
limit 1000

sleep 20;
```

```
#20s
congestiona
#120s (2min)
descongestiona
#260s (3min20s)

#congestiona
#360s (6min)

echo "Remove␣interface␣do␣TC"
tc qdisc del dev $INTERFACE root

echo "FIM"

exit
```

Cenario P4

```
#!/bin/bash

INTERFACE=tap0

function congestiona() {
  echo "CONGESTIONA:␣adiciona␣delay␣de␣275ms␣e␣perda␣de␣pacotes␣por
     ␣3s"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 275ms
     25ms distribution normal loss 0.25%
  sleep 3; #3s
}

function descongestiona() {
  echo "DESCONGESTIONA:␣volta␣as␣regras␣basicas"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 40ms
     loss 0.25%
  sleep 65; #65s
}
```

```
echo "### Cenario P4 ###"

echo "Aplica regra basica"
tc qdisc add dev $INTERFACE root handle 1: netem delay 40ms loss
   0.25%

echo "Limitar banda"
tc qdisc add dev $INTERFACE parent 1:1 handle 10: tbf rate 1.544
   mbit buffer 1600 limit 3000

echo "pfifo"
tc qdisc add dev $INTERFACE parent 10:1 handle 20:
pfifo limit 1000

sleep 65;
#65s
congestiona
#68s
descongestiona
#133s
congestiona
#136s
descongestiona
#201s
congestiona
#204s
descongestiona
#269s

echo "Remove interface do TC"
tc qdisc del dev $INTERFACE root

echo "FIM"

exit
```

Cenario S2

```
#!/bin/bash

INTERFACE=tap0

function congestiona() {
  echo "CONGESTIONA: adiciona delay de 1000ms e perda de pacotes
     por 3s"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 1000ms
     loss 30% duplicate 0% reorder 0%
  sleep 3; #3s
}

function descongestiona() {
  echo "DESCONGESTIONA: volta as regras basicas"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 60ms 10
     ms loss 2% duplicate 1% reorder 0.1%
  sleep 20; #20s
}

echo "### Cenario S2 ###"

echo "Aplica regra basica"
tc qdisc add dev $INTERFACE root handle 1: netem delay 60ms 10ms
   loss 2% duplicate 1% reorder 1%

echo "Limitar banda"
tc qdisc add dev $INTERFACE parent 1:1 handle 10: tbf rate 0.5mbit
   buffer 1600 limit 3000

echo "pfifo"
tc qdisc add dev $INTERFACE parent 10:1 handle 20: pfifo limit 1000

sleep 20;
#20s
```

```
congestiona
#23s
descongestiona
#43s
congestiona
#46s
descongestiona
#66s
congestiona
#69s
descongestiona
#89s
congestiona
#92s
descongestiona
#112s
congestiona
#115s
descongestiona
#135s (2m15s)

echo "Remove␣interface␣do␣TC"
tc qdisc del dev $INTERFACE root

echo "FIM"

exit
```

Cenario S1

```
#!/bin/bash

INTERFACE=tap0

function congestiona() {
  echo "CONGESTIONA:␣adiciona␣delay␣e␣perda␣de␣pacotes␣por␣3s";
```

```
  tc qdisc change dev $INTERFACE root handle 1: netem delay 1000ms
     loss 30% duplicate 0% reorder 0%
  sleep 3; #3s
}

function descongestiona() {
  echo "DESCONGESTIONA: volta as regras basicas"
  tc qdisc change dev $INTERFACE root handle 1: netem delay 60ms 10
     ms loss 2% duplicate 1% reorder 0.1%
  sleep 20; #20s
}

echo "### Cenario S1 ###"

echo "Aplica regra basica"
tc qdisc add dev $INTERFACE root handle 1: netem delay 60ms 10ms
   loss 2% duplicate 1% reorder 1%

echo "Limitar banda"
tc qdisc add dev $INTERFACE parent 1:1 handle 10: tbf rate 0.1mbit
   buffer 1600 limit 3000

echo "pfifo"
tc qdisc add dev $INTERFACE parent 10:1 handle 20: pfifo limit 1000

sleep 20;
#20s
congestiona
#23s
descongestiona
#43s
congestiona
#46s
descongestiona
#66s
```

```
congestiona
#69s
descongestiona
#89s
congestiona
#92s
descongestiona
#112s
congestiona
#115s
descongestiona
#135s (2m15s)

echo "Remove␣interface␣do␣TC"
tc qdisc del dev $INTERFACE root

echo "FIM"

exit
```

A.2 Arquivos de configuração do OpenVPN

Arquivo de configuração do cliente OpenVPN, no Windows:

```
dev tap
remote 192.168.56.2
tls-client

ns-cert-type server

ca      keys/ca.crt #certificado raiz
cert    keys/vpnclient.crt
key     keys/vpnclient.key

cipher AES-128-CBC
#cipher BF-CBC
```

```
#cipher DES-EDE3-CBC
#cipher none
```

Arquivo de configuração do servidor OpenVPN, no Linux:

```
dev tap
ifconfig 10.1.1.1 255.255.255.0 10.1.1.2 255.255.255.0

tls-server

ca      2.0/keys/ca.crt #certificado raiz
cert    2.0/keys/server.crt
key     2.0/keys/server.key
dh      2.0/keys/dh1024.pem

cipher AES-128-CBC
#cipher BF-CBC
#cipher DES-EDE3-CBC
#cipher none

status      /var/log/openvpn-status.log
log         /var/log/openvpn.log
log-append  /var/log/openvpn.log

verb 3
```