Vinicius Teixeira Coelho

Monitoramento de Qualidade de Serviço em VoIP

Florianópolis - SC, Brasil

08 de julho de 2008

Vinicius Teixeira Coelho

Monitoramento de Qualidade de Serviço em VoIP

Monografia apresentada para obteno do Grau de Bacharel em Sistemas de Informação pela Universidade Federal de Santa Catarina.

Orientador:
Dr. Roberto Willrich

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis - SC, Brasil

08 de julho de 2008

Monografia de Projeto Final de Graduação sob o título *"Monitoramento de Qualidade de Serviço em VoIP"*, defendida por Vinicius Teixeira Coelho e aprovada em 08 de julho de 2008, em Florianópolis, Estado de Santa Catarina, pela banca examinadora constituda pelos professores:

Prof. Dr. Roberto Willrich
Orientador

Prof. Dr. Mario Antonio Ribeiro Dantas
Orientador

Prof. Dr. Vitório Bruno Mazzola
Universidade Federal de Santa Catarina

Dedico este trabalho
a meus pais e meu irmão
que ajudam na minha caminhada

Agradecimentos

Agradeço primeiramente ao professor Roberto Willrich pela orientação neste trabalho.
Aos amigos que formam o grupo NERDS SIN, pela ajuda e apoio nas horas difíceis e pelas festa nas horas fáceis.
E por último, mas não menos importante, a minha namorada Samanta pelo grande apoio, compreensão e ajuda nos momentos de estresse.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

"A voz era uma commodity. Estamos reinventando-a 100 anos depois", diz John Blake, executivo da BT Global Services, subsidiária da British Telecom, operadora que vez uma reviravolta e garantiu seu lugar na vanguarda da telefonia IP. Com o VoIP não importa se um cabo telefônico esta ou não chegando a sua casa, também não importa o destino da ligação. Como tudo o que passa pela Internet, os pacotes de voz são apenas mais um tipo de dado trafegando. "Basicamente é o fim da ligação a longa distância", afirma Cássio Garcia, diretor da subsidiária da Nortel. "Em 2008, provavelmente o modelo de longa distancia não exista mais", diz Patrícia Volgi, gerente da Yankee Group (FORTES, 2005).

Um estudo da Infonetics Research, estima que os gastos com VoIP deverão movimentar U$ 120 bilhões de dólares até 2009, entre os países da América do Norte, Europa, Ásia e Oceânia. Uma pesquisa da In-Stat corrobora com a estimativa, mostrando que em 2006, 34 milhões de pessoas passou a adotar algum tipo de serviço de voz sobre IP, fazendo o número de usuários da telefonia IP saltar de 16 milhões para 50 milhões.

Com todo este crescimento e o aumento de investimento em torno do VoIP, as empresas buscam meios que lhe garantam e mostrem que os recursos utilizados para a implantação do sistema de telefonia IP em seus escritórios e prédios, não foram indevidos. Para tal se faz necessário o uso de mecanismos que permitam que o serviço de voz sobre IP tenha uma qualidade assegurada, a isso damos o nome de Qualidade de Serviço (QoS – Quality of Service). Trata-se de que a taxa de transmissão, a taxa de erro, o atraso sofrido nos fluxos de dados, possam ser medidos, melhorados e, em alguns casos, garantidos para atender as expectativas dos usuários. E quando se trata dos usuários da telefonia IP, estas expectativas são maiores e mais observadas (GOMES, 2005).

Tomado pelos crescentes investimentos na área da telefonia IP, este trabalho visa fazer um estudo sobre qualidade de serviço em voz sobre IP, mostrando conceitos fundamentais, desafios de QoS, sendo mostrado o RTCP (entre outros) o protocolo desenvolvido para o controlo de qualidade de serviço em sistemas multimídias, para que desta forma seja elaborada uma ferramenta

que será utilizada para capturar tais pacotes proporcionando assim para que os administradores possam medir como esta a qualidade de voz em suas redes. Será ainda no decorrer do trabalho apresentado o PABX IP Asterisk, que será utilizado para a geração do tráfego de voz.

2 QoS - Qualidade de Serviço

Percebeu-se anteriormente que com o crescimento dos sistemas de voz sobre IP e a adoção deste sistemas por parte das empresas e dos cliente domésticos, faz-se jus o estudo da garantia da Qualidade de Serviço, que é essencial em aplicações de comunicação em tempo real. Neste capítulo estará sendo apresentado as considerações conceituais sobre QoS.

2.1 O que é qualidade de serviço

Não é apresentada pelos teóricos uma expressão única que defina qualidade de serviço. Se tem QoS em sistemas operacionais, sistemas de arquivos, entre outros. Em redes, qualidade de serviço refere-se a habilidade que uma rede tem de prover melhor serviço para um determinado tráfego. Em redes IP o QoS é um aspecto operacional fundamental para o desenvolvimento das aplicações em tempo real. É importante a compreensão dos parâmetros, mecanismos, algoritmos e protocolos desenvolvidos. A obtenção de um QoS adequado requer a estruturação em todos os níveis da uma rede, desde o topo até sua base. Qualquer má configuração ou quebra em algum ponto, a qualidade estará comprometida. (MELO, 2001)

É verdade também que nem todas as aplicações necessitarão de um controle de qualidade. Um ponto fundamental para a qualidade de serviço esta na largura de banda. Este parâmetro é normalmente considerado na fase de projeto de implantação. A largura de banda é um recurso finito por isso os administradores de rede devem projetar os mecanismos de QoS afim de assegurar a vazão que as aplicações exigem. Porém um QoS para um aplicação de prioridade alta não deve impossibilitar o uso de uma aplicação de prioridade baixa.(MELO, 2001)

Baixo segue dois mecanismos básicos para a qualidade de serviço em redes:

- Reserva de recursos: os recursos da rede são divididos de acordo com os requisitos de QoS da aplicação, e sujeitos a política de administração de largura de banda. O RSVP (Resource ReSerVation Protocol), por exemplo, fornece os mecanismos para a

implantação de serviços integrados (InterServ) baseados na reserva de recursos. (MELO, 2001)

- Priorização: o tráfego é classificado e os recursos de rede são divididos de acordo com critérios de políticas de administração de largura de banda. Para tal os pacotes de dados são classificados sofrendo tratamento preferencial as aplicações que necessitam mais recursos. A arquitetura de Serviços Diferenciados (DiffServ) é um exemplo. (MELO, 2001)

2.2 Parâmetros essenciais de Qualidade de Serviço

Além da largura de banda, outros requisitos devem ser levados em consideração para que haja uma implantação satisfatória de QoS. A minimização do atraso fim-a-fim e a minimização da taxa de perdas de pacotes são alguns dos parâmetros de QoS que serão abordados a seguir:

2.2.1 Largura de Banda

Como mencionado anteriormente, a largura de banda é o parâmetro básico para a operação adequada da qualidade das aplicações. As aplicações geram vazão que devem ser atendidas pela rede. A Tabela 2.2.1 demonstra a largura de banda de algumas aplicações de rede:

Aplicação	**Largura de Banda**
Voz	10 Kbps a 120 Kbps
Vídeo (Streaming)	100 Kbps a 1Mbps
Aplicação de Conferência	500 Kbps a 1Mbps

Tabela 2.1: Largura de banda requerida em algumas aplicações

Como mencionado este requisito é avaliado no início do projeto de implantação do uso da rede, levando em consideração todas as aplicações que será utilizada a rede e a quantidade de tráfego que as mesma podem gerar.(GOMES, 2005)

2.2.2 Latência

A latência é outro aspecto importante a ser avaliado para a implantação de qualidade de serviço. Ela denota os atrasos sofridos na transmissão dos dados em uma rede. Os fatores que influenciam a latência são:

- atraso da propagação
- velocidade de transmissão
- processamento nos equipamentos

O atraso da propagação corresponde ao tempo necessário para a propagação do sinal elétrico ou óptico e é o parâmetro que o administrador de redes não tem influência.

A velocidade de transmissão é o parâmetro controlado pelo administrador visando a adequação da rede à qualidade de serviço solicitado. Atualmente as velocidades de transmissão de redes locais (LAN) e redes de longa distância (MAN e WAN), estão equiparadas, temos hoje velocidades chegando a 8Mbps, mesmo ainda tendo os custos mensais altos nada se compara os dos modelos anteriores com velocidades bem abaixo da metade dos modelos atuais.

O terceiro fator que contribui para a latência é a contribuição do atraso referente ao processamento realizado nos equipamentos de rede. Ao longo do percurso em uma rede, os pacotes de dados são processados por:

- roteadores
- switches
- servidores de acesso remoto (RAS)
- firewalls

Considerando que a latência é um parâmetro fim-a-fim, os equipamentos finais também tem sua parcela de contribuição para o atraso. No caso dos equipamentos finais o atraso depende de uma séria de fatores, como:

- capacidade de processamento
- disponibilidade de memória
- mecanismos de cache
- processamento nas camadas de níveis superiores da rede (Programa de aplicação, camadas acima da camada IP)

Em resumo, se verifica que os equipamentos finais são um fator importante para a qualidade de serviço, e em determinados casos, podem ser pontos críticos na garantia de QoS. Apesar disso são os mais fáceis de melhoria, pois somente a troca do processador para um de maior capacidade, ou memória, ajuda, principalmente os servidores a trabalhar com maiores quantidades de dados.(GOMES, 2005)

2.2.3 Jitter

O jitter é outro parâmetro importante para a qualidade de serviço. No caso, o jitter é importante para as aplicações executando em rede cuja operação adequada depende de alguma forma da garantia de que as informações (pacotes) devem ser processadas em períodos de tempo bem definidos, como é o caso de aplicações em tempo real como a voz sobre IP.

O jitter pode ser entendido como a variação no tempo e na seqüência de entrega das informações devido a latência da rede. Conforme mostrado anteriormente, a rede e seus equipamentos impõem atraso, e estes atrasos são variáveis devido ao:

- tempo de processamento diferente entre os equipamentos
- tempo de retenção diferente imposto pelo rede

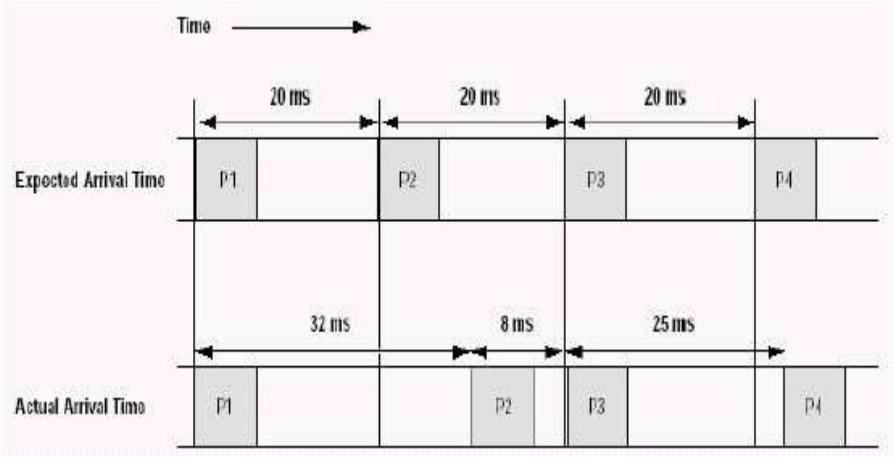

Figura 2.1: Variação de chegada de pacotes

Caso houvesse uma taxa de transmissão constante com intervalos igualmente constante entre as transmissões dos pacotes, os mesmo deveriam chegar em ordem e no intervalo em que foram mandados, mas como no caso das rede IP, cada pacote pode trafegar por rotas diferentes e pelos outros motivos já mencionados, este tempo de chegada é variável o que resulta numa

qualidade de serviço ruim. Em uma aplicação VoIP, uma pessoa fala em uma ponta, sua voz é codificada transformando-se em pacotes de dados IP, nessa fala ela diz: “Bom-dia vizinho”. A fala é dividida em dois pacotes: o pacote “Bom-dia” e o pacote “vizinho”. Se o nível de jitter for alto pode ocorrer de o pacote “vizinho” chegar primeiro que o pacote “Bom-dia”, ou ainda de pacote “Bom-dia” chegar primeiro, mas 3 segundos depois o pacote “vizinho”. Analisando isso em uma conversação convencional fica inviável a comunicação pois as duas partes não se entenderiam corretamente.

Em princípio o problema dos pacotes fora de ordem poderia ser resolvido com o auxílio de um protocolo de transporte como o TCP, que verifica a seqüência das mensagens e faz as devidas correções. Porém como as aplicações de voz sobre IP utilizam, em sua maioria, o protocolo UDP pela sua simplecidade e menor overhead, este problema teve ser resolvido por protocolos de mais alto nível normalmente incorporados na própria aplicação, como por exemplo o RTP (será abordo ainda neste trabalho). Outro controle esta associado ao tempo de chegada dos pacotes, para isso é configurado um *buffer* (*jitter buffer*), porém este *buffer* também acrescenta um certo atraso, entretanto em alguns casos, tolerável que não prejudica a compreensão da conversa.(HERSENT; GURLE; PETIT, 2002)

2.2.4 Perdas

As perdas de pacotes em redes IP ocorrem principalmente em função de fatores tais como:

- descarte de pacotes no roteadores e switches, por causa de erros e congestionamento
- erros ocorrido na camada 2 durante a transmissão dos dados

De maneira geral, as perdas de pacotes IP são um problema séria para determinadas aplicações como, por exemplo a voz sobre IP. Neste caso específico, a perda de pacotes com trechos da voz digitalizada implica numa perda de qualidade eventualmente não aceitável para a aplicação. O que fazer em casos de perdas de pacotes é uma questão específica de cada aplicação em particular.

Do ponto de vista de qualidade de serviço da rede a preocupação é normalmente no sentido de especificar e garantir limites razoáveis (Taxas de Perda) que permitam uma operação adequada da aplicação.(TANANBAUM, 2003)

2.2.5 Cabeamento

Um parâmentro muitas vezes esquecido pelo administradores de rede é a questão do cabeamento. Um projeto de cabeamento mal implementado, especificações técnicas e padrões normativos não respeitados pode levar a todas as configurações feitas nos equipamentos, de controle de tráfego, a um colapso, inviabilizando o bem estar da qualidade da rede.

2.2.6 Disponibilidade

A disponibilidade é um aspecto da qualidade de serviço abordada normalmente na fase de projeto da rede. Em termos práticos, a disponibilidade é medida da garantia de execução da aplicação ao longo do tempo e depende de fatores como:

- disponibilidade dos equipamentos utilizados na rede privada (LAN)
- disponibilidade da rede pública (ISP – Internet Service Providers)

A empresas dependem cada vez mais das redes de computadores para a viabilização de seus negócios, isso é visível na voz sobre IP, que todo o tráfego de telefonia da empresa esta passando pela rede, por isso a disponibilidade é um requisito de qualidade de serviço bastante rígido, chegando a taxas de acima de 99% do tempo.

2.2.7 Confiabilidade

Outro requisito é a confiabilidade, ele esta ligado com a disponibilidade das aplicações, e esta relacionado a rede como um todo. Maneiras de aumentar a confiabilidade é o uso de links e equipamentos redundantes, produzindo assim redes com maior disponibilidade e menos suscetíveis a falhas, aumentando a qualidade de serviço.

2.2.8 Segurança

Segurança é um fator que sempre deve ser considerado, essencialmente em redes corporativas, seja em aplicações de tempo real ou não. Para manter a qualidade de serviço a acesso a rede deve ser permitido somente a pessoas autorizadas. Deve-se também evitar ataques de denial-of-service (DoS) colocando endereços IP privados (não válidos) para os equipamentos de voz. O uso de firewalls e técnicas de criptografia também devem ser avaliados pelos administradores de rede.

3 *Controle de Tráfego*

A técnica mais comum de tratamento de qualidade de serviço são os algoritmos de controle de tráfego. Com eles é possível decidir os pacotes prioritários, diminuir o nível de congestionamento da rede e o processamento dos equipamentos de rede. Este capitulo mostrará algumas técnicas de controle de tráfego.

3.1 Enfileiramento

Um modo que os equipamentos de rede usam para diminuir o congestionamento nos pacotes de entrada de interface é usar um algoritmo de enfileiramento para ordenar o tráfego dos pacotes, que são armazenados na memória até serem processados, e depois utilizar alguma técnica de priorização para a interface de saída (MELO, 2001). Tipicamente, o enfileiramento nos roteadores quando os pacotes são recebidos pela interface (fila de entrada) e ocorre também antes da transmissão para a outra interface (file de saída).

O roteador básico é composto por um conjunto de processos de entrada, que remontam os pacotes na forma que foram recebidos, verificando a sua integridade, um ou mais processos de encaminhamento, que determina a interface de destino do pacote, e processos de saída, que estruturam e transmites os pacotes para o próximo nó da rede. O conjuto desse processos compreende as funções basicas de roteador como é mostrado na Figura 3.1.

É importante entender o papel que as estratégias de enfileiramento desempenham nas redes, a sua complexidade e o efeito que os mecanismos de enfileiramento tem no sistema de roteamento e de aplicações para que sejam gerenciados de forma adequada garantindo assim uma qualidade de serviço elevada. (MELO, 2001)

O gerenciamento de filas depende basicamente do algoritmo de enfileiramento dos pacotes e do tamanho máximo da fila. A escolha do algoritmo e o tamanho máximo da fila pode parecer relativamente simples, porém esta escolha pode ser extremamente difícil em função do padrão de comportamento do tráfego na rede ser randômico. Se um tamanho elevado é im-

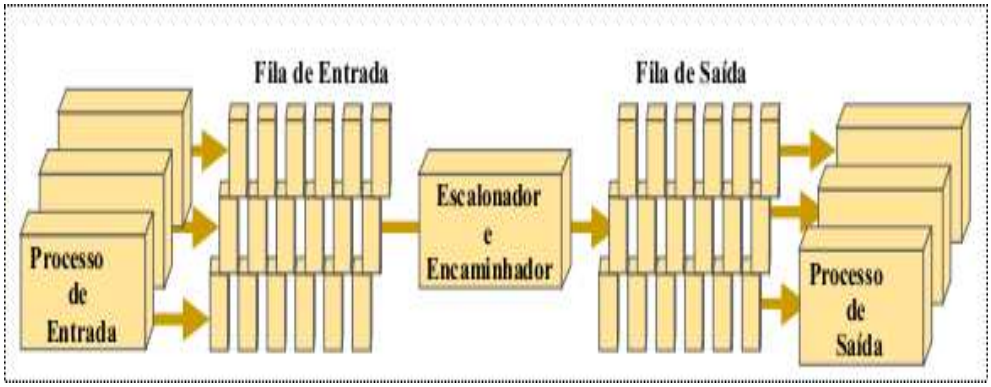

Figura 3.1: Processos de um roteador

posto à fila, introduz atraso e jitter, que como já mencionado, pode interromper ou prejudicar na comunicação. Entretanto, uma fila muito pequena pode surgir o problema de enviar dados de forma mais rápida do que a rede pode suportar, provocando assim uma excessiva perda de pacotes.(MELO, 2001)

A seguir serão apresentados alguns algoritmo de enfileiramento:

3.1.1 Enfileiramento *First In, First Out* (FIFO)

O enfileiramento first in, first out (primeiro a entrar, primeiro a sair), caracteriza o roteamento store-and-forward (armazenamento e encaminhamento). Em sua forma mais simples ele envolve o armazenamento dos pacotes quando a rede esta congestionada e o envio deles na ordem de chegada quando a rede não estiver mais congestionada. FIFO em muitos caso é o algoritmo de enfileiramento padrão, porém ele possui muitas deficiências:

- não toma decisão sobre prioridade dos pacotes
- a ordem de chegada a largura de banda que será obtida, a prioridade e a alocações de buffers
- não provê proteção contra aplicações de tráfego prejudicial (MELO, 2001)

O enfileiramento FIFO pode ser considerado o primeiro passo para o controle de tráfego, mas atualmente as rede necessitam de algoritmo mais sofisticados.

Vantagens: Quando a rede opera com suficiente nível de capacidade de transmissão e comutação, as filas são necessárias somente para assegurar tráfegos em rajada de curta duração não causem descarte de pacotes. Em tais cenários, enfileiramento FIFO é altamante eficiente,

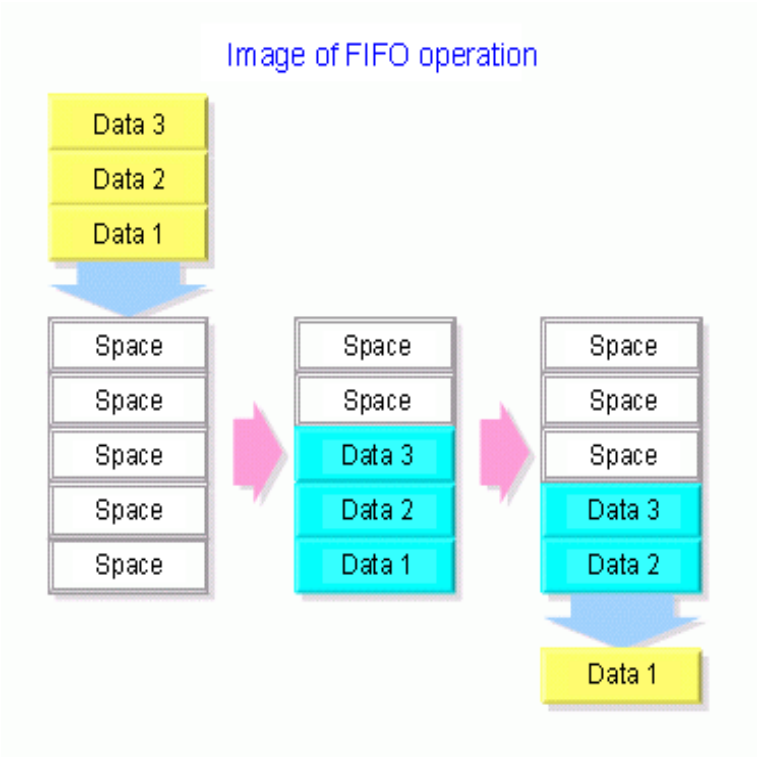

Figura 3.2: Enfileiramento FIFO - First in First Out

pois na medida que o tamanho da fila permanece pequeno, a média de atrasos dos pacotes na fila é pequeno.(MELO, 2001)

Desvantagens: Quando a carga de dados aumenta, o tráfego em rajadas causa significativo atraso de enfileiramento em relação ao tempo de transmissão total e quando a fila esta cheia todos os pacotes seguintes são descartados.(MELO, 2001)

3.1.2 Enfileiramento por Prioridade - Priority Queue

O algoritmo de enfileiramento por prioridade foi projetado para dar prioridade rígida ao tráfego importante (MELO, 2001). Este algoritmo é baseado no conceito que certos tipos de tráfego podem ser marcados e colocados a frente da fila, desta forma sendo transmitidos primeiro.

Este mecanismo de enfileiramento pode ter efeito adverso no desempenho de encaminhamento dos pacotes por causa da reordenação destes na fila e também porque o roteador tem de analisar em detalhes cada pacote para saber como ele de ser enfileirado, sobrecarregando o processador. Em um link de baixa velocidade o roteador tem mais tempo para examinar e manipular os pacotes, porém a medida que a velocidade aumenta o desempenho diminui.

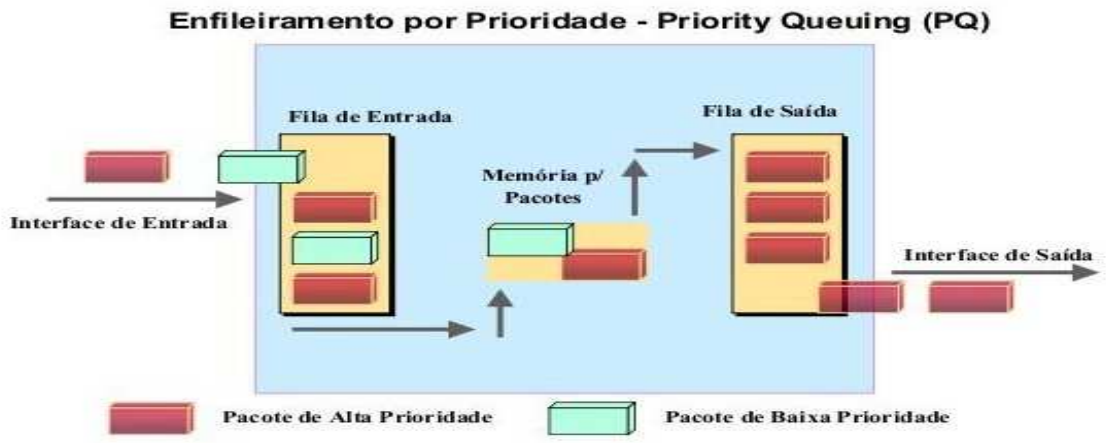

Figura 3.3: Enfileiramento por prioridade

Vantagens: Diversos níveis de prioridade pode ser definidos (alto, médio, normal, baixo). Cada um dos níveis exerce uma preferência na fila. Também a forma como o tráfego pode ser classificado na fila é flexível. Pode ser enfileirado o TCP, antes do UDP. Ainda dentro do mesmo tipo de tráfego pode ter filas diferentes como o enfileiramento do telnet (TCP 23) antes do SSH (TCP 22).

Desvantagens: Se o volume de tráfego de pacotes de alta prioridade for alto, tráfego normal esperando para entrar na fila pode ser descartado por causa de insuficiência de espaço de armazenamento.(MELO, 2001)

3.1.3 Enfileiramento Baseado em Classes – *Class-Based Queing (CBQ)*

O algoritmo de enfileiramento CBQ (*Classes-Based Queing*) ou CQ (*Custom Queing*) foi projetado para permitir que várias aplicações, com especificações de banda mínimas ou exigências de latência controlada, compartilhem a rede (MELO, 2001). Este algoritmo é uma variação do enfileiramento de prioridade, onde várias filas de saída podem ser definidas. Pode-se também definir a prioridade das filas (qual será servida primeiro, quantas vezes) e a quantidade de tráfego enfileirado que deve ser enviada a cada em cada passagem da rotação do algoritmo.

Utilizando o CBQ é possível estipular qual a largura de banda que determinada fila ira usar para enviar os pacotes, e para o tráfego restante a banda remanecente. O CBQ manipula o tráfego reservando um espaço da fila para cada classe de pacotes e depois servindo a fila via *round-robin*.

Vantagens: Ele provê tráfego para as diversas classes, fazendo assim com que todos os pa-

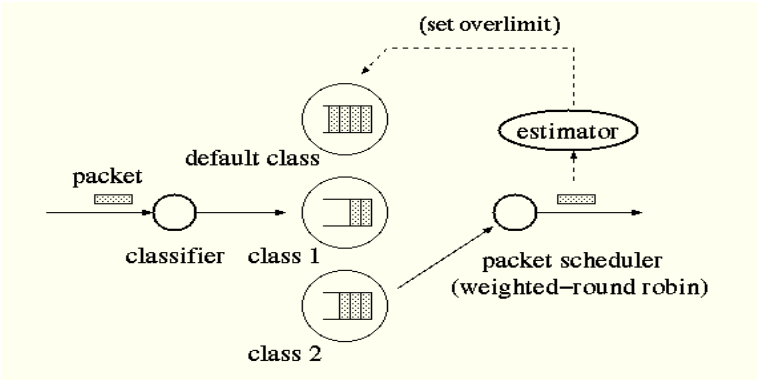

Figura 3.4: Enfileiramento CBQ - Class-Based Queing

cotes tenham vez para ser enviados diferente do enfileiramento de prioridade, em que enquanto haver tráfego de prioridade maior os pacotes de prioridade menor nunca serão enviados.

Desvantagens: Alguns anos atrás uma desvantagem do CBQ era a sobrecarga no processamento para a reordenação de pacotes e gerenciamento das filas em links de alta velocidade, atualmente com o aumento do clock dos processadores essas desvantagem vem diminuindo ainda mais, porém com o aumento do processamento, aumenta a quantidade de tráfego que um equipamento pode gerar, por isso o conhecimento do administrador de rede sobre as necessidades da infra-estrutura é fundamental para a definição de politicas de filas.

3.1.4 Enfileiramento *Fair Queuing*

O mecanismo de enfileiramento *Fair Queuing* procura prover um comportamento previsível no envio de pacotes. Ele fornece um tratamento preferencial para pacotes de baixo tráfego e o restante da banda fica para os pacotes de maior tráfego. Ele pode ser usado para situações nas quais é desejável prover tempo de resposta consistente a usuários pesados e leves de modo semelhante, sem ter que adicionar largura de banda excessiva. (MELO, 2001)

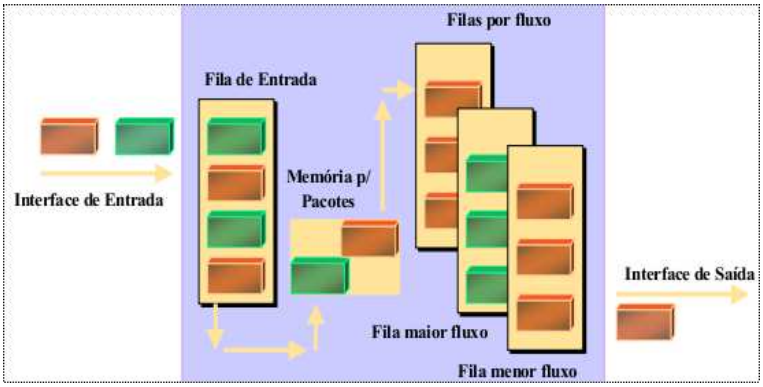

Figura 3.5: Enfileiramento FQ - Fair Queing

Vantagens: Proporciona um desempenho melhor que o enfileiramento FIFO.

Desvantages: Apesar de prover um tratamento justo com o tráfego, não é possível modificar o tipo de classifição de tráfego.

4 *Voz Sobre Internet Protocol*

Até o momento foi apresentado uma conceitualização sobre Qualidade de Serviço e Controle de Tráfego para redes. Nas próximas sessões esara sendo apresentada mais especificamente sobre Voz sobre IP, dendo com isso fechado o ciclo que ira auxiliar no desenvolvimento da aplicação proposta no inicio do trabalho.

4.1 Histórico

A mais de 100 anos atrás Alexandre Graham Bell estava patenteando sua nova invenção, o telefone. Na inicio cada telefone tinha que ser ligado por um fio a outro telefone, assim se o usuário A quisesse conversar com o usuário B e o usuário C, ele tinha que levar um fio até o telefono de cada um. Com isso as cidades ficaram emaranhadas de fios e para resolver tal problema Bell criou a *Bell Telephone Company*, que tinha por objetivo criar estações de comutação para a ligação dos diversos telofones. Com isso se tinha um ponto central onde todos os fios chegava e quando o usuário A quisesse falar com o usuário B, girava um manivela que tocava um campainha na cetral onde um operador fazia a comutação manual. Porém com o tempo usuário de uma cidade gostariam de falar com usuário de outra cidade, fazem com que a *Bell Telephone Company* criasse ponto de comutação entres os ponto centrais. (TANANBAUM, 2003)

Em pouquissimo tempo os Estados Unidos estavam tominados pelas centrais telefonicas e de comutação, em um sistema que não mudou sua operação básica por 100 anos. Nessa meio tempo novas tecnologias foram empregadas como a criação de comutadores (switches) automaticos, eliminando a presença humana, os fios de cobre que interligavam as cetrais telefonicas foram substituidas por fibra óptica, etc. (SOUZA, 2005)

Durante as décadas de 1960 e 1970 o governo dos Estados Unidos invetiram em um rede que pudesse inteligar os vários setores e com que quanto um ponto da rede caisse ela continuasse a funcionar. Foi criado então a ARPANET, que para não ter que re-cabear todo o país usou

o sistema de telefonia para a comunicação. No inicio inteligava setores de governo, posteriormente usado nas instituições de ensino e por último residencias, sendo chamada de Internet.

Com isso o sistema público de telefonia que antes era usado somente para voz, estava agora transportando também dados. Pórem o tráfego de dados cresceu mais e por volta de 1999 o número de bits de dados transferidos igualou o número de bits de voz. Em 2002 o volume do tráfego de dados ultrapassou o de voz em dez vezes. Com isso muitos operadoras de redes de comutação de pacotes se interessaram em transportar a voz sobre suas redes de dados. Com isso se populariza a telefonia IP ou voz sobre IP. (TANANBAUM, 2003)

4.2 Protocolos

Apesar de ser tornar popular somente nos últimos anos os protocolos utilizados em voz sobre IP e video, estão sendo elaborados desde a década de 1990. Usando o mesmo principio da pilha do protocolos TCP/IP cada protocolo deve ser aprovado por um orgão padronizador. Entre os protocolos mais utilizado para a comunição via VoIP podemos sitar o H.323, SIP, RTP e IAX.

4.2.1 H.323

Em 1996 a ITU (*International Telecommunication Union*) emitiu a recomendação **H.323** com o nome de *"Visual Telephone Systems and Equipament for Local Area Networks Wich Provide a Non-Guarenteed Quality of Service*"(Sistemas e Equipamentos de Telefonia Visual para Redes Locais que Oferecem uma Qualidade de Serviço não garantida). Em 1998 essa recomendação foi revisada dando a base para as primeiras implementações de telefonia IP.

A recomendação do H.323 é mais uma avaliação da arquitetura de telefonia que um protocolo. Ela faz referência a um grande número de protocolos para codificação de voz, sinalização, configuração de chamadas e transporte de dados em vez de especificar cada um desses elementos. Na codificação de voz o H.323 define que cada sistema implementado usando a especificação deve dar suporte ao G.711. O G.711 foi definido pelo ITU e codifica um único canal de voz realizando a amostragem 8.000/s com amostras de 8 bits a fim de fornecer voz sem compactação a 64kpbs. Com isso já podemos perceber que sistemas baseados nesse modelo utilizando o G.711 ficaria inviavel em sistemas *dial-up* que tinham uma largura de banda entre 36-52 kbps.

Apesar de exigir que seja implementado sistemas com suporte ao G.711, era permitidos

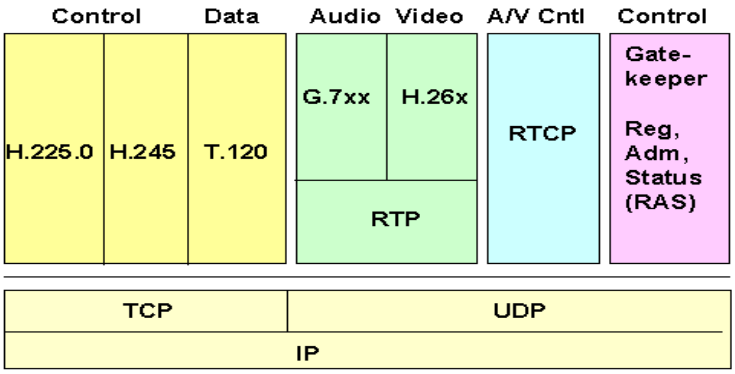

Figura 4.1: Pilha de Protocolos H.323

também outros protocolos de compactação de voz. Com isso foi desenvolvido o protocolo H.245 que faz a negociação entre os terminais o tipo de protocolo de codificação de voz. O ITU Q.931 é um protocolo que fornece e estabelecimento e encerramento de chamadas, os tipos de tons de discagem, sons de chamada entre outros dispositivos presentes na telefonia padrão. Por fim é necessário um protocolo para a transmição de dados reais, para isso é utilizado o RTP, que ainda será apresentado neste trabalho. A Figura 4.1 mostra a pilha dos protocolos.

4.2.2 SIP - *Session Initiation Protocol*

O SIP (*Session Initiation Protocol*) foi um protocolo desenvolvido pela IETF (*Internet Engineering Task Force*) para ser uma alternativa ao protocolo H.323 que era considerado pelo comitê, um produto típico das empresas de telecomunicações: grande, complexo e inflexel (TANANBAUM, 2003). Diferente do H.323, que é um conjunto de protocolos completo, o SIP é um módulo, que foi projetado para trabalhar com as aplicações da Internet. Como exemplo pode-se definir um número de telefone como uma URL, como em uma URL mailto inicia o programa de envio de mensagens a URL contendo o telefone pode iniciar a aplicação de voz sobre IP. O SIP como o nome diz é um protocolo encarregado das sessões multimídias, ele cuida da configuração, gerenciamento e encerramendo das mesmas. Outros protocolos como o RTP/RTCP são usados para o transporte dos dados de mídia. Ele é uma protocolo de camada de aplicação e pode funcionar tanto sobre o protocolo UDP quanto TCP. Os números de telefone são representados como URLs é seguem o esquema *sip*: *sip:vintc@ufsc.br*. (HERSENT; GURLE; PETIT, 2002) (TANANBAUM, 2003)

O SIP define um conjunto de métodos que são enviados as partes para a configuração e gerencia de uma chamada. A Tabela 4.1 mostra esses métodos:

A primeira etapa na comunicação SIP é abrir a sessão, para isso é enviada uma mensagem

Método	Descrição
INVITE	Solicita a inicialização de uma sessão
ACK	Confirma que uma sessão foi inicializada
BYE	Solicita o término de uma sessão
OPTIONS	Consulta um host sobre seus recursos
REGISTER	Informa um servidor de redirecionamento sobre a localização atual do usuário

Tabela 4.1: Mensagens SIP

INVITE sobre o protocolo TCP ou UDP. Dentro da mensagem INVITE encontra-se algumas informações sobre o cliente que requisitou a chamada, como o endereço e a porta que esta alocada para o recebimento dos dados, o tipo de CODEC que pode ser utilizado, entre outras

Item	**H.323**	**SIP**
Projetado por	ITU	IETF
Compatibilidade com PSTN	Sim	Ampla
Compatibilidade com a Internet	Não	Sim
Arquitetura	Monolítica	Modular
Completeza	Pilha de protocolos completa	Apenas configuração
Negociação de parâmetros	Sim	Sim
Sinalização de chamada	Q.931 sobre TCP	SIP sobre TCP ou UDP
Formato de mensagens	Binário	ASCII
Transporte de mídia	RTP/RTCP	RTP/RTCP
Chamadas de vários participantes	Sim	Sim
Conferência de multimídia	Sim	Não
Endereçamento	Número de host ou telefone	URL
Término de chamadas	Explícito ou enceramento por TCP	Explícito ou por timeout
Transmissão de mensagens instantâneas	Não	Sim
Criptografia	Sim	Sim
Tamanho do documento de padrões	1.400 páginas	250 páginas
Implementação	Grande e complexa	Moderada
Status	Extensamente distribuído	Em expansão

Tabela 4.2: Comparativo entre os protocolos H.323 e SIP

informações. O destinatario então manda uma mensagem de OK e o cliente que solicitou a chamada mando uma mensagem de ACK confirmando assim o inicio da comunicação. Durante o processa do inicialização o cliente de destino pode não aceitar o CODEC que o cliente solicitando mandou, começando assim uma troca de informações de CODECs disponiveis nos dois terminais. Esse não aceitação pode ocorrer pela falta do CODEC ou pela falta de largura de

banda para o envio. Será falado sobre CODECs mais a frente neste trabalho. O SIP te outras variedades de recursos que não serão abordados neste trabalho, porém que devem ser pelo menos conhecidos por cima pelos administradores de redes e gerentes de PABX IP que implementam tal protocolo, visando com isso um melhor aproveitamento do mesmo. A Tabela 4.2 mostra um quadro comparativo entre o H.323 e o SIP.

Outro protocolo de inicio de sessão que vem ganhando espaço é o IAX - *Inter-Asterisk eXchange* que será apresentado mais a frente no capítulo sobre o Asterisk. Além desses protocolos temos também o RTP (*Real-Time Protocol*) e o RTCP (*Real-Time Control Protocol*) que devido sua importancia será apresentados em um capítulo único.

4.3 CODECs

Outro ponto que deve ser de conhecimento dos administradores de rede e administradores de PABXs IP são os tipos de CODECs para áudio existentes. Inicialmente CODEC era mais referenciado para *COder/DECoder* (Codificador/Decodificador) que era um dispositivo que convertia o sinal analógico para digital. Agora o termo tem sido mais relacionada com *COmpression/DECopression* (Compressor/Decompressor) que diminui o tamanho do arquivo. Em verdade as duas definições se mostram corretas pois são feitas um paralelo na transmissão de dados multimídia. (MEGGELEN; SMITH; MADSEN, 2005)

Cada CODEC provê uma certa qualidade de voz, gasta uma quantidade de largura de banda e apresenta um *delay* durante o processamento. O balanço dessas características levará ao administrador escolher o CODEC com a melhor relação custo/benefício. A Tabela 4.3 alguns dos CODECs utilizados para voz sobre IP.

CODEC	**Data Bitrate (kbps)**	**Taxa de Amostragem (kHz)**	**Duração do payload (ms)**	**Tamanho do payload (bytes)**	**Requer Licença**
G.711	64	8	20	160	Não
G.726	16, 24, 32	8	15	60	Não
G.723.1	5.3, 6.3	8	30	20, 24	Sim
G.729A	8	8	20	20	Sim
GSM	13	8	20	32	Não
iBLC	13.3, 15.2	8	20, 30	30, 50	Não

Tabela 4.3: CODECs utilizados para voz sobre IP

4.3.1 G.711

O G.711 é o CODEC fundamental em redes de telefonia. Ele possui duas implementações u-law e a-law, que trabalham da mesma forma porém o u-la é mais utilizado nos Estados Unidos enquanto o a-law no resto de mundo. Cada um deles entrega 8 bits transmitidos 8000 vezes por segundo, fazendo os calculos equivale a 64000 bits por segundo ou 64kbps (8*8000) (MEGGELEN; SMITH; MADSEN, 2005). Como utiliza a mesma largura de banda que uma ligação telefonica normal, o G.711 não é muito utilizado para ligações fora da rede local (LAN). Para isso é usado outros CODECs que forneção uma taxa de compressão e consumo de largura de banda menores. Na rede local como se tem uma largura de banda na faixa de 10Mbps e 100Mbps seu uso é mais aceitável, já que o processamento gerado por ele é quase nulo.

4.3.2 G.726

O G.726 é um dos primeiros CODECs de compressão de voz. Anteriormente conhecido como G.721 este CODEC utiliza o ADPCM para (*Adaptative Differential Pulse Code Modulation* para compressão de dados. Pode gerar diferentes taxas de bits que variam em 16, 24, e 32 kbps. (MEGGELEN; SMITH; MADSEN, 2005)

O G.726 oferece a mesmo qualidade de voz que o G.711, porém utiliza a metade da largura de banda. Entretando na década de 1990 perdeu espaço por não conseguir trabalhar satisfatoriamente com sinais de modem e fax, mas como o processamente requirido e a largura de banda são baixos, este CODEC esta novamente sendo aceito.

4.3.3 G.723.1

O G.723.1 foi desenvolvido com o intuito de ser um CODEC de baixo *bitrate*, ele pode operar tanto com 5.3 kbps quanto com 6.3 kbps. Ele é um dos protocolos exigidos pelo H.323 (porém como já mensiondade outros protocolos podem ser usados). Apesar da baixa largura de banda exigida pelo protocolo ele deve ser licenciado para uso em aplicações comerciais, como a interligação de filiais de empresas. Assim como o G.726 esse CODEC não trabalha satisfatoriamente com sinais de fax e não consegue transportar sinais DTMF (*Dual Tone MultiFrequential*) os tons utilizados em telefones modernos. (MEGGELEN; SMITH; MADSEN, 2005)

4.3.4 G.729A

Considerado a baixa largura de banda que o G.729A consegue gerar uma qualidade de voz excelente. O G.729A opera em 8 kbps, porém assim como o G.723.1 é necessário o pagamento de uma licença para o seu uso em ambientes comerciais. Ele também gera uma carga muito significativa no processamento, entretando muitos equipamentos dão suporte ao G.729A. (MEGGELEN; SMITH; MADSEN, 2005)

4.3.5 GSM

O CODEC GSM ao contrario dos outros dois CODECs anteriores não necessita de uma licença para ser usado em aplicações comerciais. Ele oferece ainda um excelente performance, não sobrecaregando o processamento. Entretanto pode ter uma qualidade de voz inferior ao G.729A. O GSM requer 13 kbps de largura de banda para seu funcionamento. (MEGGELEN; SMITH; MADSEN, 2005)

4.3.6 iLBC

O *Intenet Low Bitrate Codec* prove uma união de baixo consumo de largura de banda com qualidade de voz, mesmo um links com perdas de pacotes. Apesar de ser um padrão da IETF, poucos sistemas e equipamentos de VoIP forncem superte a ele. Em comparação com a carga de processamento ele esta equiparado com o G.729A. o iLBC é um CODEC *free* não sendo necessário o pagamento de licenças para sua utilização e ocupa uma largura de banda de 13.3 kbps ou de 15.2 kbps. (MEGGELEN; SMITH; MADSEN, 2005)

5 *Real-Time Protocol/Real-Time Control Protocol*

O *Real-Time Protocol* é um protocolo que provê serviços de transporte fim-a-fim de dados para aplicações com restrições de tempo. A especificação do protocolo (RFC. . . , 1996) inclui serviços como: identificação do tipo de dados transportados, números de seqüência, timestamping, e monitoração de envio. (COSTA, 2000) (MARCONDES et al.,)

Em aplicações de tempo real o RTP é usado em conjunto com o protocolo UDP, pois este gera menos overhead que o protocolo TCP, entretanto nada impede que o RTP use o mesmo. O RTP por si só não prove nenhuma garantia de entrega dos dados no tempo desejado, tão pouco qualidade de serviço, estas funcionalidades devem ser providas pelas camadas mais baixas da pilha de protocolos. O termo real-time significa apenas “baixo overhead”. (COSTA, 2000)

O protocolo é composto por duas partes básicas, fortemente dependentes:

- O RTP propriamente dito, responsável por transmitir os dados com características de tempo real.
- O RTCP (*Real-Time Control Protocol*), utilizado para monitorar a qualidade de serviço e transportar informações sobre os participantes de uma sessão RTP.

Na telefonia IP, cada participante da ligação envia continuamente o áudio em pequenos trechos (20ms por exemplo) encapsulados num cabeçalho RTP, que por sua vez é encapsulado em um pacote UDP. Esses pacotes de áudio são enfileirados no destino para que possam ser tocados, com a devida correspondência de tempo, de modo a se manterem sincronizados. Isto é feito com o *buffer* de *playout* (ou *jitter buffer*), cujo propósito é absorver a variação de atraso, segurando os pacotes que chegam por tempo suficiente para garantir que eles sejam tocados continuamente. Quando um pacote chegar após seu tempo de *playout*, ele deverá ser descartado, havendo assim uma relação entre o atraso e a perda de pacotes RTP. (MARCONDES et al.,)

5.1 Formato dos Pacotes RTP

Um pacote RTP, como mostrado na Figura 5.1, é formado por quatro partes:

- O cabeçalho RTP.
- O cabeçalho extendido opcional.
- O cabeçalho de *payload* opcional.
- Os dados de *payload*.

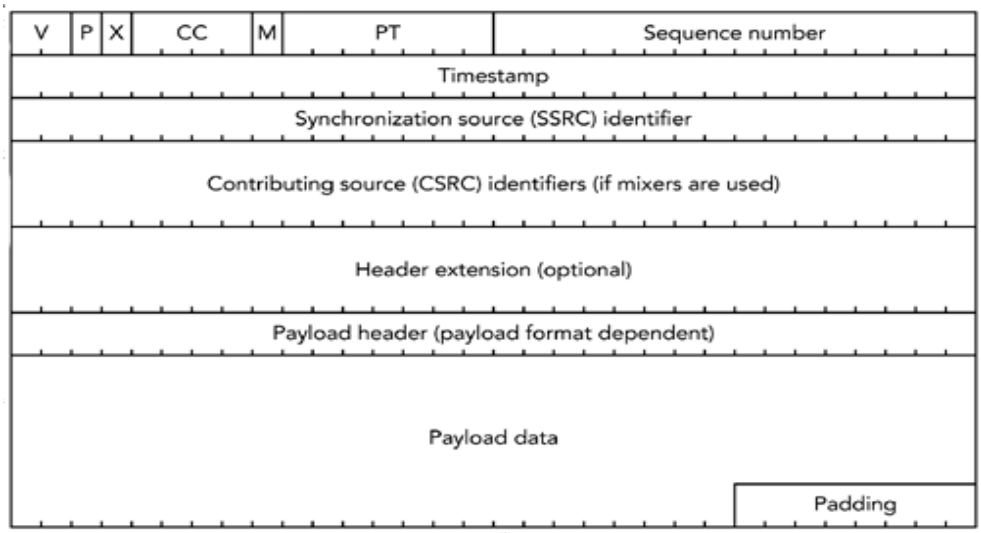

Figura 5.1: Estrutura de um Pacote *Real-Time Protocol*

O cabeçalho RTP contém 12 octetos e está presente em todos os pacotes. Os campos encontrados no cabeçalho são:

- **V - Versão:** Campo utilizado para identificar a versão do protocolo. Atualmente na versão 2.
- **P - *Padding*:** Utilizado para *padding* (enchimento). Se o bit esta ligado, o pacote contém um ou mais octetos de *padding* adicionais no final do pacote, os quais não fazem parte do da área de dados. Isso é importante para possibilitar, se desejado, o uso de algoritmos de criptografia.
- **X:** Utilizado para extensão. Se o bit está ligado, o cabeçalho fixo é seguido por uma extensão.

- **CC - Contador CSRC:** Contém o número de identificadores CSRC que seguem o cabeçalho fixo.

- **M - *Marker*:** Utilizado para marcação. A interpretação do marcador é definida por um perfil. Um perfil pode definir bits adicionais de marcação ou até mesmo excluir esses bits existentes no cabeçalho padrão. Geralmente esse bit é usado para sinalizar algum evento importante como, por exemplo, o término de um frame.

- **PT - Tipo de *payload*:** Esse campo identifica o formato da área de dados do RTP e sua interpretação pela aplicação. Alguns *payloads* são definidos no RFC 1889 e no *RFC Assigned Numbers*, como por exemplo, para o H.323 a referência é o H.225.

- **Número de Seqüencia:** O número de seqüencia é incrementado em uma unidade toda vez que um pacote de dados RTP é enviado e pode ser usado pelo destinatário para detectar perdas de pacotes e reordenação dos pacotes recebidos. O valor inicial do número de seqüencia é atribuído randomicamente.

- ***Timestamp*:** O *timestamp* reflete a amostra atual do primeiro octeto de dados RTP. A amostra é derivada de um relógio que é incrementada linearmente e monotonicamente, com o intuito de prover sincronização e cálculo de *jitter*.

- **Identificador de fonte de sincronização SSRC (*Synchronization Source Identifier*):** Esse campo identifica a origem da sincronização. Esse identificador é escolhido randonicamente, com o objetivo de evitar que hajam códigos duplicados em uma mesma sessão RTP.

- **Identificadores de fonte contribuinte CSRC (*Contributing Source Identifiers*):** Normalmente os dados RTP são gerados por apenas uma fonte, porém quando mais de uma fonte contribui para a criação dos dados passandos por um *mixer* ou tradudor seu identificador é colocado neste campo. Cada identificador é composto por 32 bits, que corresponde os SSRC de quem compribuiu. O tamanho do CSRC é definido pelo campo CC.

Além desse campos o cabeçalho RTP pode ser extendido a afim de prover maiores informaõoes caso seja requisitado por algum tipo de experimento ou aplicação. Eles são raramente usados, porém uma aplicação robusta deve ser capaz de identifica-los para que as informações do *payload* não sejam interpretadas de maneira erronea. O cabeçalho de *payload* assim como a extenção de cabeçalho provê informações adicionais sobre os dados do *payload* (PERKINS, 2003)

5.2 RTCP - *Real-Time Control Protocol*

O *Real-Time Control Protocol* (RTCP), provê relatórios periodicos sobre a qualidade, identificação de participantes e outras informações de descrição da fonte, notificação de mudanças de membros na sessão e informações necessárias para a sincronização dos streams de mídia. (PERKINS, 2003)

Todos os participantes de uma sessão RTP devem enviar esta tipo de pacote. Geralmente são enviados a cada 5 segundos, mas a medida que a quantidade de participantes aumenta, pode ser configurado para que o tempo de envio aumente também, fazendo assim com que a largura de banda não seja consumida demasiadamente por este tipo de pacote. (PERKINS, 2003) (HERSENT; GURLE; PETIT, 2002)

Cada sessão RTP é identificada por um endereço IP e um par de portas: uma para os dados RTP e outra para os dados RTCP. A porta RTP deve ser a mais alta enquanto a porta RTCP deve vir abaixo dessa, por exemplo se a comunicação RTP se da através da porta 10000, e fluxo de comunicação RTCP deve ser feito através da porta 10001. (PERKINS, 2003)

5.2.1 Formato dos pacotes RTCP

Cinco pacotes são definidos pela especificação do RTP: *receiver reports* (RR), *sender reports* (SR), *source description* (SDES), *membership management* (BYE) e *application-defined* (APP) (PERKINS, 2003). Este trabalho se aterá apenas aos pacotes *receiver reports* e *sender reports*.

O cabeçalho para todos os pacotes RTCP é mostrado na Figura 5.2 e é composto pelos campos:

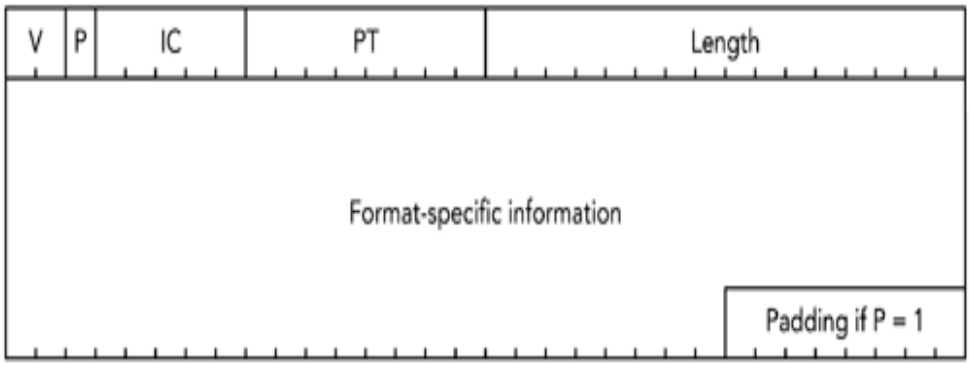

Figura 5.2: Estrutura de um Pacote *Real-Time Control Protocol*

- **V - Versão:** Campo contendo a versão do RTP.
- **P - *Padding*:** Utilizado para *padding* (enchimento). Se o bit esta ligado, o pacote contém um ou mais octetos de *padding* adicionais no final do pacote, os quais não fazem parte do da área de dados. Isso é importante para possibilitar, se desejado, o uso de algoritmos de criptografia.
- **IC - *Item Count*:** Indica quantos itens o relatório possui.
- **PT - *Packet Type*:** Indica qual é o tipo de pacote, como foi mostrato o RTCP define cinco tipos de pacotes diferentes. Para pacotes do tipo *Sender Report* o PT terá o valor 200, *Receiver Report* 201, *Source Description* 202, *Membership Management* 203 e *Application-Defined* 204.
- ***Length* (Tamanho):** Tamanho total do pacote.

Pacotes RTCP nunca são enviados individualmente, eles são agrupados em dois ou mais pacotes fomando pacotes compostos. Como mostrado na Figura 5.3 cada pacote composto é encapsulado em um único pacote da camada mais baixa para transporte. (PERKINS, 2003)

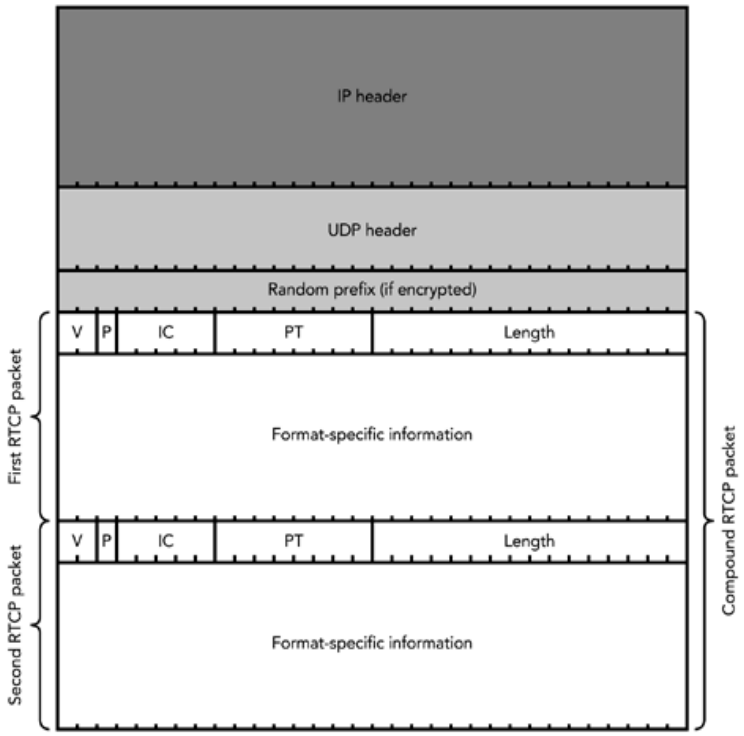

Figura 5.3: Pacote composto RTCP

5.2.2 RTCP RR - *Receiver Reports*

Um dos principais usos do RTCP é a troca de informações de qualidade o que é feito através do pacote *Receiver Report* (RR), que é enviado por todos os participantes. A Figura 5.4 mostra a estrutura de um pacote RTCP RR.

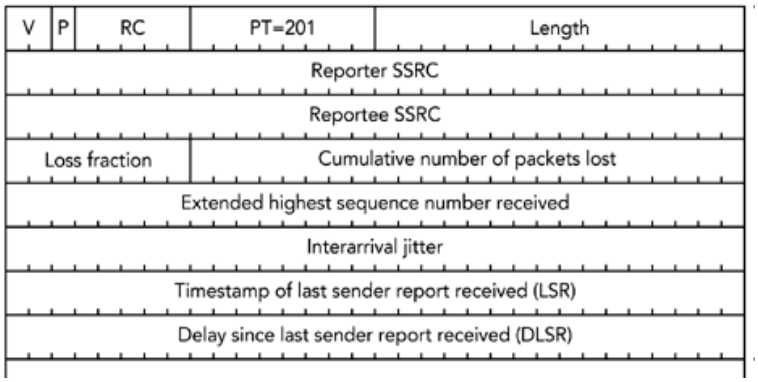

Figura 5.4: Estrutura pacote RTCP *Receiver Report*

Abaixo segue a descrição dos campos do pacote RR:

- ***Reporter SSRC*:** SSRC do participante que esta enviando o pacote.
- ***Reportee SSRC*:** Identifica de qual participante as estatísticas de qualidade do relatório esta se referindo.
- **Fração de Perda:** Valor do resultado dos pacotes perdidos / pacotes esperados multiplicando por 256.
- **Pacotes perdidos acumulados:** Número de pacotes perdidos desde o inicio da sessão.
- **Valor estendido do mais alto número de seqüencia recebido:** Os 16 bits mais significativos contêm o número de ciclos de seqüencia (isto é, o número de vezes que a contagem atingiu o valor máximo e retornou a zero) e os últimos 16 bits contêm o mais alto número de seqüencia recebido em um pacote de dados RTP proveniente dessa fonte (mesmo SSRC).
- ***Jitter* entre chegadas:** É a diferença entre o espaço de tempo entre a chegada de um par de pacotes, comparado com o *timestamp* desses pacotes, calculado da seguinte maneira: $D_{(i,j)}$ = (R_j - S_j) - (R_i - S_i), onde S_i é o *timestamp* RTP do pacote i e R_i é tempo de chegada do pacote i. Toda vez que um pacote for recebido se calcula o D e desda forma o *jitter* pela fórmula $J_n = \frac{J_{n-1}+(D-J_{n-1})}{16}$.

- **Último *timestamp* SR (LSR):** Os 32 bits do meio do *timestamp* NTP do último *Sender Report*.

- **Atraso desde o último SR (DLSR):** Tempo desde que o último pacote SR chegou. Assim como o LSR é representado na forma de *timestamp* NTP compacta.

Com esse campos é possível fazer uma serie de estimativas da qualidade da rede. Com o campo LSR e DSLR o transmissor pode calcular o tempo de ida-e-volta (*round-trip time* (RTT)) entre ele e o receptor. A Figura 5.5 mostra como é possível fazer este calculo.

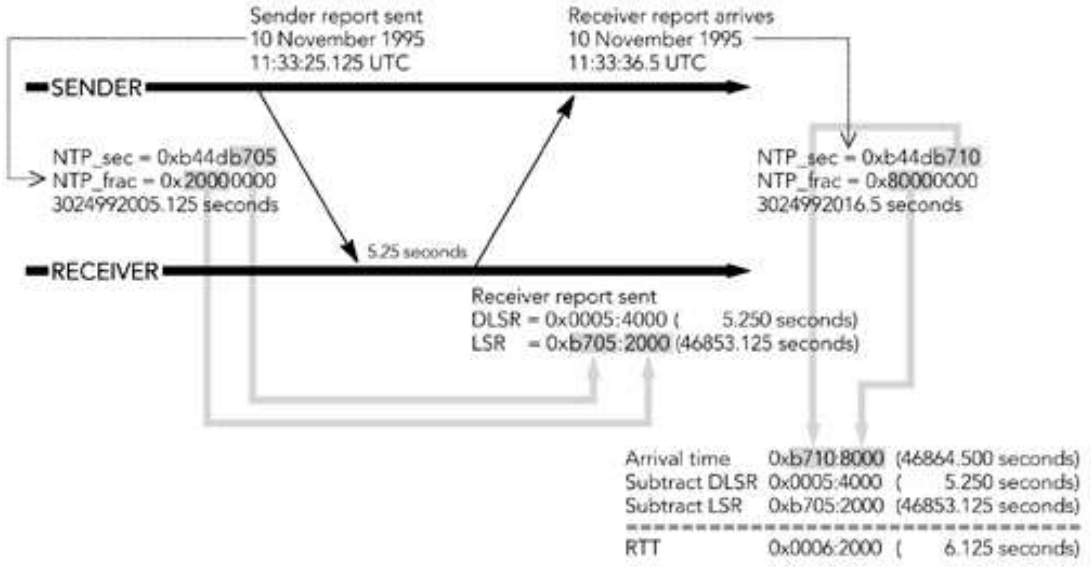

Figura 5.5: Calculo do tempo de ida-e-volta *(round-trip time)*

O tempo de ida-e-volta é importante em aplicações interativas pois o atrasa atrapalha a interatividade. Estudos mostram a dificuldade de conduzir um conversa quando o total da ida-e-volta começa a exceder o tempo de 300 milisegundos. Um transmissor ao verificar o valor de ida-e-volta pode tentar otimizar a codificaçãos da mídia transportada, gerando pacotes com menor dados, reduzindo assim o tempo de analise do pacote para transmissão. (PERKINS, 2003)

O campo de *jitter* pode ser usado para detectar o surgimento de congestionamentos na rede. Um aumento repentino no *jitter*, muitas vezes antecede o início de perdas de pacote. (PERKINS, 2003)

5.3 RTCP SR - *Sender Report*

Além dos pacotes RR enviados pelos receptores, o RTCP define os pacotes *Sender Report* que são enviados pelos prarticipantes que enviaram recentemente algum tipo de dado na sessão. Um pacote SR é identico a um RR, com o acrescimo de alguns campo como pode ser visto na Figura 5.6.

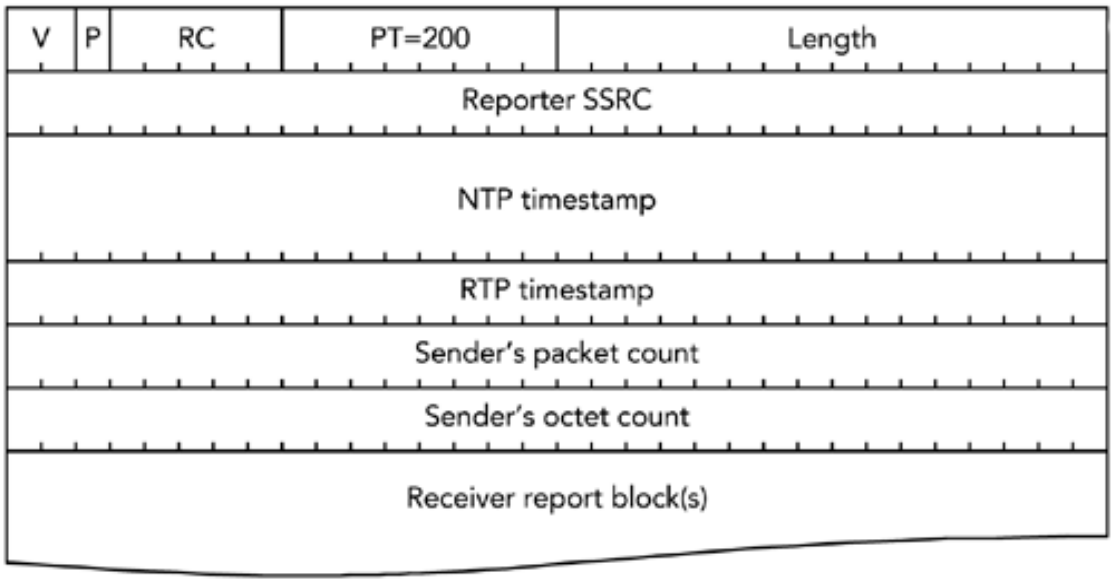

Figura 5.6: Estrutura do pacote RTCP *Sender Report*

Os campos adicionais ficam após o campo de identificação SSRC e são eles:

- **NTP *timestamp*:** Valor no formato NTP que mostra o tempo que o pacote SR foi enviado.
- **RTP *timestamp*:** Valor no mesmo formato que o NTP *timestamp* poderem medido apartir da taxa de amostragem da mídia.
- **Contagem de pacotes enviados:** Quantidade de pacotes que foi enviado desde o inicio da sessão.
- **Contagem de octetos enviados:** Quantidade de octetos contidos no pacotes enviados.

Após esse campos adicionais pode-se encontrar os campo de um pacote RR comum. (PERKINS, 2003)

6 *Asterisk*

Uma incrível revolução está acontecendo. Demorou a acontecer, mas agora que começou, não há que se possa fazer para para-lá. Ela esta acontecendo em uma área que vem se desgastando muito antes de qualquer outro setor que se entitula de alta tecnologia. A indústria das telecomunicações, e o que tem impulsionado tal revolução é o PBX *open source* Asterisk. (GONçALVES, 2006)

Todos os grandes fabricantes de telecomunicação oferecem produtos similares, não se tem flexibilidade ou escolha. O Asterisk veio para mudar isso tudo. Com ele é possível criar um sistema próprio de telefonia, decidir como serão as operações nesse sistema. Ele oferece todos os recursos que um PABX convencional, porém com a possibilidade de desenvolver aplicações capazes de interegir com ele. (MEGGELEN; SMITH; MADSEN, 2005)

O Asterisk é um software de PABX que usa o conceito de software livre (GPL), mantido pela Digium Inc., desenvolvido inicialmente por Mark Spencer e uma base de usuários em contínuo crescimento. A Digium investe em ambos, o desenvolvimento do código fonte do Asterisk e em hardware de telefonia de baixo custo que funciona com o Asterisk. O Asterisk roda em plataforma Linux e outras plataformas Unix com ou sem hardware conectando a rede pública de telefonia, PSTN (Public Service Telephony Network). Ele premite conectividade em tempo real entre as redes PSTN e VoIP. (GONçALVES, 2006)

Paralelamente ao Asterisk foi desenvolvido uma série de placas de telefonia de baixo custo para concorrer com as já existentes no mercado. Esse hardware também é de código-aberto, fazendo com que qualquer empresa possa produzir sua versão e revende-la. O projeto foi batizado de Zapata por seu desenvolvedor, Jim Dixon, e o casamento entre esse hardware e o Asterisk foi o que proporcionou a revolução emergente no meio das telecomunicações. (MEGGELEN; SMITH; MADSEN, 2005) (GONçALVES, 2006) (ZAPATA. . . , 2008)

6.1 Compatibilidade

O Asterisk é compatível com diferçõs tipos de protocolos de sinalização e CODECs. Com ele é possível integrar diversos tipos de dispositivos diferentes. Ele é capaz de prover comunicação usando protocolos SIP, H.323 e um novo tipo de protocolo: *Inter-Asteriks eXchange*. Ele além disso fornece suporte para a intergração de novos protocolos, podendo ser implementado um módulo para a integração.

Além da compatibilidade com os protocolos de sinalização, o Asterisk fornece suporte a diversos tipos de CODECs como o G.711 (u-law e a-law), G.729A (linceça que pode ser comprada com a própria Digium), iBLC, GSM, etc. Assim como nos protocolos é possível o desenvolvimento de módulos para que forneça suporte a outros tipos de CODECs. (MEGGELEN; SMITH; MADSEN, 2005)

6.2 IAX

Como já mencionado o Asterisk introduziu um novo tipo de protocolo de sinalização, o *Inter-Asterisk eXchange*. Uma das vantagens do IAX em relação ao SIP é devido o uso da porta UDP (geralmente 4569) tanto para a sinalização como para o envio da mídia. Com isso em ambientes em que um sistema de NAT (*Network Address Translation* esta presente se faz necessário o redirecionamente de somente um porta, evitando assim eventuais problemas na comunicação. Para maiores informação sobre o IAX consulte a documentação do Asterisk no site do projeto (DIGIUM, 2008) e o rascunho para RFC do IAX2 (IAX2..., 2008).

6.3 Configuração

A configuração do Asterisk é realizada por meio de uma serie de arquivos. Normalmente eles ficam no diretório */etc/asterisk*. É nesses arquivos que são configurados os ramais (*peers*) SIP, as portas usadas pelo RTP/RTCP e o plano de discagem. A Figura 6.1 mostra um exemplo da configuração de uma ramal SIP e a figura 6.2 mostra um exemplo de uma plano de discagem.

Como podemos notar foi configurado um ramal SIP que tem por número 1000 e pertence ao contexto (*context*) **default**. Quando esse ramal ligar para o número 1, de acordo com o plano de discagem ele ouvira primeiramente um áudio de Boas Vindas (*Playback(bem-vindo)*), não ouviram nada durante 2 segundos, pois a o comando de espera (*Wait(2)*) e depois ouvira o áudio de Adeus (*Playback(adeus)*) para a ligação ser finalizada (*Hangup()*). Este é apenas uma

```
[1000]
type=friend
username=1000
secret=12345
host=dynamic
context=default
nat=yes
canreinvite=no
```

Figura 6.1: Exemplo de configuração SIP

```
[default]

exten => 1,1,Playback(bem-vindo)
exten => 1,2,Wait(2)
exten => 1,3,Playback(adeus)
exten => 1,4,Hangup()
```

Figura 6.2: Exemplo de configuração de plano de discagem

exemplo básico de que pode ser feito com o plano de discagem, infelizmente este trabalho não tratará do Asterisk em si, ele só foi usado para servir de meio para os testes realizados. Para maiores informações sobre ele, consulte o site do projeto *http://www.asterisk.org/*.

7 RTCPMoni - Ferramenta Proposta

Nas analises iniciais deste trabalho, notou-se um escassez de ferramentas que pudessem auxiliar o administrador de rede a identificar alguns problemas com a rede de voz sobre IP. A ferramentas como o software Wireshark, capazes de capturar o tráfego em uma interface de rede qualquer e mostrar para o administrador os pacotes RTCP, responsável por informar como esta a qualidade como foi mostrado no Capítulo 5. Entretando esta ferramenta é limitada no sentido de extensibilidade, se dedicando apenas a captura dos pacotes.

Com isso foi proposto a desenvolver uma ferramenta que fosse capaz de realizar o trabalho dos analisadores de rede, porém voltada especificamente para voz sobre IP, capturando somente os pacotes RTCP e dando a possibilidade para que novos recursos sejam acrescentados. Nas próximas sessões estará sendo apresentada como se sucedeu o desenvolvimento da ferramenta batizada de RTCPMoni.

7.1 Arquitetura

7.1.1 Jpcap

Para realizar a captura dos pacotes que tráfegam na rede foi necessário a utilização de um biblioteca especifica para está finalidade. Como a ferramenta RTCPMoni esta sendo códificada em Java, se fez juz a utilização de uma biblioteca de captura escrita em Java, portando foi escolida a Jpcap.

A biblioteca Jpcap é baseada na libcap escrita em C. Ela fornece os mesmo recursos da libcap, tais como a captura e envios de pacotes de rede. Entre os pacotes que ela é capaz de verificar estão quadros Ethernet, pacotes IPv4, IPv6, TCP, UDP, ICMPv4, e algumas aplicações de terceiros estendem os tipos de pacotes.

Ao capturar um pacote a Jpcap fornece um objeto Java correspondente a ele, se for capturado um pacote TCP ela criara um objeto do tipo jpcap.TCPPacket, se ela capturar um pacote

na qualquer não pode verificar o tipo será criado um objeto generico do tipo jpcap.Packet, fornecendo assim recursos para que os desenvolvedores possam tratar de maneira apropriada qualquer tipo de tráfego.

7.1.2 RTCPMoni

A modelagem da ferramenta RTCPMoni pode ser vista na Figura 7.1. Ela consiste de uma classe Main.java que tem por objetivo ler os parâmentros passados pelo usuário e fazer a chamada de métodos para o inicio da execução do *software*. A classe Capture.java tem por finalidade realizar a capturas dos pacotes que tráfegam pela interface de rede selecionada e verificar se o pacote capturado contém dados RTCP. Em caso positivo as informações contidas no pacote são impressas para o usuário no *console* e gravadas em uma base de dados atravéz da classe DBMysql.java. A estrutura da tabela na base de dados é mostrada na Tabela 7.1

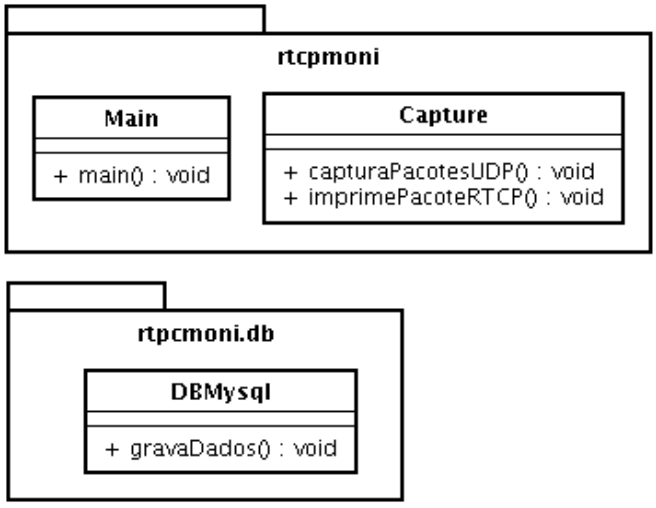

Figura 7.1: Classes da ferramenta RTCPMoni

tempoCaptura	DOUBLE
SSRC	VARCHAR(15)
jitter	INT
perdas	INT
RTT	DOUBLE

Tabela 7.1: Estrutura da tabela da base de dados RTCPMoniDB

7.2 Cenário

O cenário em que se sucedeu os testes da ferramenta RTCPMoni esta representada na Figura 7.2. Como pode ver visto foi utilizado quatro equipamentos, sendo que dois foi instalado o sistema operacional GNU/Linux Ubuntu para que pude-se ser feita a instalação do PABX Asterisk. As duas máquinas restantes foi instalado o sistema operacional Windows XP para que servi-se de base para instalação dos *softphones* para a realização das ligações. Com isso todo o tráfego de voz, juntamente com os dados RTCP, passa pelo dois PABX Asterisk.

O cenário de teste foi projetado desta maneira para se parecer com um ambiente real em que os dois servidores com o Asterisk estão dentro da rede conhecidade e de responsabilidade do administrador, enquante os dois *softphones* estão instaladas em máquinas que se conectam nos servidores através da Internet, com isso o monitoramente se faz entre os dois Asterisk.

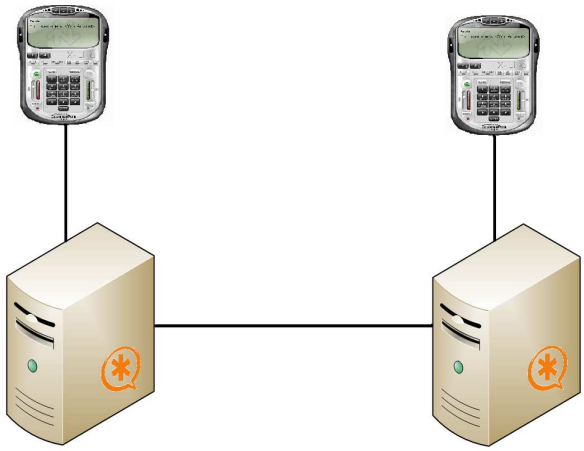

Figura 7.2: Cenário de Testes

7.3 Configurações

Como mencionado na sessão anterior foi utilizado duas máquinas com o PABX Asterisk instalado na qual foi configurado, no Asterisk A um ramal SIP referente ao *softphone* A e mais 1 ramal SIP para servir de tronco de comunicação entre as Asterisk. Foram feitas as mesmas configurações no Asterisk B. O plano de discagem dos Asterisk tem como configuração a seguinte linha: *exten => 2000,1,Dial(SIP/2:12345@172.16.0.101/2000)*, sendo feita apenas a mudança do ramal chamada entre um Asterisk e outro. Para maiores informação de configurações verificar os anexos contendo os códigos-fonte e arquivos gerais.

7.4 Monitoramento

Após terminadas as instalações e configurações necessárias, foram feitas uma serie de ligações entre os *softphones* com a ferrementa RTCPMoni rodando em um dos Asterisk. Terminadas as ligações testes uma parcela da saída da ferramenta pode ser vista na Figura 7.3.

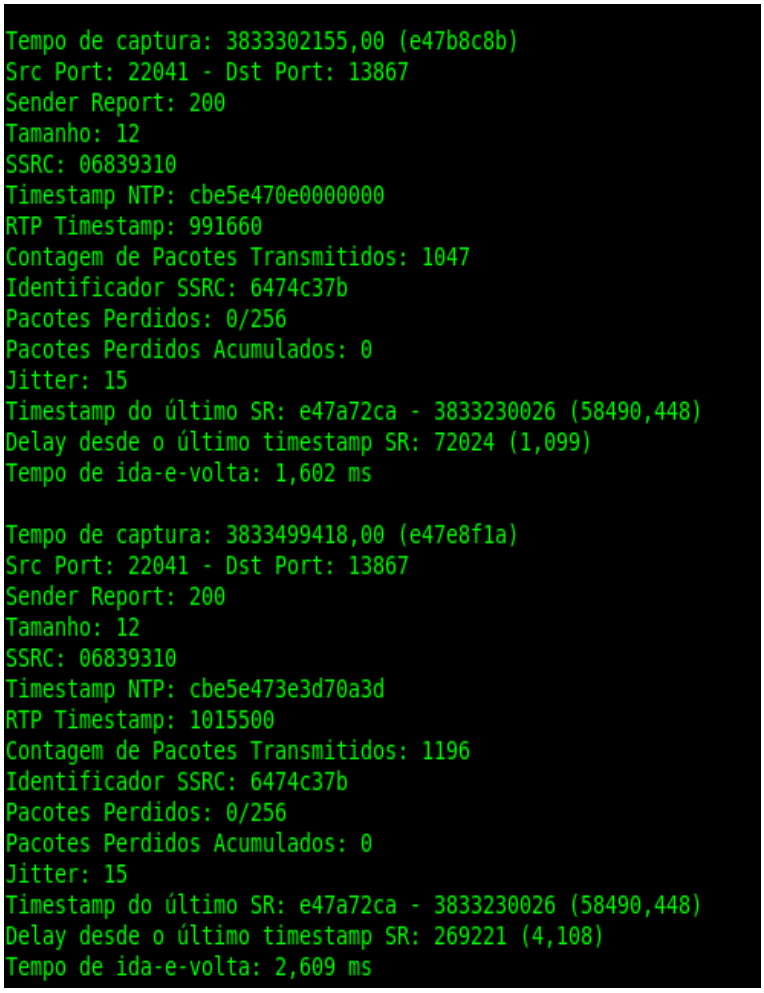

Figura 7.3: Saída da ferramenta RTCPMoni

Como pode ser visto a ferramenta é capaz de mostrar as informações do um pacote RTCP de maneira amigável e realiza o calculo do tempo de ida-e-volta entre os servidores. Como já mencionado essas informações são salvas em uma base de dados para posteriores consultas.

8 Considerações Finais

A qualidade de serviço em uma rede, especialmente em sistemas de voz sobre IP que é o foco deste trabalho, começa apartir do conhecimento que o projetista ou administrador de redes tem. Se ele conhecer bem os parâmetros de QoS, se ele conseguir configurar os equipamentos estipular uma boa política de controle de tráfego, e rede estará bem apresentável e desda forma com grandes chances de ter uma boa qualidade.

Somados a esses fatores se o administrador de rede tiver em seu poder ferramentas capazes de demonstrar como está a qualidade do serviço que a rede oferece, será possível um trabalho continua de melhoria, convergindo assim para a satisfação do usuário.

A RTCPMoni, ferramenta desenvolvida durante este trabalho, é apenas um elo da corrente. Ela é capaz de demostrar como está a qualidade de serviço do sistema VoIP para o administrador, e com a opção da gravação dos dados em uma base, é possível que o administrar possa usar posteriormente essas informações para uma demostração de cumprimento de um SLA (*Service Level Agreement*) por exemplo.

Porém é necessário salientar que como qualquer software desenvolvido a ferramenta RTCPMoni pode apresentar algum tipo de problema, por isso é pertinente comentar que o continuo desenvolvimento da mesma será feito para que cada vez mais ela possa ajudar a identificar problemas na qualidade do serviço de VoIP.

8.1 Trabalhos Futuros

Como mecionado a ferramenta desenvolvida pode apresentar algum tipo de problema que não foi identificado nos testes realizados. Por isso um dos trabalhos futuros é o continuo desenvolvimento do RTCPMoni para sanar qualquer tipo de erro que ele possa conter. Ainda será feito um estudo de integração com o Asterisk, para que apartir de verificação de alguns parâmetros como um atraso muito grande no processamento dos pacotes possa ser feita a mudança de CODECs dinamicamente, ou o aumento do *jitter buffer*.

Ainda como trabalho futuro temos o estudo de implementação de outra ferramentra para a analise de qualidade de serviço utilizando o protocolo IAX. Ainda há o intuito de desenvolver uma interface Web capaz de mostrar atrávez de gráficos a situação da qualidade de serviço, proporcionando assim mais uma ferramenta amigável disponível para o administrador de redes.

Referências Bibliográficas

COSTA, C. H. *Análise de Protocols RTP/RTCP Aplicados a Videoconferência*. Dissertação (Mestrado), 2000.

DIGIUM. *Asterisk Support*. [S.l.], 2008. Disponível em: <http://www.asterisk.org/support>.

FORTES, D. A explosão do voip. *Revista Info*, 2005.

GOMES, A. F. *Qualidade de Serviço em VOIP (Voz Sobre IP)*. Dissertação (Mestrado) — Universidade Estadual de Montes Claros, 2005.

GONçALVES, F. E. de A. *Asterisk - Guia de Configuração*. [S.l.]: V.Office Networks, 2006.

HERSENT, O.; GURLE, D.; PETIT, J.-P. *Telefonia IP - Comunicação multimidia baseada em pacotes*. [S.l.]: Prentice Hall, 2002.

IAX2 Draf 04. [S.l.], 2008. Disponível em: <http://tools.ietf.org/html/draft-guy-iax-04>.

MARCONDES, C. A. C. et al. Ambiente para simulação e monitoração de ligações telefônicas ip. In: *XXI Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.].

MEGGELEN, J. van; SMITH, J.; MADSEN, L. *Asterisk - The Future of Telephony*. [S.l.]: O'Reilly, 2005.

MELO, E. T. L. *Qualidade de Serviço em Redes IP com DiffServ: Avaliação atravás de Medições*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2001.

PERKINS, C. *RTP: Audio and Video for the Internet*. [S.l.]: Addison-Wesley Professional, 2003.

RFC 1889 - RTP: A Transport Protocol for Real-Time Applications. [S.l.], 1996. Disponível em: <http://www.ietf.org/rfc/rfc1889.txt>.

SOUZA, I. L. O. D. *SISTEMA DE GERENCIAMENTO DE PBX BASEADO NO ASTERISK EM ÂMBITO DE PEQUENO E MÉDIO PORTE*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DA BAHIA, 2005.

TANANBAUM, A. S. *Redes de Computadores*. 4º. ed. [S.l.]: Campus, 2003.

ZAPATA Telephony Project. [S.l.], 2008. Disponível em: <http://www.zapatatelephony.org/>.

Monitoramento de Qualidade de Serviço em VoIP

Vinicius T. Coelho[1]

[1]Departamento de Informática e Estatística, Universidade Federal de Santa Catarina
Campus Trindade, Caixa Postal 476, CEP 88040-900, Florianópolis-SC, BRAZIL

{vintc}@inf.ufsc.br

Resumo. *Este viza fazer um estudo de qualidade de serviço em sistema de voz sobre IP, para que seja possível a implementação de uma ferramenta capaz de monitoriar as chamadas VoIP e com isso mostrar a qualidade do serviço fornecido.*

1. Introdução

"A voz era uma commodity. Estamos reinventando-a 100 anos depois", diz John Blake, executivo da BT Global Services, subsidiária da British Telecom, operadora que vez uma reviravolta e garantiu seu lugar na vanguarda da telefonia IP. Com o VoIP não importa se um cabo telefônico esta ou não chegando a sua casa, também não importa o destino da ligação. Como tudo o que passa pela Internet, os pacotes de voz são apenas mais um tipo de dado trafegando. "Basicamente é o fim da ligação a longa distância", afirma Cássio Garcia, diretor da subsidiária da Nortel. "Em 2008, provavelmente o modelo de longa distancia não exista mais", diz Patrícia Volgi, gerente da Yankee Group (**??**).

Um estudo da Infonetics Research, estima que os gastos com VoIP deverão movimentar U$ 120 bilhões de dólares até 2009, entre os países da América do Norte, Europa, Ásia e Oceânia. Uma pesquisa da In-Stat corrobora com a estimativa, mostrando que em 2006, 34 milhões de pessoas passou a adotar algum tipo de serviço de voz sobre IP, fazendo o número de usuários da telefonia IP saltar de 16 milhões para 50 milhões.

Tomado pelos crescentes investimentos na área da telefonia IP, este trabalho visa fazer um estudo sobre qualidade de serviço em voz sobre IP, mostrando conceitos fundamentais, desafios de QoS, sendo mostrado o RTCP (entre outros) o protocolo desenvolvido para o controlo de qualidade de serviço em sistemas multimídias, para que desta forma seja elaborada uma ferramenta que será utilizada para capturar tais pacotes proporcionando assim para que os administradores possam medir como esta a qualidade de voz em suas redes. Será ainda no decorrer do trabalho apresentado o PABX IP Asterisk, que será utilizado para a geração do tráfego de voz.

2. Qualidade de Serviço

Não é apresentada pelos teóricos uma expressão única que defina qualidade de serviço. Se tem QoS em sistemas operacionais, sistemas de arquivos, entre outros. Em redes, qualidade de serviço refere-se a habilidade que uma rede tem de prover melhor serviço para um determinado tráfego. Em redes IP o QoS é um aspecto operacional fundamental para o desenvolvimento das aplicações em tempo real. É importante a compreensão dos parâmetros, mecanismos, algoritmos e protocolos desenvolvidos. A obtenção de um QoS adequado requer a estruturação em todos os níveis da uma rede, desde o topo até sua base.

Qualquer má configuração ou quebra em algum ponto, a qualidade estará comprometida. (**??**)

Baixo segue dois mecanismos básicos para a qualidade de serviço em redes:

- Reserva de recursos: os recursos da rede são divididos de acordo com os requisitos de QoS da aplicação, e sujeitos a política de administração de largura de banda. O RSVP (Resource ReSerVation Protocol), por exemplo, fornece os mecanismos para a implantação de serviços integrados (InterServ) baseados na reserva de recursos. (**??**)
- Priorização: o tráfego é classificado e os recursos de rede são divididos de acordo com critérios de políticas de administração de largura de banda. Para tal os pacotes de dados são classificados sofrendo tratamento preferencial as aplicações que necessitam mais recursos. A arquitetura de Serviços Diferenciados (DiffServ) é um exemplo. (**??**)

3. Parâmetros essenciais de Qualidade de Serviço

Além da largura de banda, outros requisitos devem ser levados em consideração para que haja uma implantação satisfatória de QoS. A minimização do atraso fim-a-fim e a minimização da taxa de perdas de pacotes são alguns dos parâmetros de QoS que serão abordados a seguir:

3.1. Largura de Banda

Como mencionado anteriormente, a largura de banda é o parâmetro básico para a operação adequada da qualidade das aplicações. As aplicações geram vazão que devem ser atendidas pela rede.

3.2. Latência

A latência é outro aspecto importante a ser avaliado para a implantação de qualidade de serviço. Ela denota os atrasos sofridos na transmissão dos dados em uma rede.

3.3. Jitter

O jitter é outro parâmetro importante para a qualidade de serviço. No caso, o jitter é importante para as aplicações executando em rede cuja operação adequada depende de alguma forma da garantia de que as informações (pacotes) devem ser processadas em períodos de tempo bem definidos, como é o caso de aplicações em tempo real como a voz sobre IP.

O jitter pode ser entendido como a variação no tempo e na seqüência de entrega das informações devido a latência da rede.

3.4. Perdas

As perdas de pacotes em redes IP ocorrem principalmente em função de fatores tais como:

- descarte de pacotes no roteadores e switches, por causa de erros e congestionamento
- erros ocorrido na camada 2 durante a transmissão dos dados

De maneira geral, as perdas de pacotes IP são um problema séria para determinadas aplicações como, por exemplo a voz sobre IP. Neste caso específico, a perda de pacotes com trechos da voz digitalizada implica numa perda de qualidade eventualmente não aceitável para a aplicação. O que fazer em casos de perdas de pacotes é uma questão específica de cada aplicação em particular.

Do ponto de vista de qualidade de serviço da rede a preocupação é normalmente no sentido de especificar e garantir limites razoáveis (Taxas de Perda) que permitam uma operação adequada da aplicação.(**??**)

3.5. Cabeamento

Um parâmentro muitas vezes esquecido pelo administradores de rede é a questão do cabeamento. Um projeto de cabeamento mal implementado, especificações técnicas e padrões normativos não respeitados pode levar a todas as configurações feitas nos equipamentos, de controle de tráfego, a um colapso, inviabilizando o bem estar da qualidade da rede.

4. Controle de Tráfego

A técnica mais comum de tratamento de qualidade de serviço são os algoritmos de controle de tráfego. Com eles é possível decidir os pacotes prioritários, diminuir o nível de congestionamento da rede e o processamento dos equipamentos de rede. Este capitulo mostrará algumas técnicas de controle de tráfego.

5. Enfileiramento

Um modo que os equipamentos de rede usam para diminuir o congestionamento nos pacotes de entrada de interface é usar um algoritmo de enfileiramento para ordenar o tráfego dos pacotes, que são armazenados na memória até serem processados, e depois utilizar alguma técnica de priorização para a interface de saída (**??**). Tipicamente, o enfileiramento nos roteadores quando os pacotes são recebidos pela interface (fila de entrada) e ocorre também antes da transmissão para a outra interface (file de saída).

O roteador básico é composto por um conjunto de processos de entrada, que remontam os pacotes na forma que foram recebidos, verificando a sua integridade, um ou mais processos de encaminhamento, que determina a interface de destino do pacote, e processos de saída, que estruturam e transmites os pacotes para o próximo nó da rede. O conjuto desse processos compreende as funções basicas de roteador como é mostrado na Figura 1.

É importante entender o papel que as estratégias de enfileiramento desempenham nas redes, a sua complexidade e o efeito que os mecanismos de enfileiramento tem no sistema de roteamento e de aplicações para que sejam gerenciados de forma adequada garantindo assim uma qualidade de serviço elevada. (**??**)

O gerenciamento de filas depende basicamente do algoritmo de enfileiramento dos pacotes e do tamanho máximo da fila. A escolha do algoritmo e o tamanho máximo da fila pode parecer relativamente simples, porém esta escolha pode ser extremamente difícil em função do padrão de comportamento do tráfego na rede ser randômico. Se um tamanho elevado é imposto à fila, introduz atraso e jitter, que como já mencionado, pode

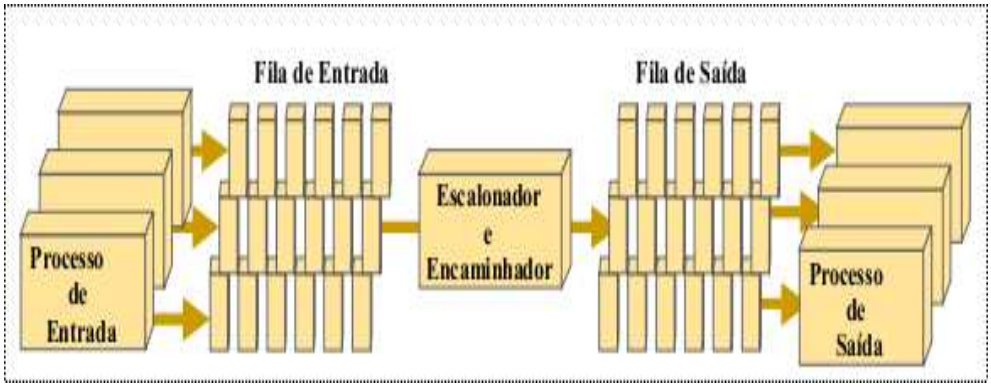

Figura 1. Processos de um roteador

interromper ou prejudicar na comunicação. Entretanto, uma fila muito pequena pode surgir o problema de enviar dados de forma mais rápida do que a rede pode suportar, provocando assim uma excessiva perda de pacotes.(**??**)

A seguir serão apresentados alguns algoritmo de enfileiramento:

5.1. Enfileiramento *First In, First Out* (FIFO)

O enfileiramento first in, first out (primeiro a entrar, primeiro a sair), caracteriza o roteamento store-and-forward (armazenamento e encaminhamento). Em sua forma mais simples ele envolve o armazenamento dos pacotes quando a rede esta congestionada e o envio deles na ordem de chegada quando a rede não estiver mais congestionada. FIFO em muitos caso é o algoritmo de enfileiramento padrão, porém ele possui muitas deficiências:

- não toma decisão sobre prioridade dos pacotes
- a ordem de chegada a largura de banda que será obtida, a prioridade e a alocações de buffers
- não provê proteção contra aplicações de tráfego prejudicial (**??**)

5.2. Enfileiramento por Prioridade - Priority Queue

O algoritmo de enfileiramento por prioridade foi projetado para dar prioridade rígida ao tráfego importante (**??**). Este algoritmo é baseado no conceito que certos tipos de tráfego podem ser marcados e colocados a frente da fila, desta forma sendo transmitidos primeiro.

Este mecanismo de enfileiramento pode ter efeito adverso no desempenho de encaminhamento dos pacotes por causa da reordenação destes na fila e também porque o roteador tem de analisar em detalhes cada pacote para saber como ele de ser enfileirado, sobrecarregando o processador. Em um link de baixa velocidade o roteador tem mais tempo para examinar e manipular os pacotes, porém a medida que a velocidade aumenta o desempenho diminui.

5.3. Enfileiramento Baseado em Classes – *Class-Based Queing (CBQ)*

O algoritmo de enfileiramento CBQ (*Classes-Based Queing*) ou CQ (*Custom Queing*) foi projetado para permitir que várias aplicações, com especificações de banda mínimas ou exigências de latência controlada, compartilhem a rede (**??**). Este algoritmo é uma

variação do enfileiramento de prioridade, onde várias filas de saída podem ser definidas. Pode-se também definir a prioridade das filas (qual será servida primeiro, quantas vezes) e a quantidade de tráfego enfileirado que deve ser enviada a cada em cada passagem da rotação do algoritmo.

Utilizando o CBQ é possível estipular qual a largura de banda que determinada fila ira usar para enviar os pacotes, e para o tráfego restante a banda remanecente. O CBQ manipula o tráfego reservando um espaço da fila para cada classe de pacotes e depois servindo a fila via *round-robin*.

5.4. Enfileiramento *Fair Queuing*

O mecanismo de enfileiramento *Fair Queuing* procura prover um comportamento previsível no envio de pacotes. Ele fornece um tratamento preferencial para pacotes de baixo tráfego e o restante da banda fica para os pacotes de maior tráfego. Ele pode ser usado para situações nas quais é desejável prover tempo de resposta consistente a usuários pesados e leves de modo semelhante, sem ter que adicionar largura de banda excessiva. (**??**)

6. Voz Sobre Internet Protocol

7. Histórico

A mais de 100 anos atrás Alexandre Graham Bell estava patenteando sua nova invenção, o telefone. Na inicio cada telefone tinha que ser ligado por um fio a outro telefone, assim se o usuário A quisesse conversar com o usuário B e o usuário C, ele tinha que levar um fio até o telefono de cada um. Com isso as cidades ficaram emaranhadas de fios e para resolver tal problema Bell criou a *Bell Telephone Company*, que tinha por objetivo criar estações de comutação para a ligação dos diversos telofones. Com isso se tinha um ponto central onde todos os fios chegava e quando o usuário A quisesse falar com o usuário B, girava um manivela que tocava um campainha na cetral onde um operador fazia a comutação manual. Porém com o tempo usuário de uma cidade gostariam de falar com usuário de outra cidade, fazem com que a *Bell Telephone Company* criasse ponto de comutação entres os ponto centrais. (**??**)

Em pouquissimo tempo os Estados Unidos estavam tominados pelas centrais telefonicas e de comutação, em um sistema que não mudou sua operação básica por 100 anos. Nessa meio tempo novas tecnologias foram empregadas como a criação de comutadores (switches) automaticos, eliminando a presença humana, os fios de cobre que interligavam as cetrais telefonicas foram substituidas por fibra óptica, etc. (**??**)

Durante as décadas de 1960 e 1970 o governo dos Estados Unidos invetiram em um rede que pudesse inteligar os vários setores e com que quanto um ponto da rede caisse ela continuasse a funcionar. Foi criado então a ARPANET, que para não ter que re-cabear todo o país usou o sistema de telefonia para a comunicação. No inicio inteligava setores de governo, posteriormente usado nas instituições de ensino e por último residencias, sendo chamada de Internet.

Com isso o sistema público de telefonia que antes era usado somente para voz, estava agora transportando também dados. Pórem o tráfego de dados cresceu mais e por volta de 1999 o número de bits de dados transferidos igualou o número de bits de voz. Em

2002 o volume do tráfego de dados ultrapassou o de voz em dez vezes. Com isso muitos operadoras de redes de comutação de pacotes se interessaram em transportar a voz sobre suas redes de dados. Com isso se populariza a telefonia IP ou voz sobre IP. (**??**)

8. Protocolos

Apesar de ser tornar popular somente nos últimos anos os protocolos utilizados em voz sobre IP e video, estão sendo elaborados desde a década de 1990. Usando o mesmo principio da pilha do protocolos TCP/IP cada protocolo deve ser aprovado por um orgão padronizador. Entre os protocolos mais utilizado para a comunição via VoIP podemos sitar o H.323, SIP, RTP e IAX.

9. *Real-Time Protocol/Real-Time Control Protocol*

O *Real-Time Protocol* é um protocolo que provê serviços de transporte fim-a-fim de dados para aplicações com restrições de tempo. A especificação do protocolo (**??**) inclui serviços como: identificação do tipo de dados transportados, números de seqüência, timestamping, e monitoração de envio. (**??**) (**??**)

Em aplicações de tempo real o RTP é usado em conjunto com o protocolo UDP, pois este gera menos overhead que o protocolo TCP, entretanto nada impede que o RTP use o mesmo. O RTP por si só não prove nenhuma garantia de entrega dos dados no tempo desejado, tão pouco qualidade de serviço, estas funcionalidades devem ser providas pelas camadas mais baixas da pilha de protocolos. O termo real-time significa apenas “baixo overhead”. (**??**)

O protocolo é composto por duas partes básicas, fortemente dependentes:

- O RTP propriamente dito, responsável por transmitir os dados com características de tempo real.
- O RTCP (*Real-Time Control Protocol*), utilizado para monitorar a qualidade de serviço e transportar informações sobre os participantes de uma sessão RTP.

Na telefonia IP, cada participante da ligação envia continuamente o áudio em pequenos trechos (20ms por exemplo) encapsulados num cabeçalho RTP, que por sua vez é encapsulado em um pacote UDP. Esses pacotes de áudio são enfileirados no destino para que possam ser tocados, com a devida correspondência de tempo, de modo a se manterem sincronizados. Isto é feito com o *buffer* de *playout* (ou *jitter buffer*), cujo propósito é absorver a variação de atraso, segurando os pacotes que chegam por tempo suficiente para garantir que eles sejam tocados continuamente. Quando um pacote chegar após seu tempo de *playout*, ele deverá ser descartado, havendo assim uma relação entre o atraso e a perda de pacotes RTP. (**??**)

10. Formato dos Pacotes RTP

Um pacote RTP, como mostrado na Figura **??**, é formado por quatro partes:

- O cabeçalho RTP.
- O cabeçalho extendido opcional.
- O cabeçalho de *payload* opcional.

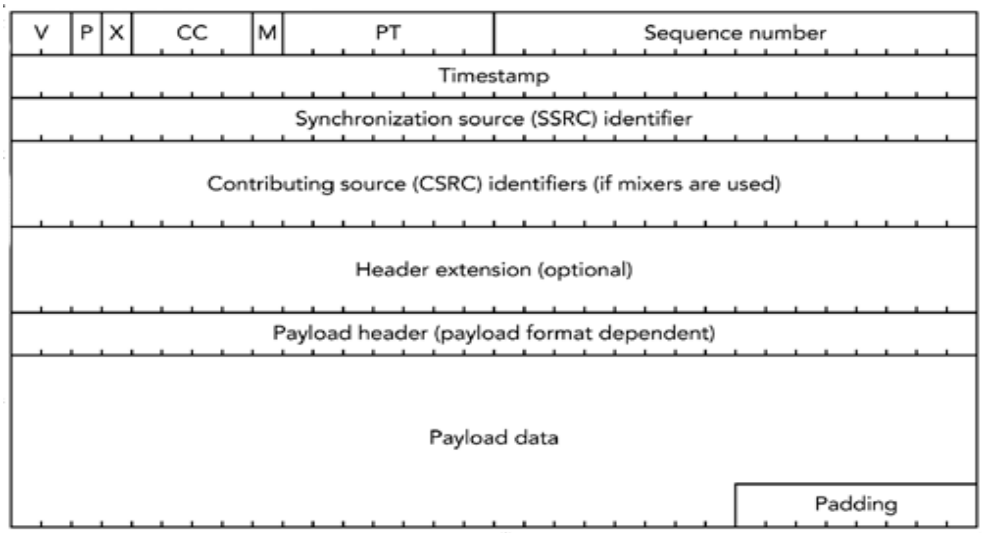

Figura 2. Estrutura de um Pacote *Real-Time Protocol*

- Os dados de *payload*.

O cabeçalho RTP contém 12 octetos e está presente em todos os pacotes. Os campos encontrados no cabeçalho são:

- **V - Versão:** Campo utilizado para identificar a versão do protocolo. Atualmente na versão 2.
- **P - *Padding*:** Utilizado para *padding* (enchimento). Se o bit esta ligado, o pacote contém um ou mais octetos de *padding* adicionais no final do pacote, os quais não fazem parte do da área de dados. Isso é importante para possibilitar, se desejado, o uso de algoritmos de criptografia.
- **X:** Utilizado para extensão. Se o bit está ligado, o cabeçalho fixo é seguido por uma extensão.
- **CC - Contador CSRC:** Contém o número de identificadores CSRC que seguem o cabeçalho fixo.
- **M - *Marker*:** Utilizado para marcação. A interpretação do marcador é definida por um perfil. Um perfil pode definir bits adicionais de marcação ou até mesmo excluir esses bits existentes no cabeçalho padrão. Geralmente esse bit é usado para sinalizar algum evento importante como, por exemplo, o término de um frame.
- **PT - Tipo de *payload*:** Esse campo identifica o formato da área de dados do RTP e sua interpretação pela aplicação. Alguns *payloads* são definidos no RFC 1889 e no *RFC Assigned Numbers*, como por exemplo, para o H.323 a referência é o H.225.
- **Número de Seqüencia:** O número de seqüencia é incrementado em uma unidade toda vez que um pacote de dados RTP é enviado e pode ser usado pelo destinatário para detectar perdas de pacotes e reordenação dos pacotes recebidos. O valor inicial do número de seqüencia é atribuído randomicamente.
- ***Timestamp*:** O *timestamp* reflete a amostra atual do primeiro octeto de dados RTP. A amostra é derivada de um relógio que é incrementada linearmente e monotonicamente, com o intuito de prover sincronização e cálculo de *jitter*.

- **Identificador de fonte de sincronização SSRC (*Synchronization Source Identifier*):** Esse campo identifica a origem da sincronização. Esse identificador é escolhido randonicamente, com o objetivo de evitar que hajam códigos duplicados em uma mesma sessão RTP.
- **Identificadores de fonte contribuinte CSRC (*Contributing Source Identifiers*):** Normalmente os dados RTP são gerados por apenas uma fonte, porém quando mais de uma fonte contribui para a criação dos dados passandos por um *mixer* ou tradudor seu identificador é colocado neste campo. Cada identificador é composto por 32 bits, que corresponde os SSRC de quem compribuiu. O tamanho do CSRC é definido pelo campo CC.

Além desse campos o cabeçalho RTP pode ser extendido a afim de prover maiores informaõoes caso seja requisitado por algum tipo de experimento ou aplicação. Eles são raramente usados, porém uma aplicação robusta deve ser capaz de identifica-los para que as informações do *payload* não sejam interpretadas de maneira erronea. O cabeçalho de *payload* assim como a extenção de cabeçalho provê informações adicionais sobre os dados do *payload* (**??**)

11. RTCP - *Real-Time Control Protocol*

O *Real-Time Control Protocol* (RTCP), provê relatórios periodicos sobre a qualidade, identificação de participantes e outras informações de descrição da fonte, notificação de mudanças de membros na sessão e informações necessárias para a sincronização dos streams de mídia. (**??**)

Todos os participantes de uma sessão RTP devem enviar esta tipo de pacote. Geralmente são enviados a cada 5 segundos, mas a medida que a quantidade de participantes aumenta, pode ser configurado para que o tempo de envio aumente também, fazendo assim com que a largura de banda não seja consumida demasiadamente por este tipo de pacote. (**??**) (**??**)

Cada sessão RTP é identificada por um endereço IP e um par de portas: uma para os dados RTP e outra para os dados RTCP. A porta RTP deve ser a mais alta enquanto a porta RTCP deve vir abaixo dessa, por exemplo se a comunicação RTP se da através da porta 10000, e fluxo de comunicação RTCP deve ser feito através da porta 10001. (**??**)

11.1. Formato dos pacotes RTCP

Cinco pacotes são definidos pela especificação do RTP: *receiver reports* (RR), *sender reports* (SR), *source description* (SDES), *membership management* (BYE) e *application-defined* (APP) (**??**). Este trabalho se aterá apenas aos pacotes *receiver reports* e *sender reports*.

O cabeçalho para todos os pacotes RTCP é mostrado na Figura **??** e é composto pelos campos:

- **V - Versão:** Campo contendo a versão do RTP.
- **P - *Padding*:** Utilizado para *padding* (enchimento). Se o bit esta ligado, o pacote contém um ou mais octetos de *padding* adicionais no final do pacote, os quais não fazem parte do da área de dados. Isso é importante para possibilitar, se desejado, o uso de algoritmos de criptografia.

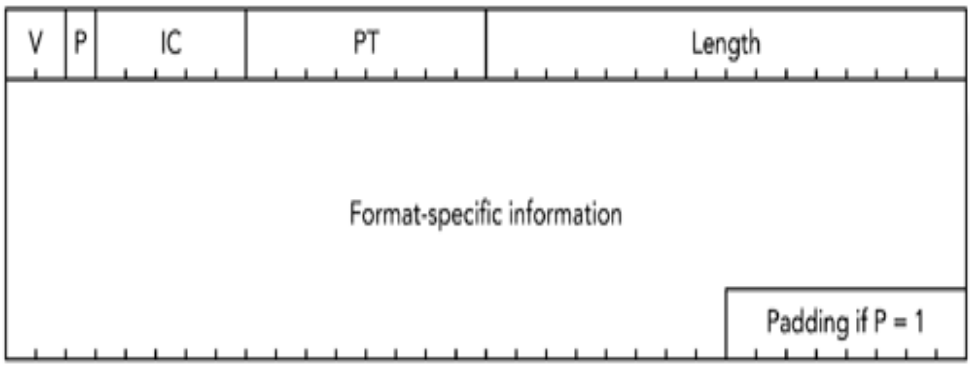

Figura 3. Estrutura de um Pacote *Real-Time Control Protocol*

- **IC - *Item Count*:** Indica quantos itens o relatório possui.
- **PT - *Packet Type*:** Indica qual é o tipo de pacote, como foi mostrato o RTCP define cinco tipos de pacotes diferentes. Para pacotes do tipo *Sender Report* o PT terá o valor 200, *Receiver Report* 201, *Source Description* 202, *Membership Management* 203 e *Application-Defined* 204.
- ***Length* (Tamanho):** Tamanho total do pacote.

Pacotes RTCP nunca são enviados individualmente, eles são agrupados em dois ou mais pacotes fomando pacotes compostos. Como mostrado na Figura **??** cada pacote composto é encapsulado em um único pacote da camada mais baixa para transporte. (**??**)

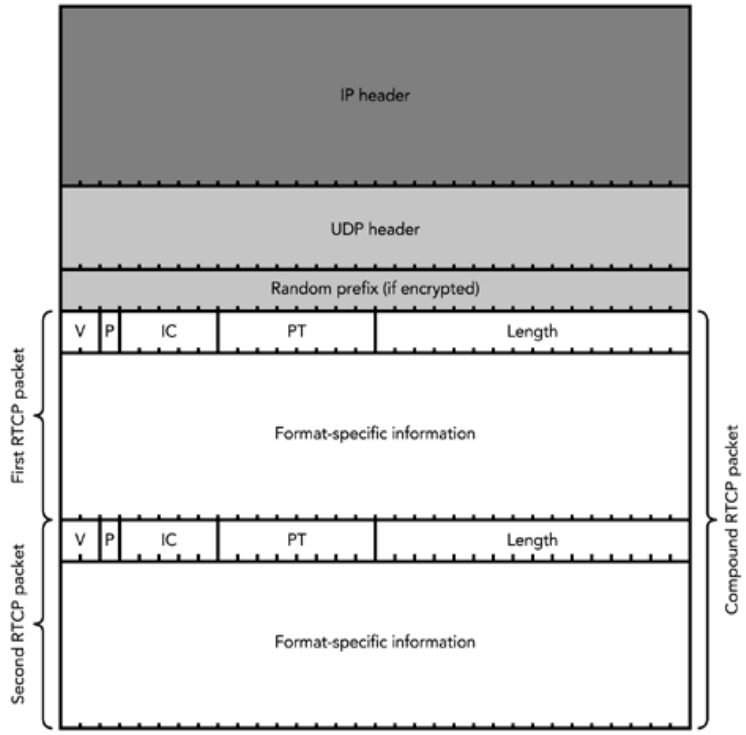

Figura 4. Pacote composto RTCP

11.2. RTCP RR - *Receiver Reports*

Um dos principais usos do RTCP é a troca de informações de qualidade o que é feito através do pacote *Receiver Report* (RR), que é enviado por todos os participantes. A Figura **??** mostra a estrutura de um pacote RTCP RR.

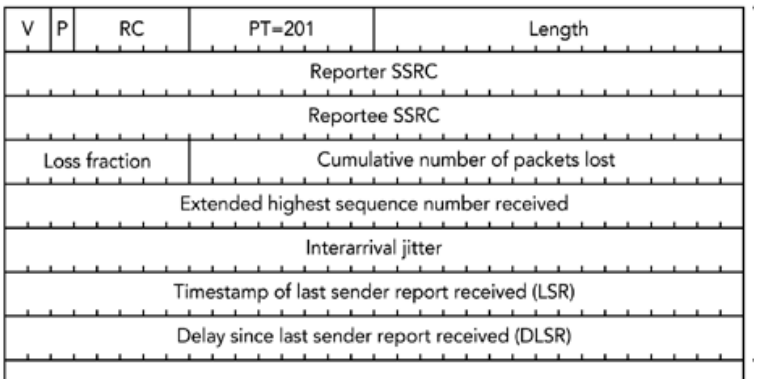

Figura 5. Estrutura pacote RTCP *Receiver Report*

Abaixo segue a descrição dos campos do pacote RR:

- ***Reporter SSRC*:** SSRC do participante que esta enviando o pacote.
- ***Reportee SSRC*:** Identifica de qual participante as estatísticas de qualidade do relatório esta se referindo.
- **Fração de Perda:** Valor do resultado dos pacotes perdidos / pacotes esperados multiplicando por 256.
- **Pacotes perdidos acumulados:** Número de pacotes perdidos desde o inicio da sessão.
- **Valor estendido do mais alto número de seqüencia recebido:** Os 16 bits mais significativos contêm o número de ciclos de seqüencia (isto é, o número de vezes que a contagem atingiu o valor máximo e retornou a zero) e os últimos 16 bits contêm o mais alto número de seqüencia recebido em um pacote de dados RTP proveniente dessa fonte (mesmo SSRC).
- ***Jitter* entre chegadas:** É a diferença entre o espaço de tempo entre a chegada de um par de pacotes, comparado com o *timestamp* desses pacotes, calculado da seguinte maneira: $D_{(i,j)} = (R_j - S_j) - (R_i - S_i)$, onde S_i é o *timestamp* RTP do pacote i e R_i é tempo de chegada do pacote i. Toda vez que um pacote for recebido se calcula o D e desda forma o *jitter* pela fórmula $J_n = \frac{J_{n-1}+(D-J_{n-1})}{16}$.
- **Último *timestamp* SR (LSR):** Os 32 bits do meio do *timestamp* NTP do último *Sender Report*.
- **Atraso desde o último SR (DLSR):** Tempo desde que o último pacote SR chegou. Assim como o LSR é representado na forma de *timestamp* NTP compacta.

Com esse campos é possível fazer uma serie de estimativas da qualidade da rede. Com o campo LSR e DSLR o transmissor pode calcular o tempo de ida-e-volta (*round-trip time* (RTT)) entre ele e o receptor. A Figura **??** mostra como é possível fazer este calculo.

O tempo de ida-e-volta é importante em aplicações interativas pois o atrasa atrapalha a interatividade. Estudos mostram a dificuldade de conduzir um conversa quando o

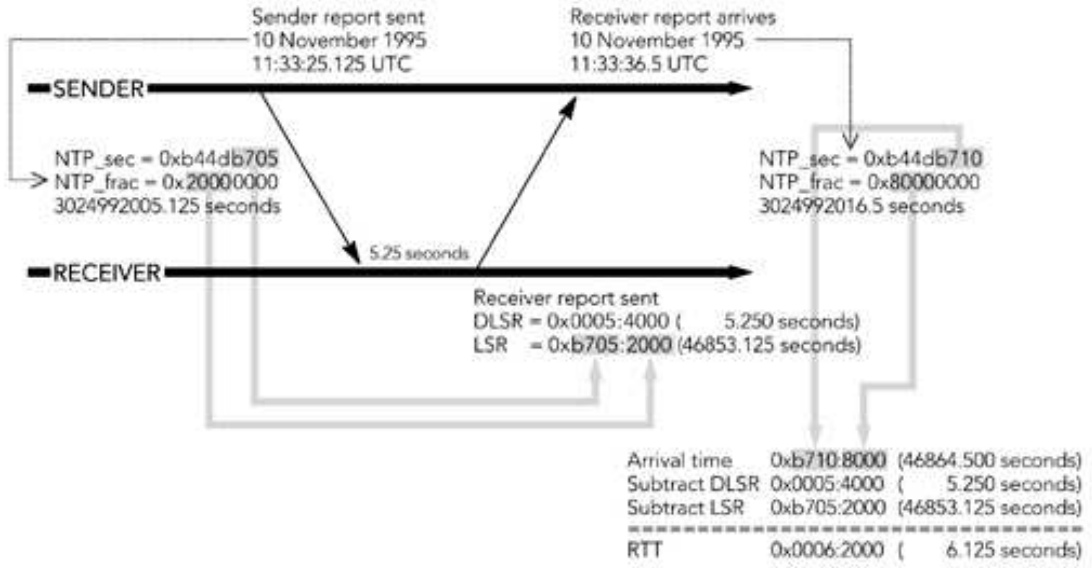

Figura 6. Calculo do tempo de ida-e-volta *(round-trip time)*

total da ida-e-volta começa a exceder o tempo de 300 milisegundos. Um transmissor ao verificar o valor de ida-e-volta pode tentar otimizar a codificaçãos da mídia transportada, gerando pacotes com menor dados, reduzindo assim o tempo de analise do pacote para transmissão. (**??**)

O campo de *jitter* pode ser usado para detectar o surgimento de congestionamentos na rede. Um aumento repentino no *jitter*, muitas vezes antecede o início de perdas de pacote. (**??**)

12. RTCP SR - *Sender Report*

Além dos pacotes RR enviados pelos receptores, o RTCP define os pacotes *Sender Report* que são enviados pelos prarticipantes que enviaram recentemente algum tipo de dado na sessão. Um pacote SR é identico a um RR, com o acrescimo de alguns campo como pode ser visto na Figura **??**.

Os campos adicionais ficam após o campo de identificação SSRC e são eles:

- **NTP *timestamp*:** Valor no formato NTP que mostra o tempo que o pacote SR foi enviado.
- **RTP *timestamp*:** Valor no mesmo formato que o NTP *timestamp* poderem medido apartir da taxa de amostragem da mídia.
- **Contagem de pacotes enviados:** Quantidade de pacotes que foi enviado desde o inicio da sessão.
- **Contagem de octetos enviados:** Quantidade de octetos contidos no pacotes enviados.

Após esse campos adicionais pode-se encontrar os campo de um pacote RR comum. (**??**)

RTCPMoni - Ferramenta Proposta

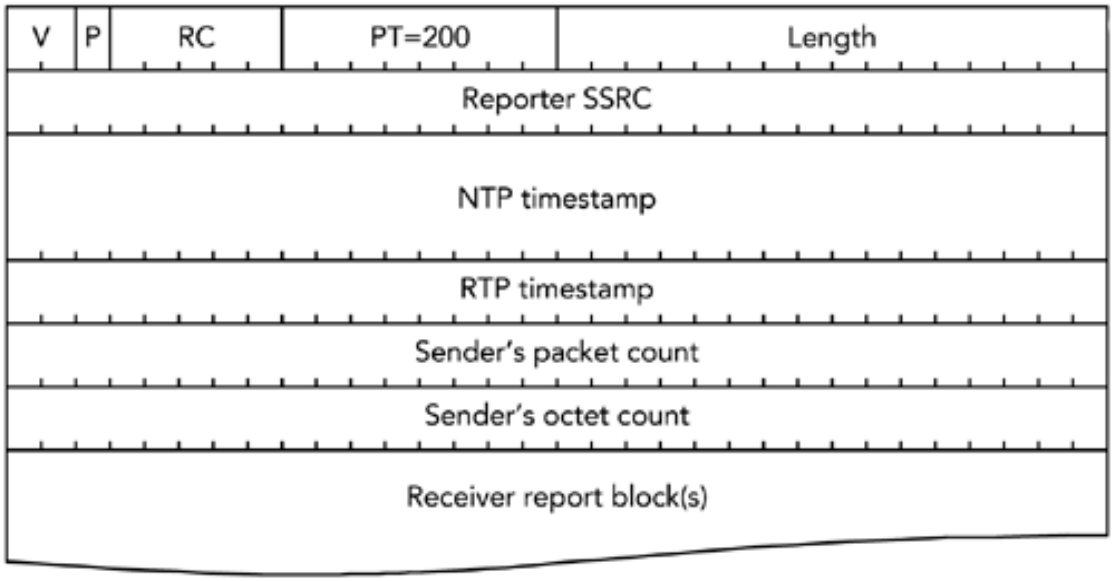

Figura 7. Estrutura do pacote RTCP *Sender Report*

Nas analises iniciais deste trabalho, notou-se um escassez de ferramentas que pudessem auxiliar o administrador de rede a identificar alguns problemas com a rede de voz sobre IP. A ferramentas como o software Wireshark, capazes de capturar o tráfego em uma interface de rede qualquer e mostrar para o administrador os pacotes RTCP, responsável por informar como esta a qualidade como foi mostrado no Capítulo 5. Entretando esta ferramenta é limitada no sentido de extensibilidade, se dedicando apenas a captura dos pacotes.

Com isso foi proposto a desenvolver uma ferramenta que fosse capaz de realizar o trabalho dos analisadores de rede, porém voltada especificamente para voz sobre IP, capturando somente os pacotes RTCP e dando a possibilidade para que novos recursos sejam acrescentados. Nas próximas sessões estará sendo apresentada como se sucedeu o desenvolvimento da ferramenta batizada de RTCPMoni.

13. Arquitetura

13.1. Jpcap

Para realizar a captura dos pacotes que tráfegam na rede foi necessário a utilização de um biblioteca especifica para está finalidade. Como a ferramenta RTCPMoni esta sendo códificada em Java, se fez juz a utilização de uma biblioteca de captura escrita em Java, portando foi escolida a Jpcap.

A biblioteca Jpcap é baseada na libcap escrita em C. Ela fornece os mesmo recursos da libcap, tais como a captura e envios de pacotes de rede. Entre os pacotes que ela é capaz de verificar estão quadros Ethernet, pacotes IPv4, IPv6, TCP, UDP, ICMPv4, e algumas aplicações de terceiros estendem os tipos de pacotes.

Ao capturar um pacote a Jpcap fornece um objeto Java correspondente a ele, se for capturado um pacote TCP ela criara um objeto do tipo jpcap.TCPPacket, se ela capturar

um pacote na qualquer não pode verificar o tipo será criado um objeto generico do tipo jpcap.Packet, fornecendo assim recursos para que os desenvolvedores possam tratar de maneira apropriada qualquer tipo de tráfego.

13.2. RTCPMoni

A modelagem da ferramenta RTCPMoni pode ser vista na Figura **??**. Ela consiste de uma classe Main.java que tem por objetivo ler os parâmentros passados pelo usuário e fazer a chamada de métodos para o inicio da execução do *software*. A classe Capture.java tem por finalidade realizar a capturas dos pacotes que tráfegam pela interface de rede selecionada e verificar se o pacote capturado contém dados RTCP. Em caso positivo as informações contidas no pacote são impressas para o usuário no *console* e gravadas em uma base de dados atravéz da classe DBMysql.java. A estrutura da tabela na base de dados é mostrada na Tabela **??**

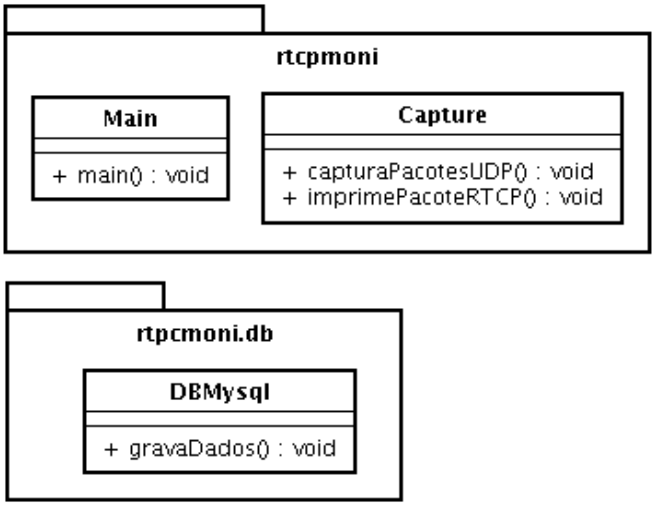

Figura 8. Classes da ferramenta RTCPMoni

tempoCaptura	DOUBLE
SSRC	VARCHAR(15)
jitter	INT
perdas	INT
RTT	DOUBLE

Tabela 1. Estrutura da tabela da base de dados RTCPMoniDB

14. Cenário

O cenário em que se sucedeu os testes da ferramenta RTCPMoni esta representada na Figura **??**. Como pode ver visto foi utilizado quatro equipamentos, sendo que dois foi instalado o sistema operacional GNU/Linux Ubuntu para que pude-se ser feita a instalação do PABX Asterisk. As duas máquinas restantes foi instalado o sistema operacional Windows XP para que servi-se de base para instalação dos *softphones* para a realização das

ligações. Com isso todo o tráfego de voz, juntamente com os dados RTCP, passa pelo dois PABX Asterisk.

O cenário de teste foi projetado desta maneira para se parecer com um ambiente real em que os dois servidores com o Asterisk estão dentro da rede conhecidade e de responsabilidade do administrador, enquante os dois *softphones* estão instaladas em máquinas que se conectam nos servidores através da Internet, com isso o monitoramente se faz entre os dois Asterisk.

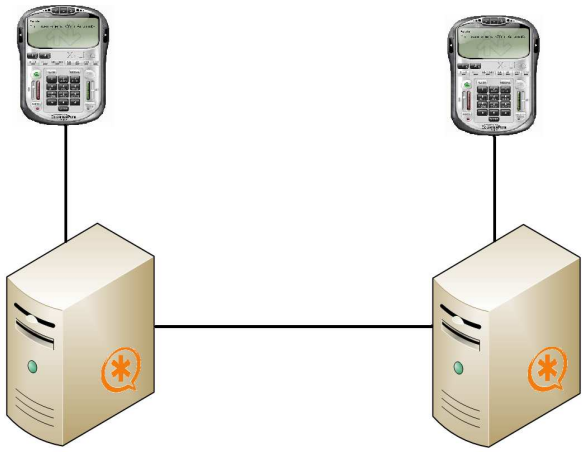

Figura 9. Cenário de Testes

15. Configurações

Como mencionado na sessão anterior foi utilizado duas máquinas com o PABX Asterisk instalado na qual foi configurado, no Asterisk A um ramal SIP referente ao *softphone* A e mais 1 ramal SIP para servir de tronco de comunicação entre as Asterisk. Foram feitas as mesmas configurações no Asterisk B. O plano de discagem dos Asterisk tem como configuração a seguinte linha: *exten => 2000,1,Dial(SIP/2:12345@172.16.0.101/2000)*, sendo feita apenas a mudança do ramal chamada entre um Asterisk e outro. Para maiores informação de configurações verificar os anexos contendo os códigos-fonte e arquivos gerais.

16. Monitoramento

Após terminadas as instalações e configurações necessárias, foram feitas uma serie de ligações entre os *softphones* com a ferrementa RTCPMoni rodando em um dos Asterisk. Terminadas as ligações testes uma parcela da saída da ferramenta pode ser vista na Figura **??**.

Como pode ser visto a ferramenta é capaz de mostrar as informações do um pacote RTCP de maneira amigável e realiza o calculo do tempo de ida-e-volta entre os servidores. Como já mencionado essas informações são salvas em uma base de dados para posteriores consultas.

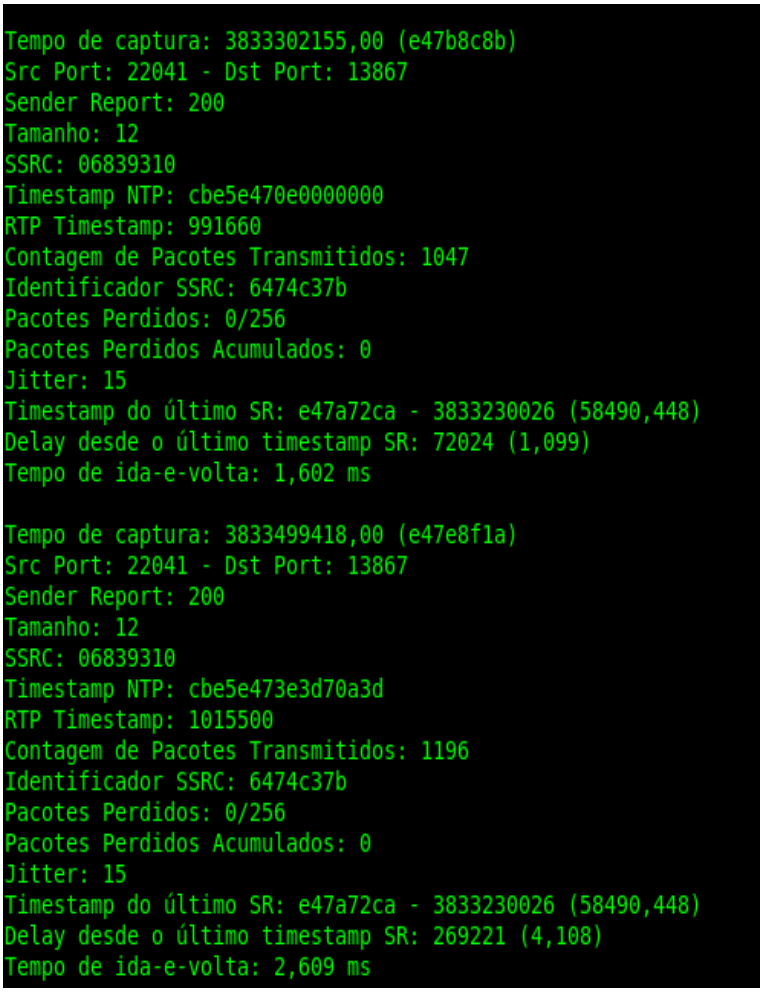

Figura 10. Saída da ferramenta RTCPMoni

Referências

aaaa.

Fortes, D. (2005). A explosão do voip. *Revista Info*.

Melo, E. T. L. (2001). Qualidade de serviço em redes ip com diffserv: Avaliação atravás de medições. Master's thesis, Universidade Federal de Santa Catarina.

Tananbaum, A. S. (2003). *Redes de Computadores*. Campus, 4º edition.

Classe Main.java:

```
package rtcpcap;

public class Main {

     public static void main(String args[]) {
          Capture capt = new Capture(Integer.parseInt(args[0]));
          capt.capturaPacotesUDP();
     }

}
```

Classe Capture.java:

```
package rtcpcap;

import java.io.IOException;
import org.apache.commons.net.ntp.TimeStamp;
import jpcap.JpcapCaptor;
import jpcap.NetworkInterface;
import jpcap.packet.Packet;
import jpcap.packet.UDPPacket;

public class Capture {

     private JpcapCaptor captor;
     private Packet p;
     private NetworkInterface[] devices;

     public Capture(int device) {
          devices = jpcap.JpcapCaptor.getDeviceList();
          try {
               System.out.println(devices[device].name);
               captor = JpcapCaptor.openDevice(devices[device],
65535, false, 20);
          } catch (IOException e) {
               e.printStackTrace();
          }
     }

     public void capturaPacotesUDP() {

          System.out.println("Iniciando captura de pacotes UDP");

          try {
               captor.setFilter("udp", true);
               while(true) {
//                  String tempoNTP =
TimeStamp.getCurrentTime().toString();
//
//                  char[] ntpComp1 = tempoNTP.toCharArray();
```

```
//                  char ntpComp2[] = {ntpComp1[4], ntpComp1[5],
ntpComp1[6], ntpComp1[7],
//                                          ntpComp1[9],
ntpComp1[10], ntpComp1[11], ntpComp1[12] };
//
//                  Long tempoCapturaL = Long.parseLong(new
String(ntpComp2),16);
//                  double tempoCaptura = tempoCapturaL;

                    p = captor.getPacket();

                    if(p instanceof UDPPacket) {

                         this.imprimePacoteRTCP((UDPPacket)p);//,
tempoCaptura, new String(ntpComp2));

                    }
               }

          } catch (IOException e) {

               e.printStackTrace();
          }
     }

     private void imprimePacoteRTCP(UDPPacket udp) { //, double
tempoCaptura, String tempoCapturaS) {

          Long rtcpType;
          String ssrc;
          String timestampNTP;
          Long timestampRTP;
          Long pacotesTran;
          String idSSRC;
          Long pacotesPerdidos;
          Long jitter;
          double lsr;
          Long lsrL;
          double dlsr;
          Long dlsrL;
          double tempoCaptura;
          String tempoCapturaS;


          if( (udp.src_port >= 10000) || (udp.dst_port >= 10000))
{

                    byte dados[] = udp.data;
                    if( ((int)(dados[1]&0xff)== 200)) {

                         String tempoNTP =
TimeStamp.getCurrentTime().toString();
```

```
                        char[] ntpComp1 = tempoNTP.toCharArray();
                        char ntpComp2[] = {ntpComp1[4],
ntpComp1[5], ntpComp1[6], ntpComp1[7],
                                                    ntpComp1[9],
ntpComp1[10], ntpComp1[11], ntpComp1[12] };

                        Long tempoCapturaL = Long.parseLong(new
String(ntpComp2),16);
                        tempoCaptura = tempoCapturaL;

                        tempoCapturaS = new String(ntpComp2);

                        System.out.printf("\nTempo de captura: %.
2f (%s)\n", tempoCaptura, tempoCapturaS);
                        System.out.printf("Src Port: %d - Dst
Port: %d\n", udp.src_port, udp.dst_port);
                        String d = Integer.toHexString((int)
(dados[1]&0xff));
                        String e;
                        rtcpType = Long.parseLong(d,16);
                        System.out.printf("Sender Report: %s\n",
rtcpType);


                        e =
Integer.toHexString((int)dados[2]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[3]&0xff);
                        d += e.length() == 1 ? "0" + e : e;

                        System.out.printf("Tamanho: %s\n",
Long.parseLong(d, 16));

                        e =
Integer.toHexString((int)dados[4]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[5]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[6]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[7]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        ssrc = d;
                        System.out.printf("SSRC: %s\n", ssrc);

                        e =
Integer.toHexString((int)dados[8]&0xff);
```

```
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[9]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[10]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[11]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[12]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[13]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[14]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[15]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        timestampNTP = d;
                        System.out.printf("Timestamp NTP: %s\n",
timestampNTP);

                        e =
Integer.toHexString((int)dados[16]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[17]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[18]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[19]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        timestampRTP = Long.parseLong(d, 16);
                        System.out.printf("RTP Timestamp: %s\n",
timestampRTP);

                        e =
Integer.toHexString((int)dados[20]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[21]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[22]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
```

```
Integer.toHexString((int)dados[23]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        pacotesTran = Long.parseLong(d, 16);
                        System.out.printf("Contagem de Pacotes
Transmitidos: %s\n", pacotesTran);

                        e =
Integer.toHexString((int)dados[28]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[29]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[30]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[31]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        idSSRC = d;
                        System.out.printf("Identificador SSRC:
%s\n", idSSRC);

                        d =
Integer.toString((int)dados[32]&0xff);

                        System.out.printf("Pacotes Perdidos:
%s/256\n", d);

                        e =
Integer.toHexString((int)dados[33]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[34]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[35]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        pacotesPerdidos = Long.parseLong(d, 16);
                        System.out.printf("Pacotes Perdidos
Acumulados: %s\n", pacotesPerdidos);

                        e =
Integer.toHexString((int)dados[40]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[41]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[42]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[43]&0xff);
```

```
                        d += e.length() == 1 ? "0" + e : e;
                        jitter = Long.parseLong(d, 16);
                        System.out.printf("Jitter: %s\n",
jitter);


                        e =
Integer.toHexString((int)dados[44]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[45]&0xff);
                        d += e.length() == 1 ? "0" + e : e;

                        e =
Integer.toHexString((int)dados[46]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[47]&0xff);
                        d += e.length() == 1 ? "0" + e : e;

                        lsrL = Long.parseLong(d,16);

                        lsr = lsrL;
                        System.out.printf("Timestamp do último
SR: %s - %d (%.3f)\n", d, lsrL , lsr/65536);

                        e =
Integer.toHexString((int)dados[48]&0xff);
                        d = e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[49]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[50]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        e =
Integer.toHexString((int)dados[51]&0xff);
                        d += e.length() == 1 ? "0" + e : e;
                        dlsrL = Long.parseLong(d, 16);
                        dlsr = dlsrL;

                        System.out.printf("Delay desde o último
timestamp SR: %d (%.3f)\n" ,dlsrL , dlsr/65536);

                        //double tempoIdaVolta =
(tempoCaptura/65536) - (dlsr/65536) - (lsr/65536);

                        double tempoIdaVolta = tempoCaptura -
dlsr - lsr;
                        System.out.printf("Tempo de ida-e-volta:
%.3f ms\n", (tempoIdaVolta/65536)/0.001);
```

```
                    }
          }
     }
}
```

Classe DBMysql.java:

```
package rtcp.db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Hashtable;

public class DBMysql {

     private String driver = "com.mysql.jdbc.Driver";
     private String url;
     private String baseDados;
     private String usuario;
     private String senha;
     private Connection conn;

     public DBMysql(String url, String baseDados, String usuario,
String senha) {
          super();
          this.url = url;
          this.baseDados = baseDados;
          this.usuario = usuario;
          this.senha = senha;
     }

     public void iniciaConexao() {
          System.out.println("Iniciando conexao com banco");
          try {
               Class.forName(driver);
               conn = DriverManager.getConnection("jdbc:mysql://"
+ url + "/" + baseDados ,usuario,senha);
               System.out.println("Conexão efetuada com
sucesso!");
          } catch (ClassNotFoundException e) {
               System.out.println("Erro: Driver MySQL nao
encontrado!");
               e.printStackTrace();
          } catch (SQLException e) {
               System.out.println("Erro: Não foi possível
conectar-se ao banco!");
               e.printStackTrace();
```

```
          }

     }

     public void finalizaConexao() {
          try {
               conn.close();
               System.out.println("Conexão com banco finalizada
com sucesso!");
          } catch (SQLException e) {
               System.out.println("Erro: Não foi possivel
finalizar a conexão com o banco!");
               e.printStackTrace();
          }
     }

     public void Gravar() {

     }

     public ArrayList getResultados(String[] dados, String tabela,
String where) {
          ArrayList retorno = new ArrayList();
          Hashtable resultado;
          String sql = "SELECT ";
          for (int i = 0; i < dados.length -1; i++) {
               sql += dados[i] + ", ";
          }
          sql += dados[dados.length -1] + " FROM " + tabela +
where;
          System.out.println(sql);
          System.out.println("Iniciando consulta!");
          try {
               Statement stm = conn.createStatement();
               ResultSet rs = stm.executeQuery(sql);
               while(rs.next()) {
                    resultado = new Hashtable();
                    for (int i = 0; i < dados.length; i++) {
                         resultado.put(dados[i],
rs.getString(dados[i]));
                    }
                    retorno.add(resultado);
               }
          } catch (SQLException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }
          return retorno;


     }
```

```
}
```