ALGORITMO ADAPTATIVO PARA PREVISÃO E RECUPERAÇÃO DE PERDA DE PACOTES EM APLICAÇÕES MULTIMÍDIAS USANDO CADEIAS DE MARKOV OCULTAS

por

Flávio Pimentel Duarte

UFRJ

Tese submetida para a obtenção do título de

Mestre em Ciências em Engenharia de Sistemas e Computação

ao Programa de Pós-Graduação de Engenharia de Sistemas e Computação

da COPPE/UFRJ

por

Flávio Pimentel Duarte

Julho 2003

ALGORITMO ADAPTATIVO PARA PREVISÃO E RECUPERAÇÃO DE PERDA DE PACOTES EM APLICAÇÕES MULTIMÍDIAS USANDO CADEIAS DE MARKOV OCULTAS

Flávio Pimentel Duarte

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

Prof. Virgílio Augusto Fernandes Almeida, Ph.D.

Prof. Donald F. Towsley, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

JULHO DE 2003

DUARTE, FLÁVIO PIMENTEL

Algoritmo Adaptativo para Previsão e Recuperação de Perda de Pacotes em Aplicações Multimídias usando Cadeias de Markov Ocultas [Rio de Janeiro] 2003

XIV, 80 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2003)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Perdas
2. Previsão
3. Hidden Markov Models

I. COPPE/UFRJ II. Título (Série)

Aos meus familiares com muito carinho.

Agradecimentos

Agradeço ao pessoal do LAND por terem me ouvido nos momentos difícies deste trabalho, em especial ao Allyson, a Ana e o GD que, dentre outras funçoes, foram revisores e co-orientadores.

Gostaria de agradecer também aos amigos Kelvin, Drika e Marluce pelo incentivo que me ajudou a chegar ao fim deste trabalho.

Agradeço também a Carolina pelo apoio e pelo cafezinho que me mantinha alerta em todos os momentos.

ao Fernando pelas dicas e idéias que ajudaram a melhorar este trabalho.

ao Sidney e Raniery do RNP pelo suporte sempre rápido e prestativo.

Agradeço em especial ao Edmundo e a Rosa por terem me aturado durante esses anos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ALGORITMO ADAPTATIVO PARA PREVISÃO E RECUPERAÇÃO DE PERDA DE PACOTES EM APLICAÇÕES MULTIMÍDIAS USANDO CADEIAS DE MARKOV OCULTAS

Flávio Pimentel Duarte

Julho/2003

Orientadores: Edmundo de Souza e Silva
Rosa Maria Meri Leão
Programa: Engenharia de Sistemas e Computação

Forward Error Correction (FEC) funciona adicionando informação redundante ao fluxo sendo transmitido com o objetivo de reconstruir o fluxo original no cliente quando ocorrem perdas de pacotes. Dentre suas vantagens estão o pequeno atraso inserido para recuperar informações comparado com a retransmissão e simplicidade de implementação. Um clara desvantagem é o aumento na taxa de envio devido à redundância adicionada. Como a perda de pacotes é devida ao congestionamento na maioria dos casos, aumentar a quantidade de proteção usando FEC pode ser injusto com fluxos com controle de congestionamento e até pode resultar em mais perdas, o efeito inverso. Logo, algoritmos baseados em FEC devem ser cuidadosamente escolhidos de acordo com as características de erros no caminho entre o emissor e o receptor. Outra questão é desenvolver modelos precisos para o processo de perda de pacotes. Foi desenvolvido um modelo de Markov oculto (HMM) que é capaz de capturar as correlações no processo de perda de pacotes e manter um espaço de estados suficientemente pequeno para uso em um algoritmo adaptativo baseado em FEC. Explorando as possíveis correlações no processo de perda de pacotes, propomos um algoritmo adaptativo e o aplicamos em um conjunto grande de fluxos de voz sobre IP.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ADAPTIVE ALGORITHM FOR LOSS PREDICTING AND RECOVERY IN MULTIMEDIA APPLICATIONS USING HIDDEN MARKOV CHAINS

Flávio Pimentel Duarte

July/2003

Advisors: Edmundo de Souza e Silva
Rosa Maria Meri Leão
Department: Computer and System Engineering

Forward Error Correction (FEC) works by adding redundant information to the stream being transmitted in order to reconstruct the original stream at the receiver when packet losses occur. Among its advantages are the small delays introduced to recover information compared to, for instance retransmission, and simplicity of implementation. A clear disadvantage is the increase in the transmission rate due to the redundancy added. Since packet losses are mostly due to congestion, increasing the amount of protection using FEC may be unfair to flow controlled streams and also may even have an adverse effect on the results. Therefore, the FEC algorithm should be carefully chosen according to the error characteristics of the path between the sender and receiver. Another issue is to develop accurate models of the loss process. We developed an hidden Markov model that was able to capture the correlations in the loss process and yet maintain a sufficiently small state space for use in an adaptive FEC algorithm. Exploiting the possible correlations in the loss process we then propose an adaptive algorithm and apply it to a large amount of voice over IP streams.

Palavras-chave

1. Loss

2. Prediction

3. Hidden Markov Models

Glossário

ACK	:	Pacote de Confirmação(*Acknowlegde*);
ARQ	:	Requisição Automática de Reenvio(*Automatic Repeat reQuest*);
CBR	:	Taxa Constante de Bits (*Constant Bit Rate*);
EM	:	Algoritmo (*Expectation-Maximation*);
FEC	:	(*Forward Error Correction*);
HMM	:	Modelo de Markov Oculto (*Hidden Markov Model*);
IID	:	Independente e Identicamente Distribuído
QoS	:	Qualidade de Serviço (*Quality of Service*);
RTP	:	Protocolo de Tempo-Real (*Real-Time Protocol*);
RTCP	:	Protocolo de Controle de Tempo-Real (*Real-Time Control Protocol*);
UDP	:	User Datagram Protocol
VBR	:	Taxa Variável de Bits (*Variable Bit Rate*);

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A transmissão de dados em tempo real na Internet, como voz e vídeo, vem se popularizando cada vez mais. O aparecimento de equipamentos como *webcams*, de programas como o **FreePhone** [2], **GnomeMeeting** [1] e **NetMeeting** e o aumento da capacidade do canal tanto para o usuário doméstico, quanto para empresas e universidades nos últimos anos são apontados como principais fatores para esta popularização.

Um grande problema no desenvolvimento de aplicativos para transmissão de dados em tempo real é que o meio pelo qual os dados irão trafegar é essencialmente não confiável, em outras palavras, a Internet é uma rede *best-effort*. Parâmetros como banda passante mínima, retardo máximo e taxa de perda máxima de pacotes não são garantidos, comprometendo a transmissão de tráfego de tempo real.

Diferentes classes de aplicações de tempo real tem diferentes requisitos de qualidade. Por exemplo transmissão de voz em tempo real é pouco tolerante ao retardo e mais exigente ainda com relação a taxa de perda pacotes. Aplicações de transmissão de vídeo em tempo real toleram mais a perda de pacotes mas também não toleram o retardo.

1.1 Objetivos e Contribuições

As soluções para o problema de *jitter* são conhecidas, por exemplo [24, 15], e estão já amadurecidas. Entretanto o problema de perda de pacotes ainda encontra grandes desafios. Algumas soluções foram propostas [28, 9, 8].

O objetivo deste trabalho é o desenvolvimento de um algoritmo adaptativo para a previsão e recuperação de perda de pacotes para aplicativos de transmissão de áudio interativos. Este algoritmo utiliza Modelos de Markov Ocultos (HMM) como mecanismo central para a previsão.

Este trabalho está organizado da seguinte forma: no Capítulo 2 será discutido como é feita a transmissão de voz no caso de aplicativos para comunicação iterativa. O Capítulo 3 apresentará um programa de comunicação via voz usando a Internet desenvolvido pelo LAND, bem como a biblioteca de funções para recuperação de pacotes perdidos que foi baseada em [13]. No Capítulo 4 serão apresentadas as teorias sobre HMM que servirão de base para a introdução do algoritmo. No Capítulo 5 é apresentado o algoritmo de previsão e recuperação desenvolvido neste trabalho. O Capítulo 6 discute os resultados obtidos por este algoritmo e o compara com outros dois algoritmos que estão atualmente em uso. Finalmente no Capítulo 7 são feitas algumas considerações sobre o algoritmo proposto e também são discutidos possíveis trabalhos futuros.

Capítulo 2

Antecedentes da Transmissão de Voz

Este capítulo abordará as questões relacionadas à transmissão de fluxos de tempo real na Internet dando ênfase aos fluxos de áudio. Também é discutido neste capítulo a modelagem do processo de perda de pacotes e as técnicas para reparo de mídia devido à perda de pacotes. Na Seção 2.2 são discutidas as características dos modelos de perda de pacotes mais utilizados na literatura [17, 33, 21] e as técnicas de reparo de mídia normalmente utilizadas para fluxos de áudio são discutidas na Seção 2.3.

2.1 Introdução

A Figura 2.1 ilustra o processo de transmissão de áudio utilizando a Internet como meio de propagação. Na figura o emissor envia os pacotes de áudio em seqüência e igualmente espaçados. Ao passar pela rede, os roteadores podem fazer com que o tempo entre pacotes fique variável. Isto acontece devido ao tamanho da fila de pacotes no roteador na hora da chegada do pacote. Outra conseqüência comum é a perda do pacote devido à falta de espaço no roteador para armazenar o pacote recém chegado. Normalmente aplicações de tempo real utilizam o protocolo de transporte UDP para envio dos dados. Este protocolo pode fazer com que os pacotes tomem caminhos diferentes para alcançar o mesmo destino (receptor) resultando em uma terceira conseqüência que é a chegada fora de ordem dos pacotes.

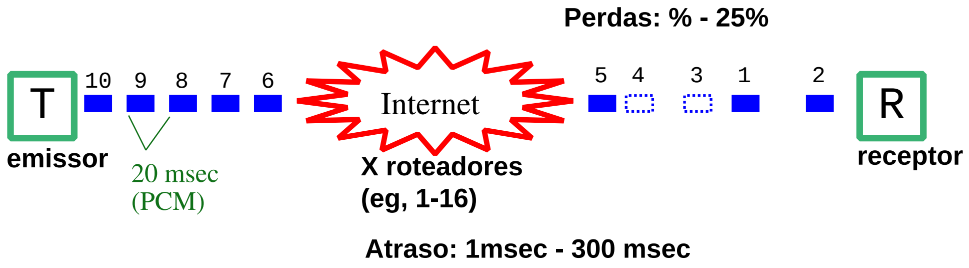

Figura 2.1: Transmissão de um Fluxo de Áudio

Para corrigir o problema do atraso variável entre os pacotes e da ordem de chegada dos mesmos basta que o receptor armazene um certo número deles em área de memória apropriada, conhecida como *playout buffer* [20]. A utilização do *playout buffer* implica em um atraso adicional para se tocar um pacote que é proporcional ao tamanho deste *buffer*. Em [15] é demonstrado que o tamanho deste *buffer* não precisa ser grande, o que é essencial para aplicativos de tempo real.

O número de aplicações de tempo real, bem como a quantidade de usuários que as utilizam, cresceu a ponto de ser criado um protocolo exclusivamente dedicado a elas: o Real-Time Protocol (RTP) [19].

2.2 Modelos de Perda de Pacotes

A perda de pacotes é um dos principais fatores que influenciam a qualidade do sinal recebido. Por essa razão entender e modelar este processo é imperativo para analisar o desempenho dos algoritmos de recuperação de perdas de pacotes.

2.2.1 *Bernoulli*

No modelo de perda de *Bernoulli*, a seqüência de variáveis aleatórias é independente e identicamente distribuída (IID), isto é, a probabilidade de perder o i-ésimo pacote é independente da perda ou não de pacotes anteriores da série temporal e as probabilidade são as mesmas independente de i. Este modelo é caracterizado por um único parâmetro, que é a probabilidade de se perder um pacote.

2.2.2 *Gilbert*

No modelo de *Gilbert*, também conhecido como modelo de Markov de ordem 2, assume-se que o número de pacotes perdidos consecutivos é uma variável aleatória geométrica. Diferentemente do modelo de *Bernoulli*, este modelo é capaz de capturar a dependência entre perdas consecutivas, mas tem um parâmetro adicional. Os dois parâmetros, p e q, são as probabilidades de transição entre os dois estados.

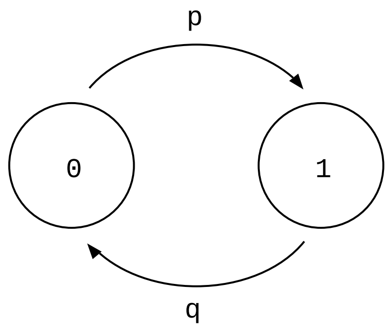

Figura 2.2: Modelo de *Gilbert*

A Figura 2.2 ilustra este modelo. Nesta figura, o estado 0 pode significar sucesso enquanto que o estado 1 significa perda de pacote.

2.2.3 Markov de ordem k

Neste modelo, o processo de perda de pacotes é modelado através de uma cadeia de Markov de ordem k. Sendo assim, o estado atual depende de um certo número de valores prévios do processo que é a ordem do processo. Os modelos de *Bernoulli* e *Gilbert* são casos especiais deste modelo, onde a ordem vale, respectivamente, 1 e 2.

Seja X_t uma série temporal binária de tempo discreto, onde $X = 0$ significa não perda e $X = 1$ significa perda. Um processo $\{X_t\}$ é uma cadeia de Makov de ordem k se a probabilidade condicional

$$P[X_t = x_t | X_{t-h} = x_{t-h}] \tag{2.1}$$

é independente de x_{t-h} para todos os valores de $h > k$.

Este modelo foi utilizado por [33] na tentativa de capturar a dependência temporal das perdas nos arquivos de registro coletados.

2.2.4 Modelos de Markov Ocultos

Num modelo de Markov Oculto (HMM) cada estado pode emitir um subconjunto de símbolos de acordo com alguma distribuição. Os estados e as transições entre estados não são diretamente observáveis, mas podem ser inferidos pela seqüência de símbolos.

Em [21] é proposto o uso de HMM para a modelagem de perda em um canal de comunicação, pois diferentemente do uso de cadeias de Markov tradicionais, as HMM's exibem infinitas dependências do processo em observação, nesse caso a perda de pacotes, mesmo com poucos estados. [21] mostra também que é possível modelar o processo de perdas usando HMM's de no máximo 4 estados, enquanto que [33] teve que usar cadeias de Markov tradicionais de até 42 estados para modelar o mesmo processo.

O uso de HMM não é restrito à modelagem de perdas em canal de comunicação. Na verdade, HMM's foram criadas nos fins da década de 60 e estudadas no início da década de 70. Sua popularidade cresceu com o uso em reconhecimento de fala na década de 80 ([22]) e hoje em dia podem ser usadas virtualmente em qualquer área, incluindo identificação de genes ([29]) e construção automatizada de portais de Internet ([7]).

2.3 Técnicas de Reparo de Mídia

As técnicas de reparo de mídia podem ser divididas em duas grandes classes: as baseadas no emissor e as baseadas no receptor.

As técnicas baseadas no receptor não requerem nenhuma assistência do emissor e basicamente tentam produzir um substituto para o pacote perdido utilizando pacotes anteriores, posteriores ou ambos. Esse conjunto de técnicas é muito utilizado para fluxos de áudio, pois os sinais de áudio exibem uma grande quantidade de auto similaridade de curta duração. As técnicas baseadas no receptor podem ser divididas em três categorias: inserção, interpolação e regeneração.

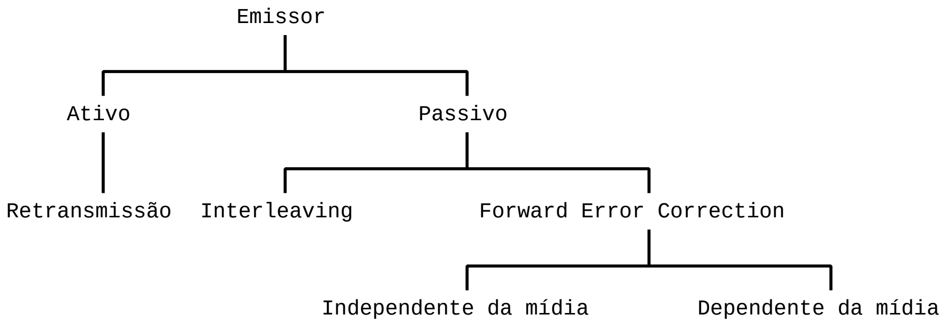

Figura 2.3: Técnicas de Reparo baseadas no emissor

A Figura 2.3 mostra as técnicas de reparo que requerem a participação do emissor para reconstruir pacotes perdidos. Técnicas ativas são aquelas que requerem que uma medida seja tomada somente quando ocorre o problema e técnicas passivas são aquelas baseadas em *loop* aberto [30], isto é, são técnicas preventivas. As técnicas baseadas no emissor podem ser divididas em: retransmissão, *Forward Error Correction* e *Interleaving*.

2.3.1 Inserção

Esta técnica consiste em se inserir uma amostra de áudio no lugar do áudio contido no pacote perdido. Geralmente a amostra de áudio inserida é um ruído de fundo, silêncio ou a última amostra recebida. Esta técnica é simples de se implementar, mas, com exceção da inserção da última amostra recebida, tem um desempenho fraco.

2.3.2 Interpolação

Esta técnica regenera o pacote perdido usando algumas formas de reconhecimento de padrões e interpolação. Esta técnica é geralmente mais difícil de implementar do que a inserção.

2.3.3 Regeneração

Esta técnica utiliza os estados do decodificador para os pacotes anterior e posterior ao pacote perdido para tentar reproduzir o último. Esse processo é caro de implementar mas pode dar bons resultados.

2.3.4 Retransmissão

A técnica de retransmissão, também conhecida por ARQ (*Automatic Repeat reQuest*), consiste no receptor requisitar a transmissão de um determinado pacote assim que o receptor detecta a perda do mesmo. Esta técnica é muito utilizada na transferência de arquivos, onde é exigida total confiabilidade, mas não é muito recomendada para aplicativos de tempo real, pois o atraso inserido pela retransmissão é quase sempre inaceitável. Uma variação desta técnica é o receptor enviar pacotes de confirmação (ACK) para cada pacote recebido. Se o emissor não receber um ACK para um determinado pacote dentro de um intervalo de tempo pré-estabelecido, então o emissor retransmite o mesmo. Esta variação sofre dos mesmos problemas da versão original com relação a transmissão de mídia em tempo real. A vantagem desta técnica é que nenhuma banda é disperdiçada, já que um pacote só é retransmitido se o mesmo for considerado como perdido.

2.3.5 Forward Error Correction

Uma outra técnica para se recuperar pacotes perdidos é conhecida como *Forward Error Correction* (FEC). Nesta técnica além de enviar os pacotes com a mídia, o emissor envia também informações adicionais. Essas informações adicionais são conhecidas como redundâncias e são usadas na reconstrução de pacotes perdidos. Estas redundâncias podem ser dependentes ou não da mídia sendo transmitida. Na técnica FEC dependente de mídia, informações como o tipo da mídia sendo transmitida, a codificação ou ambas são utilizadas para gerar a redundância. A técnica FEC independente da mídia consiste em se usar ou blocos, ou operações algébricas, ou códigos para produzir as redundâncias que serão usadas para a reconstrução dos pacotes perdidos.

A desvantagem em se utilizar pacotes adicionais para reconstruir pacotes perdidos é que normalmente a perda de pacotes é atribuída ao congestionamento da rede e adicionando mais pacotes pode-se agravar o problema resultando em um efeito inverso ao que se queria, que era diminuir a taxa de perda.

O uso de redundância como solução para reparo de mídia de tempo real é tão promissor que o autores do *Real-Time Protocol* (RTP) propuseram modificações para que o mesmo possa utilizá-las [12]. Muito esforço em pesquisa [9], [27], [6], [32], [11] e implementação [26] vem sendo gasto no uso de redundância como principal mecanismo para reconstruir pacotes perdidos.

FEC Dependente de Mídia

Um exemplo de FEC dependente da mídia é o algoritmo proposto em [10]. Neste algoritmo um pacote além de carregar a amostra de áudio original, ele também carrega uma cópia da amostra de áudio anterior possivelmente utilizando uma codificação de mais baixa qualidade. A Figura 2.4 ilustra esta abordagem. Os pacotes normais estão representados por retângulos normais, enquanto que as redundâncias estão representadas por retângulos sombreados. Podemos ver na figura que o pacote 2 carrega o pacote 1, o pacote 3 carrega o pacote 2 e assim por diante. Se o receptor não receber o pacote 2, por exemplo, o receptor poderá reconstruir o pacote 2 com a redundância contida no pacote 3. Por outro lado, se o receptor também perder o pacote 3, ele poderá recuperar este último pacote quando o pacote 4 chegar, mas ele não poderá recuperar o pacote 2 usando o pacote 3 recém recuperado, pois a redundância do pacote 3 não é recuperada. Uma desvantagem desta técnica é que ela só recupera o último pacote de uma rajada de pacotes perdidos. Uma solução para corrigir esta desvantagem seria o pacote n carregar uma cópia do pacote $n-1$ e do pacote $n-2$. O problema é o custo adicional em termos de banda para este tipo de solução.

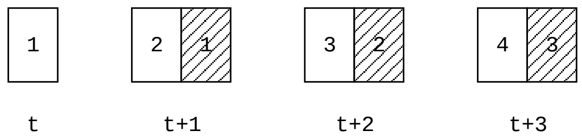

Figura 2.4: Esquema Simples de Redundância Independente de Mídia

Como mencionado anteriormente, adicionar pacotes extras indiscriminadamente pode agravar o problema da perda de pacotes, por isso [8] propõe uma alteração no algoritmo apresentado acima para torná-lo dinâmico. Nesta abordagem são usados os pacotes de controle do RTP (RTCP) para enviar informações sobre o processo de perdas de pacotes que está sendo observada pelo receptor. Este processo pode ser modelado no receptor como uma variável aleatória *Bernoulli* ou por um modelo de *Gilbert*. Estes pacotes são enviados com intervalo de 5 segundos entre eles. O emissor utiliza o valor recebido para consultar uma tabela. Esta tabela relaciona o novo esquema de redundância a ser utilizado de acordo com a taxa de perda observada pelo receptor antes do processo de recuperação dos pacotes perdidos. Um dos problemas desta abordagem é que ela só usa os últimos 5 segundos de áudio e por isso não consegue capturar a correlação entre perdas para *lags* maiores que 5 segundos. Além disso, o algoritmo supõe que as perdas podem ser modeladas por um processo de *Bernoulli* ou de *Gilbert*, o que nem sempre é verdade.

FEC Independente de Mídia

Um exemplo de FEC independente da mídia é o algoritmo proposto em [13]. Neste algoritmo, o fluxo de áudio é dividido em grupos, chamados de janelas e os pacotes das janelas são subdivididos em conjuntos disjuntos, chamados cadeias. A redundância de cada cadeia é calculada através de um operação XOR utilizando todos os pacotes da cadeia. A Figura 3.2 ilustra o conceito de janela e cadeia. A grande vantagem desta técnica é a possibilidade de recuperar mais de um pacote de uma rajada de perda. Uma desvantagem desta técnica é o atraso necessário para se recuperar um pacote perdido. Para recuperar o pacote perdido é necessário que todos os pacotes pertencentes a sua cadeia tenham chegado, bem como a redundância que o protege.

2.3.6 Interleaving

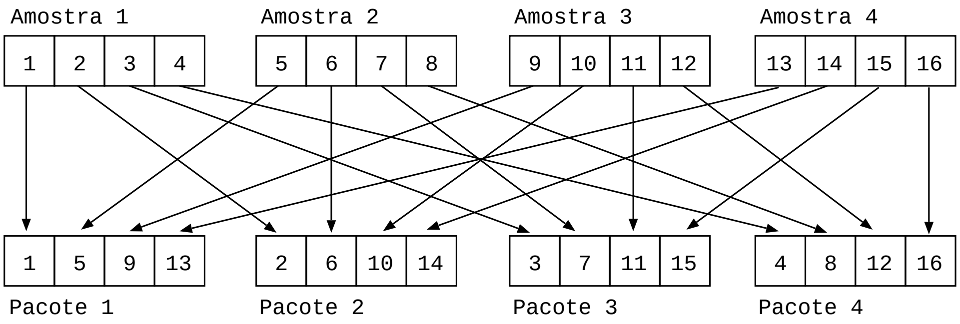

Figura 2.5: Interleaving

Apesar de a técnica de *Interleaving* não poder recuperar um pacote, ela é usada para reduzir os efeitos da perda de pacotes. Antes de transmitir os dados, o emissor mistura o conteúdo de diversos pacotes, fazendo com que, no evento da perda de um pacote, o resultado seja pequenas perdas em diversos pacotes. A Figura 2.5 ilustra este processo. Se o pacote 3 for perdido, como resultado ocorreram pequenas falhas nas amostras 1, 2 e 4 e não um vazio do tamanho de uma amostra.

Capítulo 3

VivaVoz

Neste capítulo será apresentado o programa de comunicações de áudio utilizando a Internet e a biblioteca de recuperação de perdas de pacotes desenvolvidos pelo LAND.

3.1 VivaVoz

Com o objetivo de medir e analisar o tráfego de voz, em 1997 foi desenvolvida uma ferramenta de comunicação com uma funcionalidade e qualidade de áudio semelhante ao telefone ([14]). Esta ferramenta foi chamada de **VivaVoz** e desde então vem sendo usada como ambiente de estudos de algoritmos relacionados a voz.

O programa foi originalmente escrito utilizando linguagem C e não dispunha de uma interface amigável com o usuário. Diversas alterações foram feitas no projeto original, muitas para permitir a utilização da linguagem TCL/TK para fazer a interface com o usuário. Em 1999 o programa foi totalmente reescrito para facilitar a comunicação com a interface com o usuário. Atualmente existe duas versões de interface para o programa **VivaVoz**: uma interface gráfica feita em Qt e uma texto feita com o ncurses.

O programa **VivaVoz** mostra as estatísticas observadas nos dois sentidos do fluxo de áudio. As informações relativas às estatísticas locais, isto é, que são coletadas

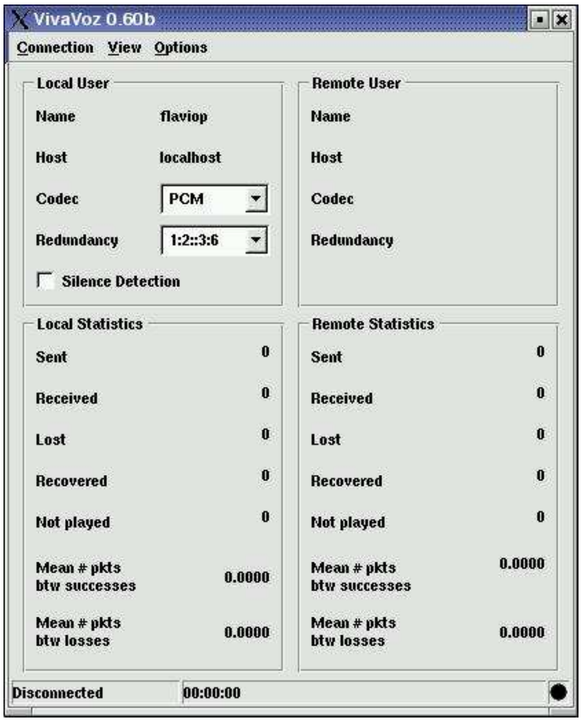

Figura 3.1: Interface do VivaVoz

pelo aplicativo local, são atualizadas a cada segundo. Como é preciso enviar um pacote do aplicativo remoto para o local para obter as estatísticas remotas, elas são atualizadas uma vez por minuto.

Além das estatísticas mostradas na tela, o programa também pode gerar um arquivo de registro contendo o número de sequência, o tamanho e a hora de chegada dos pacotes. Com essas informações é possível calcular, por exemplo, o *jitter*, a taxa de perda, etc. O formato do arquivo de registro é compatível com a ferramenta **Tangram-II** também desenvolvida pelo LAND.

3.2 Biblioteca de Redundância

Uma das modificações sofridas pelo programa **VivaVoz** foi a inserção de uma biblioteca de funções para recuperação de pacotes. Esta biblioteca recupera pacotes utilizando a técnica *Forward Error Correction (FEC)* independente de mídia e foi baseada em [13]. Optou-se por implementar uma biblioteca para facilitar a utilização deste conjunto de rotinas para recuperação de pacotes em outros aplicativos.

É importante ressaltar que apesar desta biblioteca ter sido desenvolvida para ser utilizada no programa **VivaVoz**, não existe motivo que a impeça de ser usada por aplicativos que usam outro tipo de mídia. A biblioteca poderia ser usada, por exemplo, por um aplicativo de transmissão de vídeo sem que houvesse nenhuma alteração.

3.2.1 Conceitos

A terminologia adotada em [13] foi utilizada na implementação da biblioteca e será reproduzida nesta seção. A biblioteca utiliza a operação OU-EXCLUSIVO para proteger grupos de pacotes e trabalha com o conceito de janela e cadeia. Uma cadeia designa os pacotes que são protegidos pela mesma redundância e a janela indica os pacotes que fazem parte das cadeias. Os pacotes protegidos por uma cadeia estão intercalados com os pacotes das outras cadeias. Esses conceitos podem ser visto na Figura 3.2.

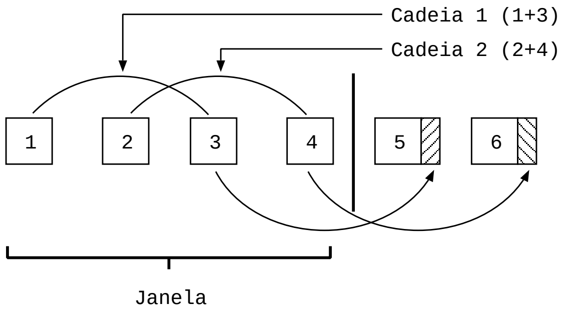

Figura 3.2: Conceito de Janela e Cadeia

O número de pacotes protegidos por uma cadeia é igual ao tamanho da janela divido pelo número de cadeias. Por esta razão o tamanho da janela deve ser um múltiplo do número de cadeias. Um esquema de redundância é designado pelo número de cadeias e pelo tamanho da janela, por exemplo 3:6 refere-se ao esquema de redundância com janela de tamanho 6 e três cadeias utilizando dois pacotes cada uma para formar a redundância.

3.2.2 Funcionamento

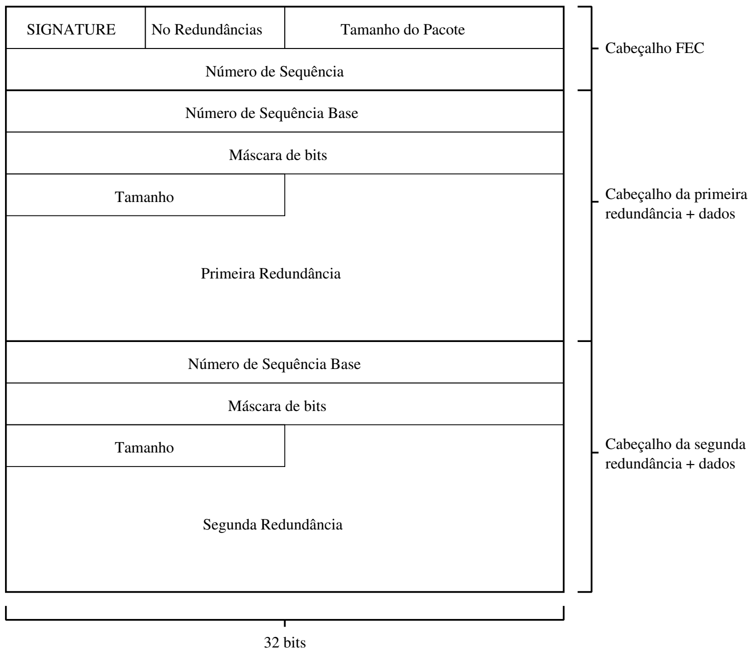

Figura 3.3: Estrutura de um Pacote com Redundâncias

A Figura 3.3 mostra a estrutura de um pacote com duas redundâncias. Os primeiros 8 bytes formam o cabeçalho de um pacote contendo redundâncias. O primeiro byte contém uma assinatura que permite a rotina de reconstrução de pacotes identificar os pacotes corretos. O segundo byte diz quantas redundâncias o pacote inteiro contém. O campo "Tamanho do Pacote" diz quantos bytes contém o pacote contando o tamanho do cabeçalho FEC e as redundâncias com os seus respectivos cabeçalhos. O último campo deste cabeçalho armazena o número de sequência dos pacotes com redundâncias. Os campos "Número de Sequência Base", "Máscara de Bits" e "Tamanho" compõem o cabeçalho de uma redundância. O campo "Tamanho" indica o tamanho da redundância sem contar o cabeçalho. Os campos "Número de Sequência Base" e "Máscara de Bits" indicam os pacotes que foram usados para construir esta redundância e que, consequentemente, podem ser recuperados por ela. Os próximos bytes contém a redundância propriamente dita, seguida do cabeçalho da segunda redundância e assim por diante.

Além das informações contidas nos pacotes de redundância, a única informação extra que a rotina de recuperação de pacotes precisa para reconstruir um pacote perdido, são os pacotes de áudio originais que chegaram com sucesso.

3.2.3 Cadeias Zombies

A biblioteca desenvolvida permite a mudança de esquemas de redundância em tempo de execução. Um problema que ocorre nesta mudança é que se as cadeias atuais forem substituídas pelas novas alguns pacotes não estarão cobertos pela redundância. Isto ocorre porque o esquema novo precisa de pelo menos uma janela para produzir a primeira redundância.

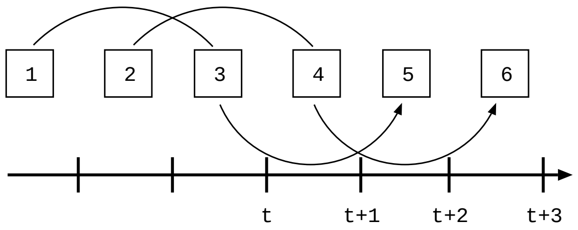

Figura 3.4: Problema da mudança de esquema

A Figura 3.4 ilustra o problema para o esquema 2:4, onde são usadas 2 cadeias para uma janela de tamanho 4. Neste exemplo o pacote número 5 carrega a redundância produzida pelos pacotes 1 e 3, enquanto que o pacote número 6 carrega a redundância que protege os pacotes 2 e 4. Se o esquema for alterado no instante de tempo t e as cadeias forem apagadas, as informações necessárias para a construção da redundância de 2 e 4 serão perdidas, fazendo com que o pacote número 6 não carregue nenhuma redundância. No caso da redundância obtida com os pacotes 1 e 3 o desperdício é ainda maior, pois a redundância já foi calculada e só esperava a requisição da aplicação para ser liberada.

A solução para o problema acima é não apagar a cadeia assim que ocorre uma mudança de esquema. Quando ocorre uma mudança de esquema as cadeias atuais são marcadas como “zombies” e só irão efetivamente ser apagadas depois que a redundância produzida por elas for requisitada pela aplicação. No exemplo da Figura 3.4, no instante de tempo t as duas cadeia serão marcadas como “zombies”. No instante de tempo $t + 2$ a cadeia formada pelos pacotes 1 e 3 será apagada da memória enquanto que a cadeia formada pelos pacotes 2 e 4 só será apagada em $t + 3$.

O termo "zombie" foi retirado através de analogia com os processos UNIX, que são processos que já terminaram o seu trabalho, mas estão a espera que o pai recupere o seu valor de término.

Capítulo 4

Cadeias de Markov Ocultas

Neste capítulo serão apresentados a notação e os conceitos sobre HMM. A notação apresentada nesta seção será usada pelas seções subseqüentes. No final deste capítulo é abordado o programa desenvolvido usando os conceitos apresentados. Este programa serviu de base de testes tanto para a pesquisa apresentada aqui bem como para a pesquisa feita em [31]. A notação usada aqui é a mesma utilizada em [21].

4.1 Teoria Básica

O artigo [22] define HMM como sendo um processo estocástico duplamente embutido onde existe um processo estocástico implícito que não é observável, mas que pode somente ser "visto" através de um outro conjunto de processos estocásticos que produzem a seqüência de observações.

Por exemplo, vamos considerar um cassino desonesto que usa dois tipos de dados para um determinado jogo. Um dado é normal, ou seja, a probabilidade de sair qualquer uma de suas 6 faces é $1/6$. O outro dado é viciado de modo que a face com 6 saia 50% das vezes. Um jogador não sabe que o cassino está usando mais de um dado e vai apenas ver os resultados dos dados, ou, em outras palavras, a seqüência de observações deste experimento. Como o jogador não vê a troca dos dados, ele pode apenas inferir sobre esse processo através da seqüência de observações, portanto este processo é oculto para o jogador. A Figura 4.1 mostra o diagrama de estados deste modelo de Markov Oculto.

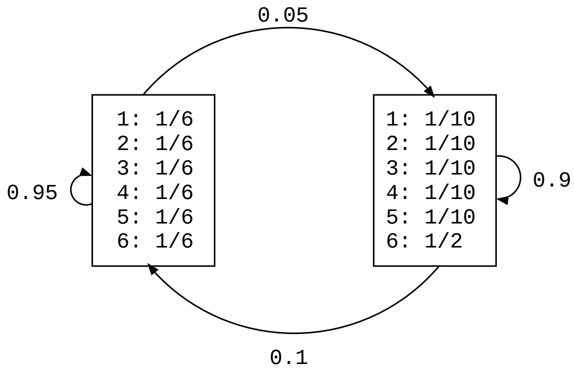

Figura 4.1: Cassino Desonesto

4.1.1 Elementos de um HMM

Formalmente um HMM é caracterizada por cinco elementos:

1. **N: número de estados do modelo**

   Apesar de em geral os estados serem ocultos, em muitas aplicações existe um sentido físico em cada estado. No exemplo citado acima, o analista pode criar um modelo onde cada estado representa um dado diferente com diferentes probabilidades de emissão de cada face. Os estados individuais serão denotados por $S = \{S_1, S_2, \cdots, S_N\}$, onde N é o número total de estados, e o estado no instante de tempo t como q_t.

2. **M: número de símbolos distintos do modelo**

   Também conhecido como tamanho do alfabeto. Os símbolos são as saídas físicas do sistema sendo modelado. No exemplo do cassino desonesto, os símbolos são as faces dos dados. Os símbolos serão denotados por $V = \{v_1, v_2, \cdots, v_M\}$.

3. **A: matriz de probabilidade de transição**

   Indica a probabilidade do sistema sair de um estado para outro em um passo. Ainda no exemplo do cassino, a matriz de probabilidade de transição indica a probabilidade de o cassino trocar do dado normal para o dado viciado e vice-versa. A probabilidade de transição do estado i para o estado j em um passo é $a_{ij} = P[q_{t+1} = S_j | q_t = S_i], \qquad 1 \leq i, j \leq N$.

4. **B: matriz de probabilidade de emissão**

   A probabilidade de emissão do símbolo k dado que o sistema está no estado j é $b_j(k) = P[v_k \text{ no tempo t } | q_t = S_j]$.

5. **π: distribuição do estado inicial**

   A probabilidade de o sistema começar no estado i é $\pi_i = P[q_1 = S_i], \qquad 1 \leq i \leq N$.

Para simplificar a notação, a definição $\lambda = (A, B, \pi)$ será usada.

4.1.2 Treinamento de uma HMM

Seja $O = O_1 O_2 \cdots O_T$ uma seqüência de observações de um determinado processo. O problema mais difícil relacionado com HMM's é como ajustar os parâmetros $\lambda = (A, B, \pi)$ de modo a maximizar $P(O|\lambda)$. Para resolver este problema precisamos primeiro encontrar uma maneira eficiente de calcular $P(O|\lambda)$. A maneira mais simples de se calcular $P(O|\lambda)$ seria calcular todas as possibilidades da seqüência $O = O_1 O_2 \cdots O_T$ por todos os possíveis caminhos de tamanho T. A probabilidade de O dado o modelo é obtida através da soma das probabilidades conjuntas sobre todas as possíveis seqüências de estados.

$$\begin{aligned} P(O|\lambda) &= \sum_{todos Q} P(O|Q,\lambda) P(Q|\lambda) \\ &= \sum_{q_1, q_2, \cdots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(Q_T), \end{aligned} \tag{4.1}$$

onde $Q = q_1 q_2 \cdots q_T$ é uma determinada seqüência de estados.

Este cálculo não é factível, mesmo para pequenos valores de N e T. Por exemplo, para $N = 5$ estados e $T = 100$ observações existem cerca de $2.100.5^{100} \approx 10^{72}$

operações. Existe um procedimento mais eficiente para calcular $P(O|\lambda)$ e esse procedimento é conhecido como Procedimento *Forward-Backward*. O procedimento *Forward-Backward* não só calcula $P(O|\lambda)$ como também ajuda a calcular a nova matriz de transições($\bar{A}$), os novos vetores de emissões($\bar{B}$) e o novo valor do vetor de distribuição dos estados iniciais($\bar{\pi}$).

Procedimento *Forward-Backward*

Seja a variável *forward* $\alpha_t(i)$ definida como:

$$\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i|\lambda) \tag{4.2}$$

isto é, a probabilidade de a seqüência parcial de observações, $O_1 O_2 \cdots O_t$ e o estado S_i no tempo t dado o modelo λ. O valor de $\alpha_t(i)$ pode ser calculado com o seguinte algoritmo:

1. Inicialização:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N \tag{4.3}$$

2. Indução:

$$\alpha_{t+1}(j) = \left[\sum_{j+1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \qquad \begin{array}{l} 1 \leq t \leq T-1 \\ 1 \leq j \leq N \end{array} \tag{4.4}$$

3. Término:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.5}$$

A Figura 4.2 ilustra como o estado S_j pode ser alcançado no instante de tempo $t+1$ a partir dos N possíveis estados $S_i, 1 \leq i \leq N$ no instante de tempo t.

Considere agora a variável *backward* $\beta_t(i)$ definida como:

$$\beta_t(i) = P(O_{t+1} O_{t+2} \cdots O_T | q_t = S_i, \lambda) \tag{4.6}$$

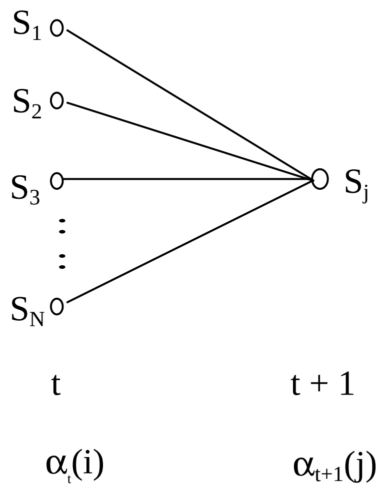

Figura 4.2: Procedimento Forward

isto é, a probabilidade de a seqüência de observações parcial de $t+1$ até o instante final T dado o estado S_i no instante de tempo t e o modelo λ. A variável $\beta_t(i)$ pode ser calculada usando o seguinte procedimento:

1. Inicialização:

$$\beta_T(i) = 1, \qquad 1 \leq i \leq N \tag{4.7}$$

2. Indução:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \qquad t = T-1, T-2, \cdots, 1 \qquad 1 \leq i \leq N \tag{4.8}$$

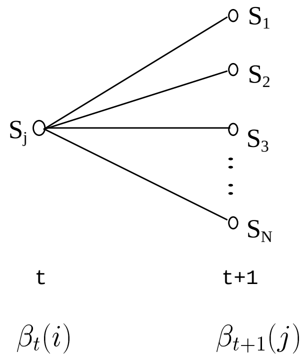

Figura 4.3: Procedimento Backward

A Figura 4.3 mostra que para chegar ao estado S_i no instante de tempo t tem que se considerar a seqüência de observações a partir do tempo $t+1$. É necessário levar em conta todos os possíveis estados S_j no instante de tempo $t+1$ contando a transição de S_i para S_j e a observação O_{t+1} no estado j e então contar a seqüência de observações parcial a partir do estado j.

Cálculo das Transições

Seja a variável $\xi_t(i,j)$ a probabilidade de transição do estado i no instante de tempo t para o estado j no instante de tempo $t+1$ dado o modelo λ e a seqüência de observações O.

$$\begin{aligned}
\xi_t(i,j) &= P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}
\end{aligned} \tag{4.9}$$

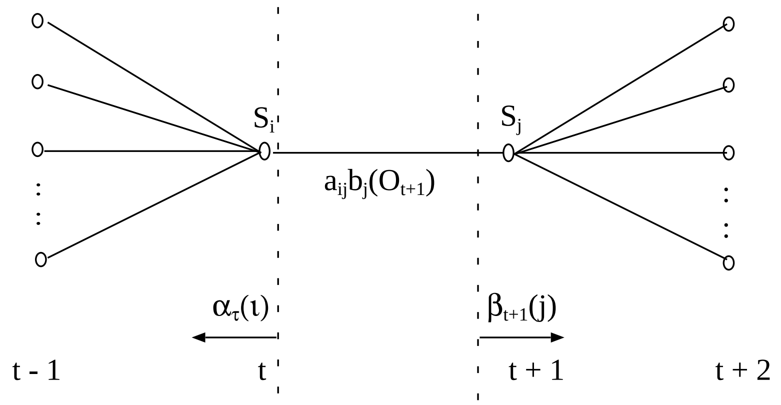

Figura 4.4: Cálculo da transição

A Figura 4.4 ilustra graficamente como funciona o cálculo de uma transição usando as variáveis *forward* e *backward*. Para se ter uma transição do estado i no instante de tempo t para o estado j no instante de tempo $t+1$ com a emissão do símbolo O_{t+1} no estado j, é necessário calcular a probabilidade de se estar no estado i no instante de tempo t, representado pelo termo $\alpha_t(i)$. A seguir, tem que se calcular a probabilidade de transição de i para j, que é exatamente o termo a_{ij}

e finalmente a probabilidade do estado j emitir o símbolo O_{t+1}, representada na fórmula pelo termo $b_j(O_{t+1})$. O termo $\beta_{t+1}(j)$ significa a probabilidade de obter a seqüência parcial a partir do instante de tempo $t+1$ até o final sabendo-se que neste instante de tempo o sistema está no estado j dado o modelo e a seqüência de observações.

Se o termo $\xi_t(i,j)$ for somado para todos os intervalos de tempo t, obtém-se o número esperado de transições do estado i para o estado j.

$$\sum_{t=1}^{T} \xi_t(i,j) = \text{Número esperado de transições de } i \text{ para } j \tag{4.10}$$

Cálculo do Estado Atual

Como base na equação apresentada acima, pode-se definir a variável $\gamma_t(i)$ como sendo a probabilidade de se estar no estado i no instante de tempo t dado o modelo λ e a seqüência de observações O.

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(ij) \tag{4.11}$$

Com $\gamma_t(i)$ pode-se calcular o número esperado de visitas ao estado i. Para isso, basta somar $\gamma_t(i)$ para todos os instantes de tempo, isto é,

$$\sum_{t=1}^{T} \gamma_t(i) = \text{Número esperado de visitas a } S_i \tag{4.12}$$

Seqüência de estados - Algoritmo de *Viterbi*

Em alguns casos é útil saber a seqüência de estados de uma cadeia de Markov Oculta que melhor representa uma determinada seqüência de observações. Uma maneira de se obter a seqüência de estados é escolher os estados q_t que são individualmente mais prováveis. O resultado final usando esta metodologia maximiza o número esperado de estados individualmente corretos. Conforme definido acima,

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) \tag{4.13}$$

$\gamma_t(i)$ pode ser representado em função das variáveis forward-backward:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{4.14}$$

Usando $\gamma_t(i)$ pode-se calcular o estado q_t individualmente mais provável no instante de tempo t da seguinte forma:

$$q_t = argmax_{1 \leq i \leq N}[\gamma_t(i)], \qquad 1 \leq t \leq T \tag{4.15}$$

Apesar de a Equação 4.15 maximizar o número esperado de estados corretos, não garante que a seqüência de estados produzida seja válida, pois esta equação simplesmente determina o estado mais provável para cada instante de tempo sem levar em consideração a probabilidade da seqüência de estados.

Outra maneira é maximizar $P(Q|O, \lambda)$, ou seja, encontrar a seqüência de estados mais provável, o que é equivalente a maximizar $P(Q, O|\lambda)$. O algoritmo de *Viterbi* é uma técnica baseada em métodos de programação dinâmica para encontrar a seqüência de estados mais provável.

Para encontrar a seqüência mais provável de estados para uma dada seqüência, $Q = q_1 q_2 \cdots q_t$, de observações $O = O_1, O_2, \cdots O_T$ é preciso atribuir uma pontuação para a seqüência:

$$\delta_t(i) = max_{q_1, q_2, \cdots, q_{t-1}} P[q_1 q_2 \cdots q_t = i, O_1 O_2 \cdots O_t|\lambda] \tag{4.16}$$

isto é, $\delta_t(i)$ é a mais alta pontuação (mais alta probabilidade) para uma seqüência de estados, no tempo t para as primeiras t observações e que termina no estado S_i. Por indução:

$$\delta_{t+1}(j) = [max_i \delta_t(i) a_{ij}] b_j(O_{t+1}) \tag{4.17}$$

É necessário armazenar o argumento que maximiza a Equação 4.17 para cada t e j. Esse argumento é armazenado no vetor $\psi_t(j)$.

O procedimento completo para encontrar a seqüência de estados mais provável utilizando os conceitos acima é:

1. Inicialização:

$$\delta_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N \tag{4.18}$$

$$\psi_1(i) = 0 \tag{4.19}$$

2. Recursão:

$$\delta_t(j) = max_{1 \leq i \leq N}[\delta_{t-1}(i) a_{ij}] b_j(O_t), \qquad \begin{matrix} 2 \leq t \leq T \\ 1 \leq j \leq N \end{matrix} \tag{4.20}$$

$$\psi_t(j) = argmax_{1 \leq i \leq N}[\delta_{t-1}(i) a_{ij}] \qquad \begin{matrix} 2 \leq t \leq T \\ 1 \leq j \leq N \end{matrix} \tag{4.21}$$

3. Término:

$$P^* = max_{1 \leq i \leq N}[\delta_T(i)] \tag{4.22}$$

$$q_T^* = argmax_{1 \leq i \leq N}[\delta_T(i)] \tag{4.23}$$

4. Seqüência de estados:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \cdots, 1. \tag{4.24}$$

Treinamento Iterativo

Tendo-se definido uma metodologia para o cálculo de cada componente individual de uma cadeia de Markov Oculta, é possível elaborar um procedimento iterativo para treinar esta cadeia. O algoritmo apresentado aqui é o algoritmo *Expectation-Maximation* (EM) e é baseado no algoritmo de *Baum-Welches* [22].

A cada iteração do algoritmo a probabilidade do modelo dada a seqüência de observações é incrementada. Isto garante a convergência do modelo para um máximo. O maior problema desta técnica é que este máximo é local e não global.

Esse procedimento está representado graficamente na Figura 4.5.

O primeiro passo é escolher os parâmetros π, A e B do modelo. Esta escolha é feita de modo arbitrário e pode levar ao máximo global ou ao máximo local longe do máximo global. O segundo passo é o cálculo das variáveis *forward* (α) e *backward* (β). Com esses valores pode-se ir para o próximo passo que é o cálculo do número esperado de visitas ao estado i e o número esperado de transições de i para j. Os parâmetros do novo modelo ($\bar{\lambda}$) são calculados pelas fórmulas abaixo:

$$\begin{aligned}
\bar{\pi}_i &= \text{número esperado de visitas ao estado } S_i \text{ no instante de tempo } (t=1) \\
&= \gamma_1(t) \\
\bar{a}_{ij} &= \frac{\text{número esperado de transições do estado } S_i \text{ para o estado } S_j}{\text{número esperado de transições de } S_i} \\
&= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\
\bar{b}_j(k) &= \frac{\text{número esperado de vezes no estado } S_j \text{ gerando o símbolo } v_k}{\text{número esperado de vezes no estado } S_j} \\
&= \frac{\sum_{\substack{t=1 \\ t.q. O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}
\end{aligned} \tag{4.25}$$

De posse dos novos parâmetros, a probabilidade do novo modelo dada a seqüência de observações ($P(O|\bar{\lambda})$) é novamente calculada. A probabilidade do novo modelo é comparada com a probabilidade do modelo antigo e se o ganho relativo ($P(O|\bar{\lambda}) - P(O|\lambda)$) for pequeno pode-se parar. O algoritmo pode parar também pelo número de iterações, pois o tempo para atingir um ganho relativamente pequeno pode ser relativamente grande principalmente se este algoritmo estiver sendo usado em um aplicativo de tempo real.

As fórmulas de reestimação apresentadas em 4.25 podem ser derivadas direta-

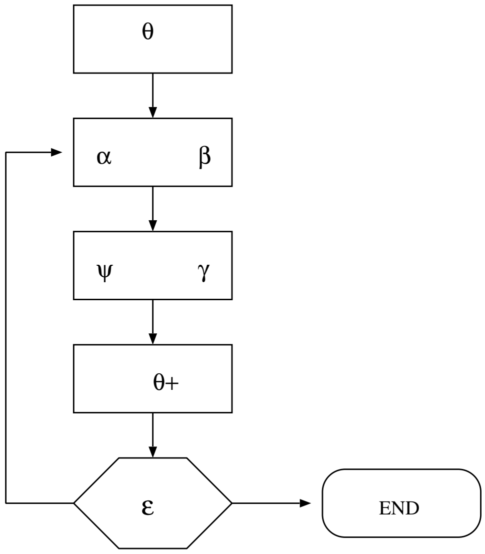

Figura 4.5: Treinamento do cadeia

mente pela maximização da função auxiliar de *Baum*

$$Q(\lambda, \bar{\lambda}) = \sum_{Q} P(Q|O, \lambda) log[P(O, Q|\bar{\lambda})] \tag{4.26}$$

em $\bar{\lambda}$.

As fórmulas de reestimação podem ser interpretadas como uma implementação do algoritmo EM no qual o passo E (*expectation*) é o cálculo da função auxiliar $Q(\lambda, \bar{\lambda})$, e o passo M (*modification*) é a maximização sobre $\bar{\lambda}$. Portanto as equações de reestimação de *Baum-Welch* são essencialmente idênticas aos passos EM para este problema em particular.

A Influência dos Valores Iniciais

Como foi mencionado acima, a escolha dos parâmetros A, B e π influenciam no treinamento da cadeia. Uma má escolha destes parâmetros pode levar o modelo ao um máximo local bem distante do máximo global. Uma pergunta crucial é como escolher os parâmetros de modo que o máximo local é também o máximo global. [22] indica que uma escolha aleatória dos parâmetros de A e π é adequada para a maioria dos casos, mas que o parâmetro B tem uma influência decisiva para modelos com alfabeto discreto.

4.1.3 Scaling

Pela definição de α apresentada em 4.2, pode-se observar que o mesmo é constituído pela soma de um grande número de termos, cada termo com a forma: $(\prod_{s=1}^{t-1} a_{q_s q_{s+1}} \prod_{s=1}^{t} b_{q_s}(O_s))$

Como os termos a e b são menores que 1, os termos de $\alpha_t(i)$ tendem a zero exponencialmente conforme se aumenta o número de observações t. Para valores grandes de t (100 ou mais), o cálculo de $\alpha_t(i)$ começa a exceder a precisão numérica computacional. Por isso é necessária a incorporação de um procedimento capaz de re-escalar os valores de $\alpha_t(i)$ e $\beta_t(i)$.

A solução mais usual é multiplicar $\alpha_t(i)$ por um fator de escala que é independente de i, e então só depende de t. A vantagem desta solução é que ela mantém os termos $\alpha_t(i)$ dentro do limite numérico do computador para $1 \leq t \leq T$. Devido a semelhança de magnitude dos termos α e β, podemos aplicar o mesmo fator de escala para os dois termos.

Para melhor entender o procedimento de *Scaling*, considere a Fórmula 4.25 reescrita usando diretamente os termos em função das variáveis α e β:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{4.27}$$

Considere o cálculo de α. Para cada t, é preciso primeiro calcular $\alpha_t(i)$ de acordo com a indução apresentada em 4.4, e então multiplicá-lo pelo fator de escala c_t, onde:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{4.28}$$

Portanto, para um valor fixo de t, é primeiro calculado

$$\alpha_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t) \tag{4.29}$$

Então $\hat{\alpha}_t(i)$ é calculado por

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)} \tag{4.30}$$

Por indução $\hat{\alpha}_{t-1}(j)$ pode ser reescrito como

$$\hat{\alpha}_t(j) = (\prod_{\tau=1}^{t-1} c_\tau) \alpha_{t-1}(j) \tag{4.31}$$

Portanto $\hat{\alpha}_t(i)$ pode ser reescrito como

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \alpha_{t-1}(j) (\prod_{\tau=1}^{t-1} c_\tau) a_{ij} b_j(O_t)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_{t-1}(j) (\prod_{\tau=1}^{t-1} c_\tau) a_{ij} b_j(O_t)} = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \tag{4.32}$$

isto é, cada $\alpha_t(i)$ é efetivamente escalado pela soma sobre todos os estados de $\alpha_t(i)$.

De maneira análoga pode-se calcular $\hat{\beta}_t(i)$ em função de $\beta_t(i)$

$$\hat{\beta}_t(i) = c_t \beta_t(i) \tag{4.33}$$

4.2 Ferramentas HMM

Apesar de já existirem aplicativos, tanto comerciais quanto de uso público, que permitem o tratamento das equações acima ([25], [16]) foi desenvolvido um aplicativo capaz de treinar uma cadeia de Markov Oculta a partir de uma seqüência de observações usando a teoria apresentada neste capítulo. Essa escolha foi feita para facilitar a inserção das rotinas nos aplicativos **VivaVoz** [5] e **Tangram-II** [4].

O programa foi totalmente desenvolvido utilizando linguagem C e foi batizado de **hmm**. As rotinas foram implementadas de modo a serem facilmente transportadas para outro aplicativo, em especial o **VivaVoz** [5]. O programa **hmm** utiliza o módulo de visualização de matriz desenvolvido para a ferramenta **Tangram-II** [4] para mostrar tanto a matriz de transição quanto a matriz de emissão. Além disso, o programa permite que o usuário salve o modelo atual usando o formato OBJ usado pelo programa **tgif** que é a base para a confecção de modelos na ferramenta **Tangram-II**. A seguir é mostrada a tela com a lista dos comandos do programa **hmm**.

```
$ hmm
Hidden Markov Models environment
version 0.2b

HMM> ?
add      - add and normalize matrix
guess    - estimate observation sequence
exit     - exit program
load     - load model parameters and/or observations
train    - perform parameter reestimation
viterbi  - find the most probable state path
set      - set/show internal variables
print    - print model parameters, observations or trace
show     - show model parameters and/or observations
history  - list most recent commands
?        - shows this help screen
HMM>
```

Apesar deste programa ter sido inicialmente desenvolvido para esta pesquisa e com o objetivo de ser incorporado na ferramenta **VivaVoz** [5], ele foi utilizado na área de medicina tendo como objetivo determinar a influência de determinados genes sobre fatores como hipertensão, obesidade, etc. Os resultados deste experimento estão relatados em [31]. Para atender as necessidades deste estudo, novos comandos foram desenvolvidos.

Capítulo 5

Algoritmo de Previsão e Recuperação

Neste capítulo será descrito em detalhes o algoritmo de previsão e recuperação de pacotes de áudio. O algoritmo é baseado em Cadeias de Markov Ocultas, cuja teoria foi apresentada no Capítulo 4, e é composto por seis fases. O algoritmo com suas fases estão ilustrados na Figura 5.5.

O objetivo do algoritmo é prever o processo de perda de pacotes de áudio baseado num passado recente. Os testes realizados neste capítulo e no capítulo 6 utilizaram os arquivos de registro produzidos por um aplicativo gerador de tráfego configurado para gerar um tráfego de voz de duração de uma hora.

5.1 Coleta de Dados

Os testes realizados para obter os arquivos de registro consistem na geração de tráfego entre duas máquinas utilizando a ferramenta **Traffic Generator** do **Tangram-II**. O programa **Traffic Generator** é composto por dois módulos: um módulo emissor e um módulo receptor. O módulo emissor pode ser configurado para enviar pacotes usando diversos modelos de taxa, como por exemplo: CBR, Markov Modulated ou de acordo com um arquivo de registro. Além disso, o módulo emissor pode enviar pacotes para um único destino (UNICAST) ou para vários destinos ao mesmo tempo (MULTICAST). O módulo receptor é executado na máquina destino e tem por tarefa fazer um registro em disco dos pacotes recebidos. Este módulo armazena a hora de chegada, o tamanho e o número de seqüência de cada pacote recebido. O arquivo com estes dados era compactado e transferido para um máquina na UFRJ no final do teste para posterior análise.

Para emular um aplicativo de voz, o módulo emissor foi configurado para enviar pacotes UDP a uma taxa constante (CBR) com intervalo entre os pacotes de 20 milissegundos. O tamanho dos pacotes foi constante e igual 324 bytes e a duração de cada experimento era de uma hora. O intervalo entre pacotes escolhido representa o valor usado pela maioria dos programas de comunicação por voz usando a internet. O tamanho dos pacotes igual a 324 bytes representa o tamanho dos pacotes enviados pela ferramenta **VivaVoz** [5] e além disso é um valor próximo do tamanho do pacote usado pela ferramenta **FreePhone** [2] quando configurada para enviar 1 redundância por pacote.

Estes testes foram realizados entre 2001 e 2002. Em 2001 entre a Universidade Federal do Rio de Janeiro (UFRJ) e a Universidade Federal de Minas Gerais (UFMG), entre a UFRJ e a University of Maryland (UMD) e entre a UFRJ e a University of Massachusetts Amherst (UMASS) em diferentes horários para que se pudesse observar o comportamento da perda de pacotes com a rede em diferentes estágios de congestionamento. No total foram gerados 74 arquivos. Destes 74 arquivos, 28 representam o tráfego iniciado na UFRJ e coletado na UMD, no período de 9 de agosto a 22 de agosto, 21 entre UFRJ e UFMG, no período de 9 de agosto a 17 de agosto, e 25 entre a UFRJ e a UMASS, no período de 8 de agosto a 16 de agosto.

Em 2002 estes testes foram repetidos entre a UFRJ e a UMASS e dentro da UFRJ totalizando 198 arquivos. No período de 16 de janeiro de 2002 a 4 de fevereiro de 2002 foram coletados 71 arquivos com o tráfego originado na UFRJ e com destino UMASS, e 27 arquivos no sentido oposto. No período de 19 de junho de 2002 a 5 de julho de 2002 foram coletados 97 arquivos de registro entre a UFRJ e a UMASS. 3 arquivos representam o tráfego de uma máquina na UFRJ para outra máquina na UFRJ com o objetivo de verificar a influência dos roteadores internos no processo de perda de pacotes.

Do total de 272 arquivos de registro, 3 encontram-se corrompidos e foram, portanto, descartados.

5.2 Características dos Arquivos de Registro

Dos 269 arquivos, 196 (72.87%) apresentaram um taxa de perda menor do que 1%, 67 (24.91%) tiveram taxa de perda entre 1 e 10% e os outros 6 (2.23%) arquivos tiveram taxa de perda entre 10 e 17.64%. O número de pacotes que chegaram fora de ordem foi desprezível em todos os arquivos.

É importante ressaltar que o canal internacional entre Brasil e EUA foi alterado em agosto de 2001, elevando significativamente a taxa de transmissão a 45 Mbps [3]. É claro então que, devido à baixa utilização do canal na época da mudança, houve uma redução da taxa de descarte. Entretanto a coleta inclui dados numa faixa razoável de taxa de perda e o algoritmo de recuperação a ser usado deve se adaptar a essas variações.

Para 107 dos 269 arquivos, foram traçados o gráfico da auto-correlação [23] e em apenas 30 o gráfico não apresentava correlação entre as perdas, isto é, o processo de perdas poderia ser modelado por uma variável aleatória *Bernoulli*. 61 arquivos mostraram um gráfico de auto-correlação semelhante à Figura 5.3. Esse comportamento sugere a presença de um evento periódico que sobrecarrega algum enlace no caminho fim-a-fim, possivelmente devido a atualização da tabela de roteamento como foi observado em [18].

Uma característica importante é a distribuição da rajada de perdas de pacotes, isto é, a distribuição do número de pacotes perdidos entre dois pacotes que chegaram com sucesso. Esta métrica tem influência direta na qualidade do som e portanto na escolha do esquema de redundância. A Figura 5.1 mostra o número médio de ocorrências de 6 faixas de tamanho das perdas. Os resultados mostram que a maioria das médias tem tamanho entre 1 e 2. Resultados similares foram observados em outros estudos ([9] e [33]).

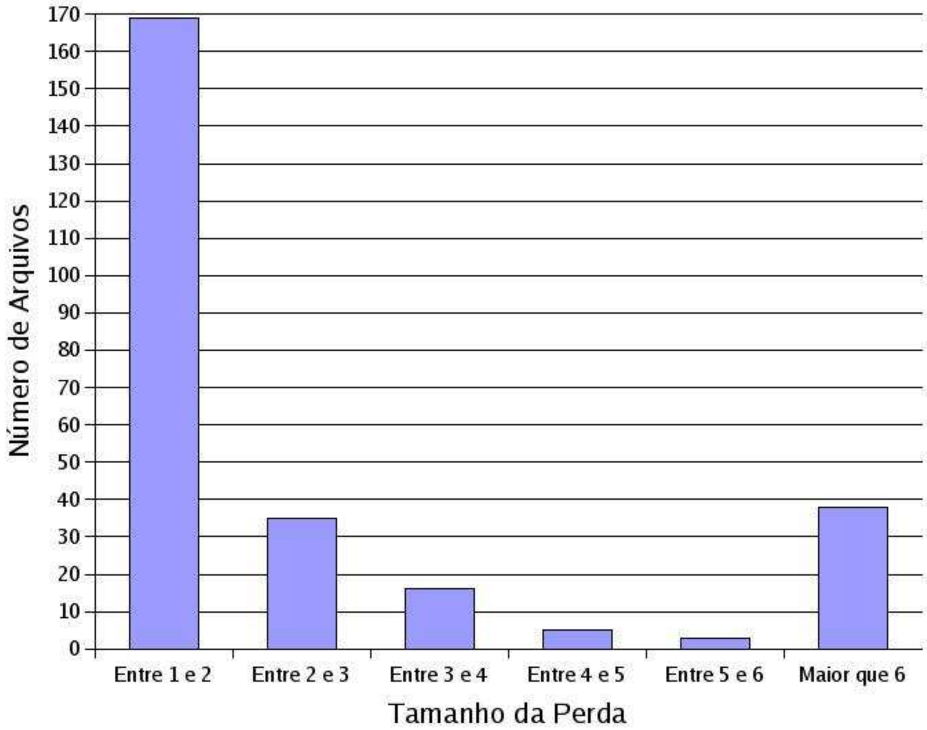

Figura 5.1: Tamanho da Perda X Número de Arquivos

A Figura 5.1 também mostra um número médio relativamente alto de arquivos com perdas maiores que 6. Destes arquivos 8 tiveram o tamanho médio das perdas maiores que 100 pacotes. A Figura 5.2 mostra o gráfico do número de perdas em cada segundo para um dos arquivos com esta característica. Pela figura pode-se observar que o caminho entre o módulo emissor e o módulo receptor do **Traffic Generator** foi interrompido em algum ponto ocasionando uma parada de aproximadamente 5 minutos, gerando 15250 perdas consecutivas. Arquivos com tamanho médio de perdas maiores que 6 também mostram gráfico semelhantes ao da Figura 5.2.

A Tabela 5.1 mostra algumas estatísticas sobre o tamanho médio do sucesso, que é o número médio de pacotes que chegaram com sucesso entre duas perdas, para um arquivo de registro.

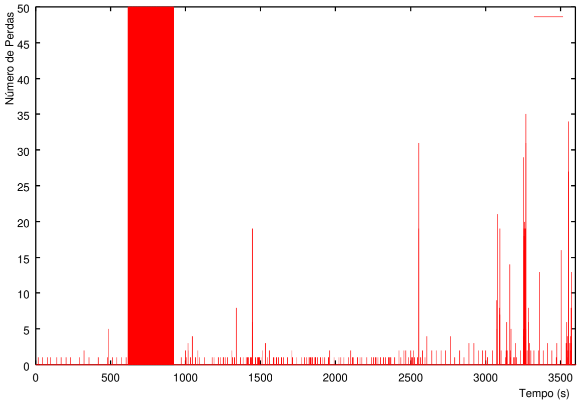

Figura 5.2: Perdas

5.3 O Modelo do Processo de Perda de Pacotes

Com o objetivo de modelar o processo de perda de pacotes com auto-correlação semelhante à Figura 5.3, uma série de testes foi realizada para se determinar a matriz de transição e a ordem desta matriz.

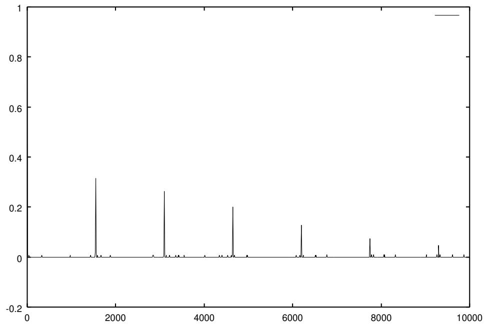

Figura 5.3: Auto-correlação

O primeiro teste consistiu em se usar uma matriz de transição com apenas dois

Medida	**Valor**
Menor	9.97
Média	4676.64
Maior	133871
Variância	1.77541e+08

Tabela 5.1: Estatísticas do tamanho do sucesso

estados e com valores aleatórios para as transições. Após o treinamento com 100 iterações, a cadeia de Markov Oculta degenerou para uma cadeia de Markov Normal, onde um estado era responsável pela emissão do símbolo zero e o outro estado responsável pela emissão do símbolo 1. Este modelo é conhecido como modelo de *Gilbert* ([8]).

Foram experimentadas cadeias com um número maior de estados mas, em todos os testes, nenhuma cadeia foi capaz de gerar uma seqüência de observações com um comportamento da auto-correlação semelhante ao da Figura 5.3.

Os próximos testes usaram como base o intervalo entre os picos de auto-correlação. Pode-se verificar que o intervalo é de aproximadamente 30 segundos. Criar uma cadeia onde cada estado representa uma perda e a cadeia represente um período da auto-correlação mostrou-se inviável, pois tal cadeia teria aproximadamente 1500 estados e o tempo de treinamento desta cadeia não seria factível para aplicações de tempo real. Por isso foi necessária a agregação de amostras consecutivas em um único estado. A idéia é que cada estado represente agora o número de perdas no intervalo de um segundo. Como o tamanho dos pacotes de áudio da aplicação considerada era de 20 milisegundos, cada estado modela o processo de perda de 50 pacotes de áudio e, portanto, cada estado pode emitir 51 símbolos, indicando o número de perdas em um segundo. Essas novas amostras serão chamadas de amostras agregadas.

Com essa nova representação do processo de perda de pacotes, perde-se informação sobre a distribuição das perdas dentro do segundo. Não é possível determinar, por exemplo, o espaçamento dos pacotes perdidos dentro da perda. Em trabalhos futuros pode-se explorar outros modelos, por exemplo, um símbolo poderia representar o número máximo de perdas consecutivas dentro do intervalo de um segundo.

É necessário escolher ainda a estrutura para a cadeia a ser usada. Obviamente poder-se-ia escolher uma estrutura geral onde existem transições de cada estado para todos os outros estados. Entretanto estruturas mais especializadas podem produzir melhores resultados dependendo do tipo de sinal a ser modelado [22]. No caso em questão, observou-se alguma periodicidade no processo de perda, ou em outras palavras, uma forte correlação em *lags* da ordem de segundos. Caso existisse uma periodicidade de exatamente 30 segundos, obviamente uma cadeia periódica com 30 estados representaria com exatidão o processo de perda. Esta observação motivou a estrutura escolhida mostrada na Figura 5.4, onde uma cadeia do tipo "*Branch-Erlangian*" (semelhante a *phase-type*) foi escolhida.

O primeiro teste com esse modelo utilizou uma cadeia com 30 estados, valor aproximado do período da auto-correlação. Cada estado podia emitir qualquer um dos 51 símbolos com igual probabilidade. Além disto, cada estado podia fazer uma transição para o próximo ou para o primeiro como está ilustrado na Figura 5.4. Este modelo falhou em representar o comportamento da auto-correlação observado mesmo para um número de iterações superior a 500. O período da auto-correlação foi recomputado e verificado que o mesmo se encontrava mais próximo de 31 do que 30 segundos, por isso o próximo modelo tem a mesma estrutura que o anterior exceto pelo número de estado, que passou a ser 31 ao invés de 30. Como o resultado desta última cadeia foi um gráfico muito semelhante ao da Figura 5.3.

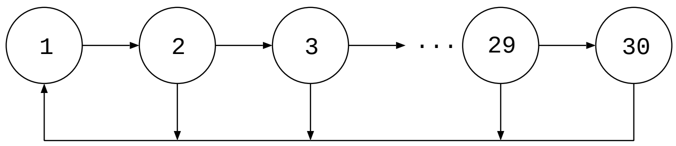

Figura 5.4: Matriz de transição

Foram feitos outros testes semelhantes aos dois últimos aumentando-se apenas o número de estados. Nestes testes a cadeia final, após o treinamento, não era ergódica e apenas os 31 primeiros estados eram atingíveis.

5.4 Algoritmo

Uma vez escolhido o modelo HMM para o processo de perdas é necessário elaborar um algoritmo a ser usado na previsão das perdas na rede.

O algoritmo proposto está ilustrado na Figura 5.5 e encontra-se quase em sua totalidade implementado na ferramenta **hmm** mencionada na Seção 4.2. A única exceção foi a calculo da quantidade de perdas a partir da distribuição gerada pelo programa **hmm** que foi implementado em *awk* por conveniência. Este cálculo pode ser facilmente incorporado no programa **hmm**.

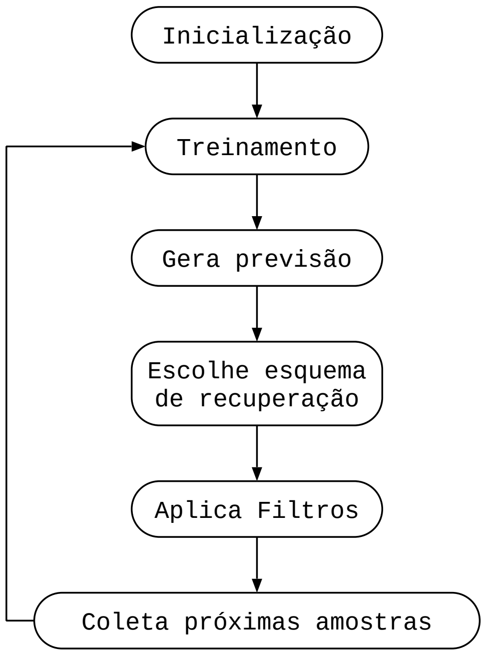

Figura 5.5: Algoritmo de previsão e recuperação

5.4.1 Inicialização

Para que o algoritmo possa inferir o estado da rede em um tempo futuro é necessário que o mesmo tenha um histórico recente da rede. Este histórico é conhecido como seqüência de observações e é usado para o treinamento da cadeia de Markov Oculta. O tamanho da seqüência de observações é de fundamental importância, pois se for muito pequeno o treinamento pode gerar uma cadeia que não reflita corretamente o estado da rede. Por outro lado, se o tamanho da seqüência de observações for muito grande, pacotes que estão sendo usados no processo de treinamento já poderiam estar sendo usados para a previsão e recuperação. Além disso, o tempo de processamento para esta seqüência inicial grande pode não ser factível para uma aplicação de tempo real.

O tamanho da seqüência inicial de observações foi obtido de forma experimental. Nos primeiros testes, a seqüência inicial consistia dos primeiros 500 segundos de áudio. Esse valor foi considerado muito grande para ser usado por um aplicativo interativo de transmissão de voz, pois é possível que o tempo de conexão seja menor do que este valor. Uma seqüência de testes foi elaborada de modo a encontrar o valor mínimo para a seqüência inicial de observações. Estes consistiam simplesmente em ir reduzindo o tamanho da seqüência inicial até que o treinamento levasse a uma cadeia que não reproduzisse o comportamento esperado da auto-correlação. O valor mínimo encontrado foi próximo 150 amostras agregadas. Foi arbitrado que o valor da seqüência seria de 3 minutos, ou seja, 180 amostras agregadas.

5.4.2 Treinamento

Uma vez obtida a cadeia inicial, esta cadeia deve ser atualizada durante a conexão de forma a se adaptar às mudanças do processo de perda com tempo. O algoritmo apresentado na seção 4.1.2 é usado para cumprir este objetivo. O número de iterações foi obtido experimentalmente. O valor arbitrado foi 100, pois, para este valor, a probabilidade de se obter a seqüência dada a cadeia já havia convergido na maioria dos experimentos. Além disso, esse número de iterações mostrou-se factível de ser computado por uma aplicação de tempo real mesmo em um computador modesto.

5.4.3 Previsão

A cadeia de Markov obtida após a fase de treinamento é usada para prever as perdas que irão acontecer no período subseqüente. A previsão é calculada em três etapas: a primeira etapa é responsável por calcular o estado mais provável de ter gerado o último símbolo observado, para este fim é usado o algoritmo de *Viterbi* visto na Seção 4.1.2. O resultado dessa etapa é armazenado no vetor π. A segunda etapa calcula a probabilidade de se ter pelo menos x perdas no período de um segundo para cada um dos próximos 60 segundos. É importante relembrar que um estado da cadeia oculta agrega informações de um período de um segundo. Então, a probabilidade da cadeia emitir o símbolo i após t segundos:

$$P(X(t) = i) = \pi A^t B'[i] \qquad 1 \leq t \leq 60$$
$$0 \leq i \leq 50 \qquad (5.1)$$

onde $X(t)$ representa o número de perdas no t-ésimo segundo, π o vetor de probabilidade de estados no início do período de previsão, encontrado pelo algoritmo de *Viterbi*, A é a matriz de transição da cadeia de Markov Oculta, B' é a matriz onde a linha j é o vetor de emissão do estado j e $B'[i]$ é a $[i+1]$-ésima coluna desta matriz, obtidas na fase de treinamento do período anterior.

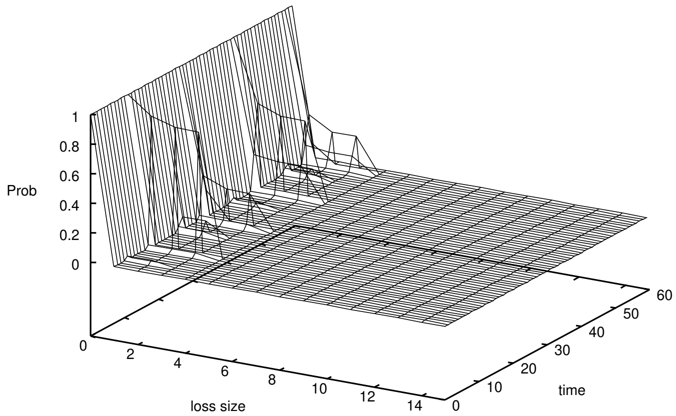

Figura 5.6: Distribuição de perdas em função do tempo

A Figura 5.6 ilustra graficamente um possível arquivo de saída desta etapa. Na figura é possível observar que no décimo primeiro segundo haverá uma alta probabilidade de se perderem 3 pacotes. Também é possível notar que a probabilidade de se observar mais que 3 perdas a cada período de um segundo durante os próximos 60 segundos é desprezível.

A terceira e última etapa calcula o número estimado de perdas que ocorrem a cada segundo com probabilidade P, onde P é uma dada tolerância. Esse número é a soma dos elementos do vetor $P(X(t))$ até que esta soma alcance P, retornando o valor de i no final. O valor escolhido para P nos experimentos foi 0.95.

Formalmente, seja $L(\delta)$ uma variável aleatória que representa o número de perdas que ocorrem em um intervalo de tamanho $\delta < \tau$. Como cada estado no modelo de Markov oculto representa um intervalo de um segundo, uma escolha apropriada de

δ é 1 segundo. Como resultado da análise transiente efetuada em cada intervalo de duração τ, obtém-se $P[L(\delta) = n]$, para $n = 0, 1, \cdots, N$, onde $N = 50$ nos testes realizados.

Foi usada a hipótese conservadora que, com probabilidade $P[L(\delta) = n]$, o tamanho da rajada no intervalo δ tem tamanho n. Seja M_j tal que $P[L(\delta_j) \leq M_j] \geq \epsilon$, onde ϵ é um dada tolerância. Além disso, seja $M_k = max_{j=k-1,k,k+1}\{M_j\}$, que é, para o intervalo δ_k, M_k é o valor máximo para M_j considerando os intervalos vizinhos. Para cada intervalo δ_k, o algoritmo escolhe um esquema de redundância apropriado de acordo com os esquemas mencionados em [13] que corrigem rajadas de tamanho M_k. Claramente, se $M_k = 0$ então não é preciso usar redundância para o intervalo considerado.

As duas primeiras etapas foram implementadas no programa **hmm** apresentado na Seção 4.2 enquanto que a terceira foi implementada utilizando a linguagem *awk*.

5.4.4 Escolha do Esquema de Redundância

Para o processo de recuperação foi utilizada a biblioteca citada na seção 3.2. Após termos a previsão da quantidade de perda de pacotes para os segundos seguintes, o próximo passo é a escolha do esquema de redundância apropriado para cada segundo subsequente como indicado acima. Esta escolha foi feita utilizando informações colidas dos arquivos de registro. A medida indica quantos pacotes serão perdidos no k-ésimo segundo. Uma média das medidas encontradas estão na Figura 5.1 na Seção 5.2. Foi dado maior peso aos esquemas que recuperam perdas consecutivas. A relação entre a quantidade de perdas prevista e o esquema de redundância, bem como o *overhead* introduzido pelo esquema, pode ser visto na Tabela 5.2. Nesta tabela, a coluna "Tamanho da perda" indica a previsão do número de perdas retornada pelo passo anterior.

Pela Tabela 5.2 pode-se observar que o algoritmo apresentado utiliza o mesmo esquema de redundância para previsões de perdas maiores ou iguais a 6. Isto foi necessário porque quanto maior for o tamanho da janela do esquema de redundância maior será o tempo necessário para se recuperar um pacote perdido. Como o objetivo é aplicar em um programa de conversação iterativa, tornou-se obrigatório não permitir esquemas com uma janela muito grande, limitando a latência da transmissão. O *overhead* indica a porcentagem extra de informações que é preciso enviar empregando um determinado esquema, em relação a informação original, supondo que a mesma técnica de codificação foi utilizada tanto para as amostras originais quanto para as redundâncias.

5.4.5 Filtros

Após o processo de treinamento algumas transições da cadeia oculta podem estar iguais a zero. Apesar deste fato não constituir um erro, ele pode levar a uma escolha indevida no próximo treinamento. O processo de treinamento descrito na capítulo 4 pode fazer com que uma transição inicialmente positiva seja levada a zero, mas não pode fazer o contrário. Uma transição que antes do processo de treinamento é igual a zero será igual a zero no final do treinamento. Como a cadeia treinada representa o estado atual do canal de comunicação, estas transições que foram anuladas podem não ser nulas em um futuro próximo, por isso é preciso garantir que todas as transições não nulas da matriz de transição inicial possam assumir valores positivos durante todo o processo de treinamento. Isto é feito em dois passos: primeiro é adicionada a matriz resultante do treinamento uma matriz

Tamanho da perda	Esquema	Overhead %
1	1:6	16.7
2	2:6	33.3
3	3:6	50
4	4:4	100
5	5:5	100
≥ 6	6:6	100

Tabela 5.2: Esquemas de redundância

com estrutura idêntica a matriz inicial. Depois é necessário fazer a normalização para garantir que a matriz resultante do primeiro passo seja uma matriz estocástica.

Seja $T(k)$ a matriz obtida após o treinamento da fase k e E uma matriz com uma estrutura correspondente ao diagrama de transição de estados mostrado na Figura 5.5, onde os elementos não nulos de E são todos iguais a ϵ, e ϵ é um valor a ser escolhido. A matriz inicial $J^*(k+1)$ a ser usada no treinamento do período $(k+1)$ é:

$$J^*(k+1) = wJ(k+1) \tag{5.2}$$

onde $J(k+1) = T(k) + E$ e w é um vetor onde o i-ésimo elemento w_i é tal que normaliza a i-ésima linha de $J(k+1)$, isto é:

$$w_i = \frac{1}{\sum_j J_{ij}(k+1)} \tag{5.3}$$

A normalização é então bem simples: todos os valores de uma linhas são divididos pela soma dos valores dessa linha. Foram realizados alguns testes para determinar o menor valor ϵ para que a matriz voltasse a ter a estrutura original minimizando as alterações nas características da matriz atual. O valor obtido para ϵ foi de 0.1.

Problema semelhante ocorre com o vetor de emissão. O valor escolhido para um elemento do vetor E de emissão foi de 0.001.

5.4.6 Coleta

O último passo no processo de previsão e recuperação é realimentar o algoritmo com amostras mais recentes garantindo assim que a cadeia represente o processo de perda atual da rede. O número de amostras escolhido para esse ítem do processo foi o equivalente a 60 segundos de conversa, ou seja, 3000 amostras.

A Figura 5.7 mostra a relação temporal de quatro das seis fases do processo de

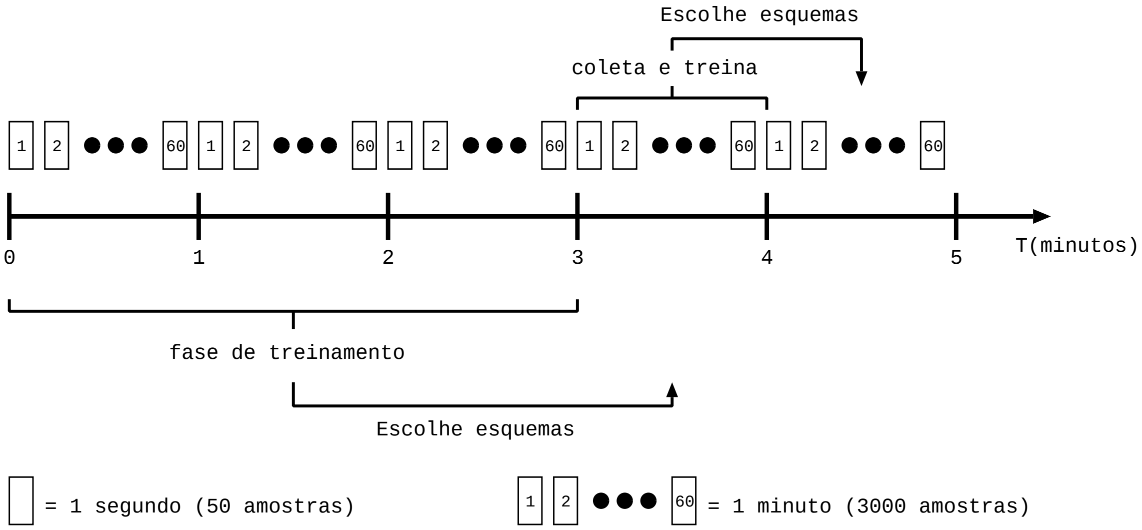

Figura 5.7: Processo de treinamento e coleta

previsão e recuperação. As fases de treinamento, inicialização e escolha de esquemas são aplicadas com informações do minuto passado, enquanto que a fase de coleta armazena o minuto presente. A aplicação dos filtros é realizada na transição de um minuto para outro enquanto que a fase de recuperação é realizada dentro do minuto.

5.5 Considerações sobre Implementação

O algoritmo apresentado neste capítulo pode ser implementado tanto no módulo emissor quanto no módulo receptor de um aplicativo de tempo real. Se o algoritmo for implementado no módulo emissor, o módulo receptor deverá enviar a quantidade de perdas observadas a cada segundo para os primeiros três minutos, relativo à fase de treinamento, e de um em um minuto ao longo da execução do aplicativo. Como o número de perdas em um segundo pode variar de 0 até 50, é preciso de pelo menos 6 bits para cada segundo, totalizando 360 bits ou 45 bytes. Por outro lado, se o algoritmo for implementado no módulo receptor, torna-se necessário o envio dos esquemas de redundância escolhidos para cada segundo deste módulo para o módulo emissor. Na atual concepção do algoritmo, são necessários apenas 3 bits para cada segundo, já que existem apenas 6 esquemas de redundância, totalizando 180 bits ou 23 bytes.

Outro ponto a ser considerado é a sincronização entre os módulos para garantir que o módulo emissor usará o esquema correto no segundo apropriado. Uma solução para este problema seria enviar junto com o vetor, que contém os esquemas a serem usados, o número de seqüência do primeiro pacote a ser protegido com o novo esquema, adicionando 32 bits à mensagem.

Como a questão da sincronização é indiferente, o fator determinante sobre a decisão de onde implementar o algoritmo proposto é a quantidade de bytes extras, necessários para o funcionamento do algoritmo, trocados entre o emissor e o receptor. Por essa razão, o algoritmo proposto deveria ser implementado no receptor.

Capítulo 6

Resultados

Este capítulo aborda os principais resultados obtidos com o algoritmo adaptativo proposto. Para validar o algoritmo proposto, foram feitas comparações com o algoritmo introduzido em [9, 10], que é um algoritmo de recuperação de perdas de pacote estático, isto é, esse algoritmo sempre envia a redundância independente da taxa de perda, ou qualquer outra variável. Em [8] é proposta um alteração no algoritmo apresentado em [9, 10] para deixá-lo dinâmico, podendo modelar o processo de perda de pacotes por um modelo de *Gilbert* ou por *Bernoulli*. O algoritmo proposto aqui também foi comparado com essa versão dinâmica modelando o processo de perda de pacotes através de uma variável aleatória *Bernoulli*.

Neste capítulo, o algoritmo apresentado em [9, 10] será chamado de algoritmo de referência estático, enquanto que a sua versão dinâmica, que está documentada, em [8] será chamada de algoritmo de referência dinâmico.

6.1 Descrição do Processo de Comparação

Para obter os resultados do algoritmo proposto, foram utilizados o programa **hmm**, a biblioteca de redundância e um conjunto de programas utilizando linguagem *script* awk e bash. O programa **hmm** foi utilizado para a previsão das perdas juntamente com o *script* descrito na Seção 5.4.3. A saída desses programas, bem como o arquivo de registro sendo analisado eram usados como entrada para o programa **VV-simul**, que utiliza a biblioteca de redundâncias descrita na Seção 3.2. Este programa simula tanto a entidade emissora quanto a receptora de um programa de transmissão de áudio pela Internet. A perda de pacotes é feita de acordo com o arquivo de registro que é passado como parâmetro. Como saída é gerado um arquivo com os números de sequência dos pacotes recuperados.

Como o algoritmo de referência estático é simples, basta recuperar o último pacote de uma rajada de perda, foi implementado um programa utilizando a linguagem *script* awk para simulá-lo.

O algoritmo de referência dinâmico foi implementado em um programa utilizando linguagem C. Assim como o programa **VV-simul**, este programa simula a emissão e a recepção de áudio pela Internet. A saída gerada por esse programa é um arquivo onde cada linha contém 1 se o pacote foi perdido e não pode ser recuperado ou 0 caso contrário. A partir deste arquivo são obtidas as informações sobre a qualidade de áudio resultante do uso deste algoritmo. Além deste arquivo, o programa calcula o *overhead* e o número de pacotes recuperados.

Em posse destes programas, foram confeccionados outros programas utilizando a linguagem *script bash* para executar cada um desses programas para todos os arquivos de registro em análise e armazenar a saída em um único arquivo de dados.

É importante ressaltar que o foco principal deste trabalho é a perda de pacotes. Por esta razão o atraso entre pacotes não foi considerado nos testes realizados.

6.2 Resultados

A Figura 6.1 mostra, no mesmo gráfico, as perdas reais, em verde, e a previsão, em vermelho durante uma hora de experimento. O tempo, em segundos, está representado no eixo das abscissas enquanto o eixo das ordenadas mostra a quantidade de perdas. Nesta figura é possível observar que o algoritmo superestima a perda na maioria das previsões. Isto acontece devido à alta tolerância (95%) escolhida na etapa de previsão (5.4.3).

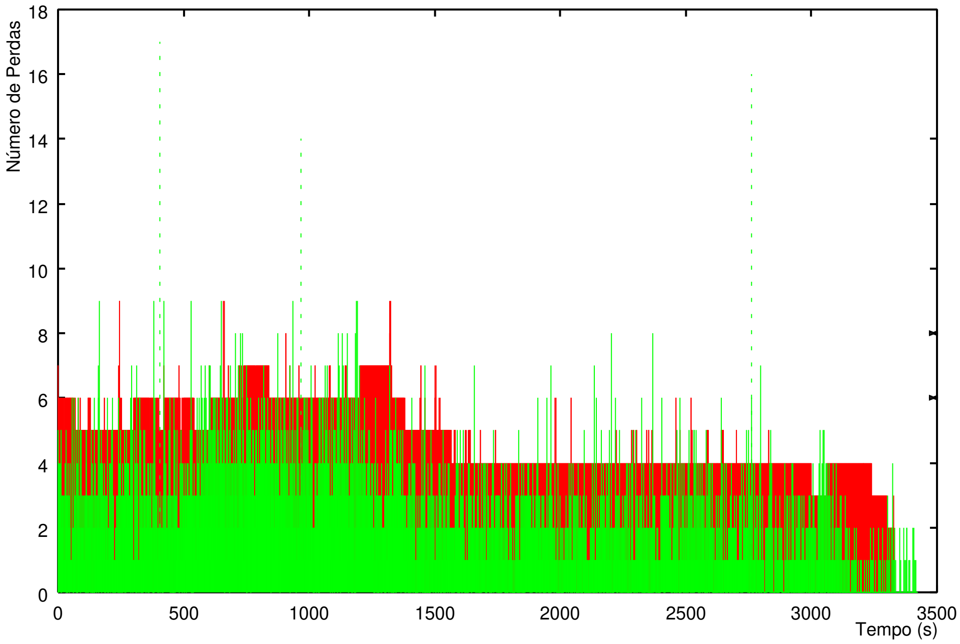

Figura 6.1: Perdas Reais versus Previsão

A Tabela 6.1 mostra um resumo da comparação do algoritmo proposto com o algoritmo de referência estático e que é usado na ferramenta FreePhone [2] para diversos arquivos de registro coletados durante vários meses. Nesta tabela as colunas Origem, Data e Hora indicam respectivamente a máquina de origem, a data e a hora em que o arquivo de registro foi coletado. A coluna "Pacotes Perdidos" mostra o número de pacotes que foram perdidos durante a coleta do arquivo. A coluna "Fase Trein." indica o número de pacotes que foram perdidos nos primeiros 3 minutos do experimento. Neste período o algoritmo proposto ainda está "aprendendo" o processo de perdas e por isso não pode prever as perdas. Uma possível solução para recuperar pacotes durante a fase de treinamento seria aplicar o algoritmo de referência estático durante os 3 primeiros minutos. A coluna "Recuperados Alg. Ref. Estático" mostra o número de pacotes que seriam recuperados se o arquivo estive protegido pelo algoritmo de referência estático. Similarmente, a coluna "Recuperados Proposto" mostra o número de pacotes que seriam recuperados se o arquivo estivesse protegido pelo algoritmo proposto aqui. As duas últimas colunas mostram o *overhead* dos dois esquemas de redundância, o de referência estático e o proposto respectivamente.

Origem	Data	Hora	Pacotes Perdidos	Fase Trein.	Recuperados Alg. Estático	Recuperados Proposto	Overhead Alg. Estático	Overhead Proposto
UMASS	03/07	16	10480	93	2082	6642	1.0	0.998965
UMASS	04/07	09	17441	103	2414	9604	1.0	0.997637
UMASS	19/06	12	4969	206	4223	4412	1.0	0.904690
UMASS	20/06	09	10238	315	8684	9118	1.0	0.999678
UMASS	20/06	12	19331	631	15491	16063	1.0	0.999678
UMASS	20/06	16	7398	172	6301	6562	1.0	0.922675
UMASS	21/06	12	4335	110	3676	3777	1.0	0.857041
UMASS	21/06	16	13984	346	11356	11998	1.0	0.992899
UMASS	24/06	09	6533	40	5704	5909	1.0	0.918938
UMASS	24/06	12	13362	807	10767	11152	1.0	0.999004
UMASS	24/06	16	13450	1229	9258	9658	1.0	0.563593
UMD	13/08	09	11104	6	172	75	1.0	0.157776
UMD	13/08	12	2597	56	2034	2195	1.0	0.856450
UMD	14/08	09	16129	6	231	228	1.0	0.309353
UMD	14/08	16	1525	15	1269	1071	1.0	0.564585
UMD	15/08	16	28560	11	3440	3276	1.0	0.435503

Tabela 6.1: Comparação Algoritmo de Referência Estático versus Proposto

Origem	Hora	Taxa de Perda	Taxa de Perda Alg. Ref. Estático	Taxa de Perda Alg. Proposto	Overhead Normalizado
UMASS	16	5.80	4.64 (0.20)	2.15 (0.63)	1.00
UMASS	09	9.70	8.34 (0.14)	4.36 (0.55)	1.00
UMASS	12	3.70	0.67 (0.82)	0.37 (0.90)	0.74
UMASS	16	7.80	1.48 (0.81)	1.09 (0.86)	0.99
UMASS	16	7.50	2.33 (0.69)	2.10 (0.72)	0.56
UFRJ	15	11.50	4.14 (0.64)	2.53 (0.78)	1.00
UFRJ	09	2.60	0.55 (0.79)	0.21 (0.92)	0.78
UFRJ	17	0.70	0.10 (0.86)	0.13 (0.81)	0.24
UFRJ	15	4.30	1.07 (0.75)	0.73 (0.83)	0.72
UFRJ	08	3.10	0.81 (0.74)	0.28 (0.91)	0.86
UFRJ	08	1.60	1.12 (0.30)	0.64 (0.60)	0.52
UFRJ	11	1.00	0.23 (0.77)	0.22 (0.78)	0.44
UMD	09	6.20	6.08 (0.02)	6.14 (0.01)	0.16
UMD	12	1.40	0.31 (0.78)	0.21 (0.85)	0.86
UMASS	16	1.40	0.18 (0.87)	0.14 (0.90)	0.49
UMASS	09	1.50	0.12 (0.92)	0.13 (0.91)	0.43
UMASS	12	0.70	0.36 (0.48)	0.27 (0.62)	0.29
UMASS	09	1.90	0.13 (0.93)	0.11 (0.94)	0.60
UMASS	16	1.70	0.15 (0.91)	0.12 (0.93)	0.47
UMASS	09	0.70	0.06 (0.92)	0.10 (0.86)	0.26

Tabela 6.2: Comparação da Taxa de Perda após passar pela Recuperação

A Tabela 6.2 mostra a comparação das taxas de perda real, após passar pelo algoritmo de referência estático e após passar pelo algoritmo proposto. Nesta tabela também é mostrado o *overhead* do algoritmo proposto normalizado pelo algoritmo de referência estático na coluna “Overhead normalizado”.

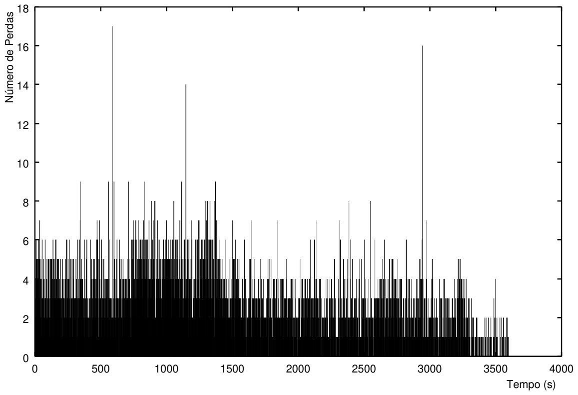

Figura 6.2: Perdas Reais

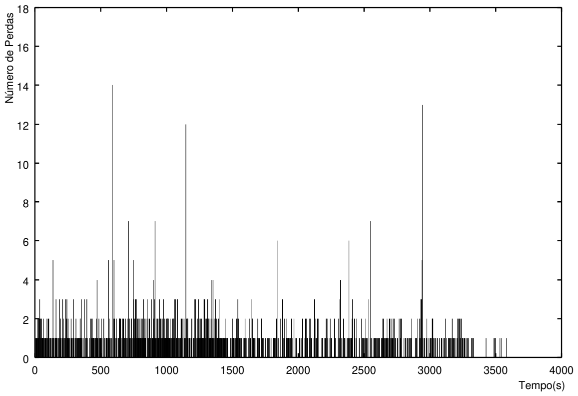

Figura 6.3: Perdas após Algoritmo de Referência Estático

As Figuras 6.2, 6.3, 6.4 e 6.5 mostram a quantidade de pacotes perdidos por segundo a cada segundo para um arquivo de registro escolhido. A Figura 6.2 mostra o gráfico sem nenhum tratamento, enquanto as Figuras 6.3, 6.4 e 6.5 mostram a quantidade de pacotes perdidos após a aplicação do algoritmo de referência estático, do algoritmo de referência dinâmico e do algoritmo proposto neste trabalho respectivamente. É evidente que o algoritmo proposto é bem superior aos algoritmos de referência estático e dinâmico.

Em apenas 8 dos 269 arquivos, o algoritmo de referência dinâmico recupera mais pacotes que o algoritmo aqui apresentado. A Tabela 6.3 mostra a comparação entre

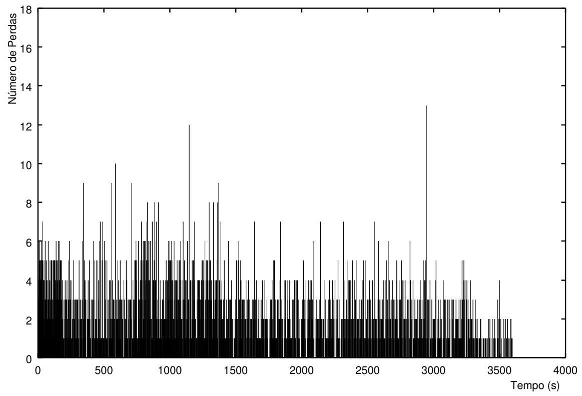

Figura 6.4: Perdas após Algoritmo de Referência Dinâmico

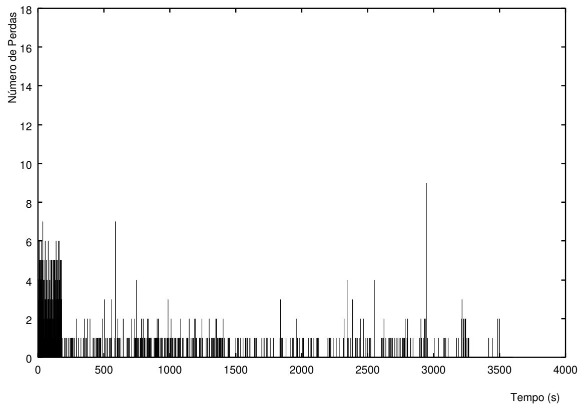

Figura 6.5: Perdas após Algoritmo Proposto

estes dois algoritmos para alguns arquivos de registro. Assim como na comparação com o algoritmo de referência estático, o número de pacotes recuperados durante os três primeiros minutos não foi contabilizado para nenhum dos dois algoritmos. As três primeiras linhas desta tabela mostram ocorrências onde o algoritmo de referência dinâmico recuperou mais pacotes do que o algoritmo proposto. O algoritmo de referência dinâmico utilizou um *overhead* menor nas duas primeiras linhas. As demais linhas desta tabela mostram um desempenho superior do algoritmo proposto. No caso da penúltima linha o algoritmo proposto recuperou mais pacotes utilizando um *overhead* menor.

A Tabela 6.4 mostra o número de pacotes recuperados por tamanho da perda para um determinado arquivo de registro. Cada entrada (x, y) mostra o número de vezes que y pacotes foram recuperados em uma rajada de erros de tamanho $x \geq y$. Por exemplo, a entrada $(5, 4)$ indica que existiram 6 rajadas de tamanho 5 que tiveram 4 pacotes recuperados (portanto apenas um pacote permaneceu perdido). Note que devido as características do algoritmo de referência estático, sempre é recuperado o último pacote de cada rajada de perdas.

Ainda com relação ao arquivo de registro analisado que deu origem a Tabela 6.4, foram recuperados $181/200 * 100\% = 90.5\%$ de todas as perdas de tamanho 3 (entrada $(3,3)$), e em 3.5% das perdas de tamanho 3 do arquivo de registro 2 pacotes foram recuperados. Por outro lado o algoritmo de referência dinâmico recuperou 37.5% de todas as perdas de tamanho 3 e ainda 2 dos 3 pacotes perdidos em 14% das rajadas de tamanho 3. Outros arquivos de registro mostram comportamento semelhante.

Origem	Data	Hora	Pacotes Perdidos	Recuperados por Alg. Ref. Dinâmico	Recuperados Proposto	Overhead Alg. Ref. Dinâmico	Overhead Proposto
UMD	15/08/01	16	28560	8218	3432	0.606851	0.413999
UFRJ	30/04/02	11	382	67	45	0.018060	0.119445
UFMG	11/08/01	12	5487	15	9	0.100009	0.059761
UMASS	20/06/02	12	19331	9858	16071	0.687486	0.949964
UMASS	04/07/02	09	17441	3878	9609	0.993055	0.948025
UMD	13/08/01	09	11104	53	272	0.194442	0.150155
UMD	13/08/01	12	2597	878	2204	0.138889	0.813900

Tabela 6.3: Comparação Algoritmo de Referência Dinâmico versus Proposto

	Algoritmo Proposto							Alg. Ref. Dinâmico						
	0	1	2	3	4	5	6	0	1	2	3	4	5	6
1	69	527	0	0	0	0	0	141	455	0	0	0	0	0
2	18	3	134	0	0	0	0	17	63	75	0	0	0	0
3	9	3	7	181	0	0	0	22	75	28	75	0	0	0
4	20	1	0	2	193	0	0	33	67	24	92	0	0	0
5	12	0	0	1	6	183	0	21	62	29	90	0	0	0
6	10	1	0	1	0	0	195	28	80	27	72	0	0	0
≥ 6	35	3	3	7	10	6	876	101	334	126	379	0	0	0

Tabela 6.4: Número de ocorrências de recuperações para Algoritmo Proposto

Capítulo 7

Conclusão

O principal objetivo deste trabalho foi o desenvolvimento de um novo algoritmo de previsão e recuperação de pacotes usando cadeias de Markov Oculta. Este algoritmo foi comparado com um algoritmo de recuperação estático [9] e com um dinâmico [8]. Os resultados destas comparações mostraram um excelente desempenho do algoritmo proposto tanto com relação ao algoritmo estático quanto ao dinâmico. Ao longo desta pesquisa foi também elaborado um conjunto de experimentações e ferramentas. Os ítens abaixo enumeram as contribuições:

- Desenvolvimento e implementação de um novo algoritmo de previsão de perda de pacotes e análise de sua eficácia para aplicação de voz sobre IP.
- Estudo do processo de perda de pacotes na Internet para aplicações de voz.
- Elaboração de um modelo de perdas de pacotes usando cadeias de Markov Ocultas.
- Implementação da teoria apresentada no Capítulo 4 no programa **hmm**. É importante ressaltar que a implementação desse programa não se restringe a aplicações de transmissão de voz em tempo real. O programa **hmm** pode ser usado separado e aplicado em outro tipo de pesquisa. Apesar da teoria estar implementada em um programa, as rotinas podem ser facilmente copiadas para outras aplicações.

- Implementação de uma biblioteca de rotinas para recuperação de perdas de pacotes baseada em [13]. Esta biblioteca foi implementada levando em consideração as recomendações feitas em [12] para transmissão de redundâncias para aplicativos de tempo real.

- Implementação do algoritmo dinâmico de recuperação de perda de pacotes apresentado em [8] usando *Bernoulli* como processo de perda de pacotes. Vale a pena lembrar que o algoritmo estático é um caso particular do algoritmo apresentado em [13] e, por isso, pode ser usado através da biblioteca descrita na Seção 3.2.

- Elaboração do ambiente de testes. Confecção de diversos programas e *scripts* para a análise de arquivos de registro coletados, filtragem de dados e apresentação dos resultados intermediários e finais.

7.1 Trabalhos Futuros

O algoritmo dinâmico desenvolvido ainda não está incorporado ao aplicativo **VivaVoz** e, consequentemente, sua avaliação foi feita utilizando arquivos de registro pré-gravados. Entretanto o algoritmo está pronto a ser incorporado ao **VivaVoz** e estará presente na sua nova versão.

Uma potencial utilização para o algoritmo de previsão seria como mecanismo de controle de congestionamento. Como o algoritmo apresentado neste estudo indica a probabilidade de um ou mais pacotes serem descartados em um intervalo de tempo, esta medida pode ser relacionada com congestionamento na rede no período em questão. Este comportamento contrasta com o algoritmo do TCP atual que reage posteriormente a detecção do descarte. O aplicativo, usando este algoritmo, poderia, então, reduzir a sua taxa de transmissão. Vale a pena lembrar que este uso não se limita a aplicativos de transmissão de mídia de tempo real, mas pode também ser usado em protocolos de transporte como o TCP.

Neste trabalho, utilizou-se o modelo onde cada estado pode emitir 51 símbolos, sendo o significado de cada símbolo o número esperado de perdas para um determinado segundo. Uma modificação a ser explorada seria alterar o significado dos símbolos para indicar o tamanho da maior rajada de perdas no intervalo. Outra mudança seria fazer com que cada estado representa-se meio segundo e treinar a cadeia de meio em meio minuto para ter um tempo de resposta mais rápido no caso de uma mudança no estado da rede. Comparações de eficácia destas modificações podem ser feitas.

Outro ponto a ser explorado constitui na variação da tolerância descrita na Seção 5.4.3. Uma diminuição no valor desta tolerância pode levar a uma diminuição da quantidade de redundâncias enviadas sem, com isso, comprometer significativamente o número de pacotes recuperados.

As rotinas desenvolvidas no programa **hmm** podem ser incluídas na ferramenta **Tangram-II** aumentando o seu poder de análise permitindo o uso de cadeias de Markov ocultas.

Referências Bibliográficas

[1] GnomeMeeting. URL http://www.gnomemeeting.org/.

[2] GNU FreePhone. URL http://www-sop.inria.fr/rodeo/fphone/.

[3] Rede Nacional de Ensino e Pesquisa. URL http://www.rnp.br/rnp/backbone-historico.html.

[4] Tangram-II. URL http://www.land.ufrj.br/.

[5] VivaVoz. URL http://www.land.ufrj.br/tools/tangram2/vivavoz/vivavoz.html.

[6] Altman, E., Barakat, C., e R., V. M. R. Queueing analysis of simple FEC schemes for IP Telephony. In *Proceedings of IEEE INFOCOM* (2001), pp. 796–804.

[7] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie and Kristie Seymore. Automating the Construction of Internet Portals with Machine Learning.

[8] Bolot, J.-C., Fosse-Parisis, S., e Towsley, D. F. Adaptive FEC-Based Error Control for Internet Telephony. In *Proceedings of IEEE INFOCOM* (March 1999), pp. 1453–1460.

[9] Bolot, J.-C., e Vega-Garcia, A. Control mechanisms for packet audio in the internet. In *INFOCOM (1)* (March 1996), pp. 232–239.

[10] Bolot, J.-C., e Vega-Garcia, A. The case for FEC-based error control for packet audio in the Internet. *to appear in ACM Multimedia Systems* (1998).

[11] Boudec, J. L., e Boutremans, C. Adaptive delay aware error control for internet telephony, 2001.

[12] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. C. Bolot, A. Vega-Garcia and S. Fosse-Parisis. RTP Payload for Redundant Audio Data, February 1997. RFC 2198.

[13] D. R. Figueiredo, E. de Souza e Silva. Efficient Mechanisms for Recovering Voice Packets in the Internet. In *Proceedings of IEEE/Globecom'99, Global Internet: Application and Technology Symposium.* (December 1999), pp. 1830–1837.

[14] D. R. Figueiredo, F. P. Duarte. Ferramenta para Análise e Medição do Tráfego de Voz. Projeto Final, UFRJ, 1997.

[15] Diniz, M., e de Souza e Silva, E. Models for Jitter Control at Destination. In *Proc., IEEE Intern. Telecomm. Symp.* (1996), pp. 118–122.

[16] Eaton, J. W. GNU Octave. URL http://www.octave.com.

[17] Edmundo de Souza e Silva, Rosa M. M. Leão, Berthier Ribeiro-Neto and Sérgio Campos. Performance Issues of Multimedia Applications, 2002.

[18] Floyd, S., e Jacobson, V. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking 2*, 2 (1994), 122–136.

[19] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. RTP: A transport for real time applications, February 1996. RFC 1889.

[20] James F. Kurose and Keith W. Ross. *Computer Networking - A Top-Down Approach Featuring the Internet.* Addison-Wesley, 2000.

[21] Kavé Salamatian and Sandrine Vaton. Hidden Markov Modeling for Network Communication Channels. In *ACM SIGMETRICS* (2001).

[22] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *IEEE* (February 1989), pp. 257–286.

[23] Leão, R., de Souza e Silva, E., e de Lucena, S. C. A Set of tools for Traffic Modelling, Analysis and Experimentation. In *Lecture Notes in Computer Science 1786 (TOOLS'00)* (2000), pp. 40–55.

[24] M. C. Diniz and Edmundo de Souza e Silva. Análise de Mecanismos de Controle de jitter em Redes ATM. In *XXIII Seminário Integrado de Software e Hardware* (august 1996), pp. 303–314.

[25] MatLab. MatLab. URL http://www.mathtools.com.

[26] Matthew Podolsky, Cynthia Romer and Steven McCanne. Simulation of FEC-Based Error Control for Packet Audio on the Internet. In *IEEE* (1998), pp. 505–515.

[27] Omar Ait-Hellal, Eitan Altman, Alain Jean-Marie, Irina A. Kurkova. On loss probabilities in presence of redundant packets and several traffic sources. In *Performance Evaluation* (1999), pp. 36–37.

[28] Perkins, C., Hodson, O., e Hardman, V. A survey of packet-loss recovery techniques for streaming audio, 1998.

[29] R. Durbin, S. Eddy, A. Krogh and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998.

[30] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1994.

[31] Salis, L. *Correlação do Polimorfismo do Gene do Angiotensinogênio M235T, Hipertensão Arterial e Outros Fatores de Risco Cardiovascular em População Brasileira*. PhD thesis. UFRJ.

[32] Seong-Won Yuk, Min-Gyu Kang, Byung-Cheol Shin and Dong-Ho Cho. An Adaptive Redundancy Control Method for Erasure-Code-Based Real-Time Data Transmission Over the Internet. In *IEEE Transactions on Multimedia Vol 3, No 3* (September 2001).

[33] YAJNIK, M., MOON, S. B., KUROSE, J. F., E TOWSLEY, D. F. Measurement and modeling of the temporal dependence in packet loss. In *INFOCOM (1)* (1999), pp. 345–352.