UNIVERSIDADE FEDERAL DA PARAIBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

Gerência de Recursos em Redes Compartilhadas com Integração de Serviços - uma Proposta e Implementação

Por

José Eduardo Malta de Sá Brandão

Campina Grande, 30 de Agosto de 1996

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

José Eduardo Malta de Sá Brandão

Gerência de Recursos em Redes Compartilhadas com Integração de Serviços - uma Proposta e Implementação

Dissertação apresentada ao curso de Mestrado em Informática da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de Mestre

Área de Concentração: Redes de Computadores

Orientador: Joberto Sérgio B. Martins

Campina Grande, 30 de Agosto de 1996

B817g Brandao, Jose Eduardo Malta de Sa
Gerencia de recursos em redes compartilhadas com integracao de servicos : uma proposta e implementacao / Jose Eduardo Malta de Sa Brandao. - Campina Grande, 1996.
182 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Computadores 2. Redes Compartilhadas 3. Dissertacao I. Martins, Joberto Sergio Barbosa, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.7(043)

GERENCIAMENTO DE RECURSOS EM REDES COMPARTILHADAS COM INTEGRAÇÃO DE SERVIÇOS - UMA PROPOSTA E IMPLEMENTAÇÃO.

JOSÉ EDUARDO MALTA DE SÁ BRANDÃO

DISSERTAÇÃO APROVADA EM 30.08.96

PROF. JOBERTO SÉRGIO BARBOSA MARTINS, Dr.
Presidente

PROF. MARIA IZABEL CAVALCANTI CABRAL, Ph.D
Examinador

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Examinador

PROF. MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D
Examinador

CAMPINA GRANDE - PB

à Helana, Izabel e ao neném, pelo amor, paciência e compreensão em todos os momentos.

Agradecimentos

- Agradeço aos meus pais, avós e irmã, pela educação e amor que sempre me deram e continuam dando.
- Agradeço ao meu orientador, professor Joberto, pelo apoio e confiança.
- Agradeço à Empresa Brasileira de Pesquisa Agropecuária (EMBRAPA), pela oportunidade de realizar o curso de Mestrado.
- Agradeço também aos meus chefes e colegas do CENARGEN e do CNPA, que tanto me apoiaram na realização deste trabalho.
- Não poderia deixar de esquecer Artur, Gerusa, Milena e Patrícia, que tão bem me acolheram em Campina Grande.
- Finalmente, agradeço a todos os meus colegas e amigos, que de forma direta ou indireta contribuíram para este resultado.

Resumo

A utilização da Integração de Serviços nas redes de computadores baseadas no IP requer a introdução de alguns mecanismos de controle. Entre este mecanismos estão o protocolo de reserva de recursos e o mecanismo de Controle de Tráfego. Neste documento é proposta e implementada uma solução para o mecanismo de Controle de Tráfego. Esta solução foi projetada para funcionar em redes locais compartilhadas (como a Ethernet), em conjunto com o protocolo de reserva de recursos RSVP (*Resource ReSerVation Protocol*). Como o controle dos recursos do enlace é especialmente complexo nas redes compartilhadas, também foi desenvolvido e implementado um mecanismo de gerenciamento de recursos, o DCRP (*Distributed Control Reservation Protocol*). Este mecanismo funciona de forma distribuída, controlando os recursos de rede, assim como a banda utilizada por todas as máquinas da rede e o atraso no enlace. Além das propostas dos mecanismos e suas implementações, também são apresentados os conceitos básicos pertinentes à Integração de Serviços aos mecanismos relacionados e o procedimento de teste das propostas apresentadas.

Abstract

Integrated Services over IP-based networks implies in using control mechanisms. Among these mechanisms we have the resource reservation protocols and algorithms for Traffic Control. In this thesis work is proposed and implemented a solution for the Traffic Control problem. The proposed mechanism was designed to operate over shared networks (e.g. Ethernet), with the RSVP (Resource ReSerVation Protocol). The resource control in shared networks is a complex task. For this reason, it was designed and implemented a distributed resource management mechanism (DCRP - Distributed Control Reservation Protocol). This mechanism will control the bandwidth used for all hosts in the network and the network delay. In addition, in this document will introduce the basic concepts of Integrated Services model, related mechanisms and test procedures.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1
Introdução

Observa-se há algum tempo um grande esforço de pesquisadores e empresas em oferecer serviços de rede mais confiáveis do que os simples serviços de transmissão de datagramas, atualmente utilizados na Internet e em diversas organizações que utilizam o IP (*Internet Protocol*) [Com91] em redes corporativas e Intranets. Estes novos serviços visam principalmente oferecer um melhor suporte de rede para aplicações de tempo real, como a transmissão de vídeo, transmissão de voz e aplicações que são sensíveis à perda de pacotes e ao atraso. Ou seja, é necessário garantir a Qualidade de Serviço (*QoS - Quality of Service*) que estas aplicações precisam.

A utilização de redes e/ou tecnologias distintas para cada tipo de serviço, como ocorre com a telefonia, dados, TELEX, transmissão de TV e demais redes dedicadas, já não atende às necessidades atuais. Surge então o conceito de Integração de Serviços, onde uma mesma rede suporta os mais variados tipos serviços, de forma transparente ao usuário.

Devido à sua relevância, a Integração de Serviços utilizando IP é hoje um dos principais campos de pesquisa. Há atualmente três grupos de trabalho, coordenados pelo IETF [HC94], pesquisando o suporte para a Integração de Serviços. O grupo Int-Serv pesquisa a Integração de Serviços e a definição de classes de serviços para a Internet. O grupo RSVP

pesquisa o desenvolvimento do RSVP (*Resource ReServation Protocol*) [ZDESZ93]. O grupo ISSLL (*Integrated Services over Specific Link Layers*) pesquisa o uso de tecnologias de rede, para o suporte à Integração de Serviços.

A Integração de Serviços em IP não tem a pretensão de substituir outras tecnologias que estão sendo desenvolvidas para suportar aplicações de tempo real, como o ATM [Mon94]. Mas, pretende-se que o modelo de Integração de Serviços venha a funcionar em conjunto com estas novas tecnologias. Diversos pesquisadores envolvidos no desenvolvimento do ATM, por exemplo, também trabalham nas pesquisas de Integração de Serviços IP. Sugestões feitas pelos grupos de pesquisa em Integração de Serviços freqüentemente tem sido aceitas pelos grupos de pesquisa do ATM e vice-versa. O objetivo disso é que as aplicações que usam QoS em ATM possam também ser utilizadas sobre IP, de forma transparente ao usuário.

Para executar a Integração dos Serviços, deve-se garantir que a Qualidade de Serviço será respeitada durante todo o caminho que os dados de uma aplicação irão percorrer entre o transmissor (origem) e o receptor (destino). Para isso, estão sendo propostos protocolos de reserva de recursos, que irão efetuar a reserva de recursos de rede, ao longo do caminho que será percorrido pelo fluxo de dados da aplicação. Entre este protocolos está o RSVP, que vem sendo desenvolvido para suportar a Integração de Serviços.

As máquinas dos usuários (*hosts*) e os nós intermediários precisam ser capazes de efetuar a reserva de recursos nos enlaces por onde os dados serão transmitidos, de acordo com a Qualidade de Serviço solicitada pelo usuário, executando todo o controle necessário para garantir o serviço oferecido. O mecanismo que executa esta função é denominado Controle de Tráfego.

O mecanismo de Controle de Tráfego deve: computar as solicitações de reserva de recursos através de um mecanismo de Controle de Admissão; separar os pacotes pertencentes ao tráfego convencional (*best-effort*) dos pacotes pertencentes aos fluxos de dados que solicitaram reserva, identificando cada um destes fluxos através de um mecanismo de Classificação de Pacotes; e transmitir os pacotes com a Qualidade de Serviço solicitada, através de um mecanismo de Escalonamento de Pacotes.

O RSVP já possui algumas implementações acadêmicas e comerciais (Seção 3.6). Estudos vem sendo realizados para a inserção do modelo de Integração de Serviços nas atuais tecnologias de rede. Porém, ainda não há mecanismo de Controle de Tráfego disponível para a realização de testes do modelo de Integração de Serviços.

Neste documento é proposto e implementado um mecanismo de gerenciamento de recursos e Controle de Tráfego para redes compartilhadas, dentro do conceito de Integração de Serviços, capaz de receber as solicitações de reserva através do RSVP.

O mecanismo proposto é direcionado às redes locais compartilhadas do tipo Ethernet, mas pode ser estendido a outras tecnologias de rede. A escolha inicial de redes do tipo Ethernet se deve ao seu baixo custo e à grande base instalada no mundo.

Para executar o gerenciamento de recursos, é proposto um protocolo chamado DCRP (*Distributed Control Reservation Protocol*), que fará o controle dos recursos de rede, como o atraso experimentado pelos pacotes na rede e a banda de transmissão utilizada por cada máquina.

Para o Controle de Tráfego são propostas soluções simples, visando uma melhor eficiência da implantação. Sua implementação envolve a adaptação do mecanismo atual de funcionamento do IP. Esta adaptação deve ser a mais simples possível, alterando e/ou incluindo apenas o mínimo necessário, para garantir o funcionamento do mecanismo, sem descaracterizar o modelo IP.

A fim de reduzir os custos do trabalho e garantir uma boa performance, utilizaram-se microcomputadores do tipo PC, baseados em microprocessadores Intel, rodando o sistema operacional UNIX com TCP/IP nativo.

O sistema UNIX selecionado foi o FreeBSD [Fre96]. Este sistema foi escolhido por disponibilizar todos os seus arquivos fontes e seguir o padrão BSD (*Berkeley Software Distribution*), que é bastante difundido, possui uma excelente estrutura de bibliotecas e é muito documentado. Outro ponto que pesou bastante na escolha foi o fato do sistema operacional já possuir suporte para o RSVP e já existir implementação do RSVP portada para este sistema.

Este documento é dividido em 8 (oito) capítulos. O Capítulo 2 caracteriza a Integração de Serviços e as aplicações que devem fazer uso deste modelo, descreve os componentes necessários para a execução da Integração de Serviços e apresenta a definição dos serviços propostos para a Internet (extensível a outras tecnologias).

O Capítulo 3 descreve o RSVP, com seu mecanismo de funcionamento, seus pré-requisitos e as interfaces com alguns mecanismos com os quais ele irá cooperar. Também são citadas as implementações em curso e discutida a interface de uma destas implementações com o mecanismo de Controle de Tráfego e que é utilizada no trabalho.

No Capítulo 4 é proposto um mecanismo para o Controle de Admissão e um protocolo de gerenciamento de recursos (DCRP). A implementação destes mecanismos é apresentada e discutida ao longo do texto.

Uma proposta de Mecanismo de Classificação de Pacotes é vista no Capítulo 5, junto com a sua implementação e as adaptações necessárias ao IP para suportar este mecanismo.

O Capítulo 6 apresenta uma proposta para o mecanismo de Escalonamento de Pacotes, sua implementação e o seu posicionamento dentro do *kernel* do Sistema Operacional.

No Capítulo 7 são apresentados os resultados dos testes práticos realizados para a avaliação dos mecanismos propostos e implementados.

Finalmente, no Capítulo 8 tem-se a conclusão deste trabalho.

No início dos capítulos que tratam de implementação, há uma tabela indicando a localização dos arquivos utilizados ou citados no capítulo. Esta tabela é composta da estrutura de arquivos do Sistema Operacional e a descrição do arquivo.

As implementações são ilustradas na forma de figuras, onde é colocado o corpo da rotina, identificado pelo nome do arquivo que o comporta e o nome da rotina ou função. As linhas da rotina são numeradas em ordem crescente, a partir do início do arquivo. A Figura 1.1 dá uma idéia disso.

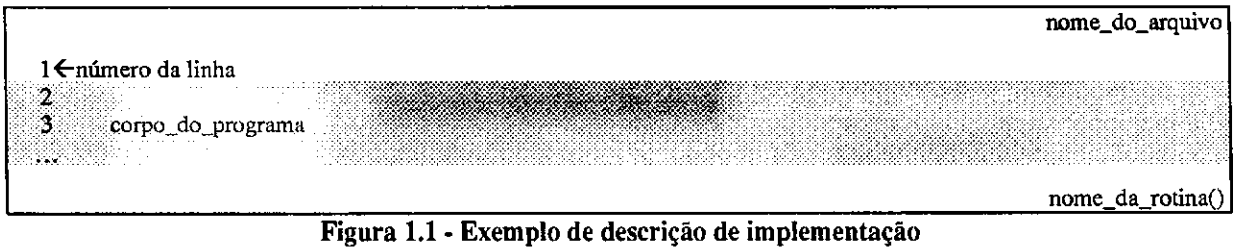


Figura 1.1 - Exemplo de descrição de implementação

Para facilitar a identificação de citações das implementações no texto, a representação de rotinas, estruturas e programas será feita utilizando as seguintes regras:

- Programas: **negrito**;
- Rotinas e funções: **negrito()**;
- Definições e constantes: MAIÚSCULAS;
- Estruturas e variáveis: sublinhado.

Novos conceitos e palavras-chave são destacados no texto em **negrito** e as palavras de origem estrangeira estão apresentadas em *itálico*.

Capítulo 2

Integração de Serviços

Está cada vez mais popular o uso de recursos multimídia. A demanda pela transmissão de dados, som e imagens é cada vez maior. A utilização destes recursos também se expande no contexto das redes de computadores. O uso de redes dedicadas e especializadas (como no caso das redes de dados, telex, telefonia e canais exclusivos de TV) para transmissão destes recursos já não atende à demanda. Procura-se então um novo conceito de Integração de Serviços, onde se possa utilizar uma mesma rede como suporte único para todas os tipos de transmissão [SBB96].

As redes mais tradicionais, como as baseadas em TCP/IP (*Transport Control Protocol / Internet Protocol*) [Com91], maximizam a utilização da rede através da multiplexação de múltiplos fluxos de dados e fazem comunicação multiponto, apesar de executar tipicamente somente um serviço de transmissão de datagramas em *best-effort*, que não oferece nenhum tipo de garantia de recursos.

As novas redes comutadas baseadas na tecnologia B-ISDN (*Broadband Integrated Service Digital Network*) [Mon94] garantem os serviços. Porém, estas redes ainda são ineficientes com tráfego em rajadas e o suporte à comunicação multiponto-a-multiponto só pode ser feito através de várias conexões ponto-a-multiponto.

Os sistemas de rede de tempo real tradicionais podem alocar recursos excessivos por duas razões [GHMN94]:

1. eles alocam recursos baseados na previsão do pior caso do tráfego atual;

2. eles tratam o tráfego em conexões diferentes independentemente ao determinar suas necessidades de recursos.

Uma nova tecnologia de redes chamada ISPN (*Integrated Services Packet Network*) [MESZ94] vem sendo discutida com o objetivo de juntar os dois paradigmas, combinando a comunicação multiponto multiplexada, a segurança das redes ISDN e a garantia dos serviços do modelo por comutação de circuitos.

Para discutir o assunto e sugerir soluções e padronizações para a integração de serviços num ambiente baseado em comutação de pacotes (tipicamente IP) e sua extensão às demais tecnologias, foi criado pelo IETF (*Internet Engineering Task Force*) [HC94] um grupo de trabalho chamado *Integrated Services Work Group* (Int-Serv). Como primeira tarefa deste grupo, foi feita a definição do modelo, a identificação dos elementos da arquitetura e a especificação das funcionalidades do modelo [BCS94].

O Int-Serv detectou que a infra-estrutura da Internet, alvo principal da norma, precisaria ser modificada para suportar Qualidade de Serviço (*QoS - Quality of Service*) de tempo real, que possibilite algum controle sobre os atrasos dos pacotes fim-a-fim, para a ampliação do uso de aplicações em tempo real.

Em [ZDESZ93] e [Tob94], são identificados os componentes necessários nas arquiteturas capazes de suportar Qualidade de Serviço. Generalizando os pré-requisitos definidos nestes documentos, temos os seguintes componentes:

- **Especificação do Fluxo**
  - ☞ O usuário comunica à rede a característica do fluxo de dados a ser gerado e a rede pode identificar a QoS necessária a este fluxo. Esta componente é conhecida como *flowspec*.
- **Controle de Tráfego**
  - ☞ Como os recursos da rede são finitos, esta não pode atender a todos os pedidos de reservas de recursos. A arquitetura da rede precisa conter um conjunto de regras para a aceitação das reservas, identificação dos fluxos de dados e controle de quando e como transmitir os pacotes.
- **Roteamento**
  - ☞ A rede precisa decidir como transmitir os pacotes da origem até o destino. O roteamento deve ser capaz de definir os caminhos para comunicação *unicast* e *multicast* [BZ93]. A extensão do modelo de roteamento para atender também ao tráfego *multicast* é necessária, pois a simples generalização a partir de conexões *unicast* (ponto-a-ponto) não funcionaria [BCS94].

- **Reserva de Recursos**
  - ☞ A rede deve ser capaz de reservar os recursos necessários ao longo do caminho a ser percorrido pelo fluxo de dados.

A Figura 2.1 mostra um exemplo de implementação do modelo de referência para Integração de Serviços em roteadores e *hosts* [BCS94].

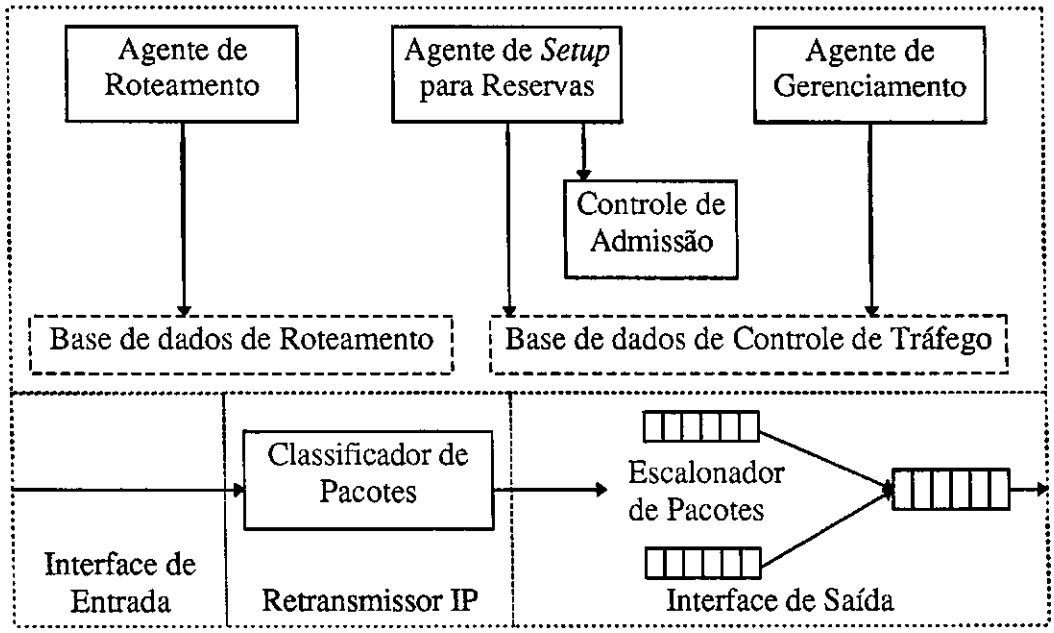


Figura 2.1 - Modelo de referência para Integração de Serviços em Roteadores

Além da Qualidade de Serviço, também é necessária a habilidade de controlar o compartilhamento da banda dos enlaces por diferentes classes de serviço. Estas classes podem, por exemplo, representar diferentes grupos ou famílias de protocolos distintas.

A extensão do modelo IP envolve a inclusão de uma série de mecanismos que viabilizem a Integração de Serviços, mantendo os serviços IP básicos. Para isso, deve haver um compromisso com dois elementos:

1. um modelo de serviço estendido, chamado modelo de Integração de Serviços e;
2. uma referência para implementação, que disponibilize um vocabulário e uma organização genérica para programas que possibilite a implementação do modelo de Integração de Serviços.

É importante separar o modelo de serviço, que define a parte visível do modelo, das discussões de implementação, que podem ser substituídas ao longo do tempo. Contudo, ambas estão relacionadas.

Cada classe de serviço definida possui características para especificação do fluxo próprias. Há, porém, um modelo mínimo a ser seguido para a troca de dados entre as classes de serviço. Este modelo é apresentado na Seção 2.5.

A Seção 2.1 apresenta o modelo de Integração de Serviços e suas peculiaridades. A Seção 2.2 dá uma breve introdução a respeito do mecanismo de Controle de Tráfego. A Seção 2.3 discute a utilização de protocolos de reserva de recursos. A Seção 2.4 trata a questão do roteamento de pacotes. A seção 2.5 apresenta as regras para especificação dos fluxos de dados. Finalmente, a Seção 2.6 apresenta as classes de serviço em padronização.

2.1 Modelo de Integração de Serviços

Um modelo de serviço consiste de um conjunto de comprometimentos do serviço, em resposta a uma solicitação de recursos de rede, necessários para a transmissão de um ou um conjunto de fluxos de dados. O comprometimento de serviços para fluxos de dados individuais visa prover uma melhor performance da aplicação, de acordo com a qualidade de serviço solicitada. O comprometimento de serviços coletivos são direcionados ao compartilhamento de recursos ou economia de solicitações. Assim, os mesmos recursos ficariam disponíveis a várias entidades ao mesmo tempo.

A seguir será discutida a terminologia usada no modelo de Integração de Serviços (Seção 2.1.1), como podem ser tipificadas as aplicações que utilizam este modelo (Seção 2.1.2), as formas de compartilhamento de recursos de rede (Seção 2.1.3) e como é montada a estrutura das especificações dos serviços (Seção 2.1.4).

2.1.1 Componentes do Modelo

No modelo de Integração de Serviços são usados diversos termos. Faz parte da especificação do modelo a definição destes termos e seus conceitos [SW95]. Os seguintes termos são usados pelo modelo:

- **Qualidade de Serviço (QoS)**
  - ☞ No contexto da Integração de Serviços, Qualidade de Serviço (QoS) se refere à natureza do serviço que a transmissão de pacotes deve receber. Isto inclui a descrição da banda a ser reservada, o atraso dos pacotes e taxas de perda de pacotes, dentre outros parâmetros possíveis. O conceito de QoS também é válido para diversas tecnologias e modelos.
- **Elemento de Rede**
  - ☞ Um elemento de rede (ou somente elemento), é qualquer componente de uma inter-rede (ou simplesmente rede) pelo qual passam pacotes de dados, sendo capaz de executar controle da qualidade de serviço sobre o fluxo de dados que passa por ele. Elementos de rede incluem roteadores, subredes e sistemas fim. Um elemento de rede capaz de suportar qualidade de serviço (*QoS-capable*) é aquele que oferece um ou mais serviços, de acordo com as regras de Integração de Serviços. Um Elemento de Rede *QoS-aware* é aquele que suporta a interface necessária para a definição dos serviços, não sendo obrigatório a este elemento de rede suportar o serviço, apenas saber como rejeitá-lo.
- **Fluxo**
  - ☞ Um fluxo é um conjunto de pacotes que atravessa os elementos de rede, os quais estão cobertos pela mesma solicitação de controle para uma qualidade de serviço. Em um determinado elemento de rede, um fluxo pode consistir de pacotes de uma simples aplicação ou a agregação de tráfegos de dados combinados de um grupo de aplicações.
- **Serviço**
  - ☞ O termo serviço descreve um denominado conjunto de capacidades de controle da qualidade de serviço, disponibilizado por um simples elemento de rede. A definição de serviço inclui a especificação de funções a serem executadas pelo elemento de rede, as informações necessárias ao elemento para a execução destas funções e a informação que será disponibilizada pelo

elemento de rede aos outros elementos de rede. Um serviço é implementado através de um módulo de serviço contido no elemento de rede.

- **Comportamento (*Behavior*)**
  - ☞ O comportamento é a performance relacionada com a qualidade de serviço fim-a-fim vista por uma aplicação. Isto é o resultado da composição de serviços oferecidos por cada elemento de rede ao longo do caminho do fluxo de dados da aplicação. Quando os elementos de rede oferecem o mesmo serviço ao longo do caminho, é freqüentemente possível explicar o comportamento fim-a-fim resultante. O comportamento de um fluxo de dados que percorre um caminho com elementos de rede usando diferentes serviços é mais difícil de ser avaliado, podendo ser indefinido.
- **Caracterização**
  - ☞ A caracterização é uma aproximação computada do comportamento fim-a-fim atual de um fluxo, solicitando uma qualidade de serviço específica para a rede.
- **Parâmetros de Caracterização**
  - ☞ A caracterização é computada a partir de um conjunto de parâmetros de caracterização dados por cada elemento de rede no caminho do fluxo de dados e uma função de composição que computa a caracterização fim-a-fim destes parâmetros. Em [SHE95] são definidos parâmetros de caracterização genéricos, que são independentes dos serviços.
- **Função de Composição**
  - ☞ A função de composição recebe parâmetros de caracterização como entrada e computa a caracterização.
- **Especificação de Tráfego - TSpec**
  - ☞ A especificação de tráfego, ou TSpec, é uma descrição do tráfego para o serviço que esta sendo solicitado. Em geral, o TSpec forma o "contrato" entre o fluxo de dados e o serviço. Uma vez sendo aceito o serviço solicitado, o módulo de serviço está obrigado a oferecer a QoS, enquanto o fluxo de dados estiver de acordo com o TSpec.

- **Especificação de Requisição de Serviço - RSpec**
  - ☞ A Especificação de Requisição de Serviço, ou RSpec, é uma especificação da qualidade de serviço que um fluxo deseja solicitar de um elemento de rede. O conteúdo de uma RSpec é específico para cada serviço. Como exemplo, esta especificação pode conter a informação a respeito de banda a ser alocada para o fluxo, o atraso máximo ou a taxa de perda de pacotes.
- **Protocolo de *Setup***
  - ☞ Um protocolo de *setup* é usado para levar as informações relacionadas com a qualidade de serviço dos sistemas-fim solicitando controle de QoS aos elementos de rede que precisam executar este controle e, ainda, instalar e manter o estado de controle da qualidade de serviço nestes elementos de rede. Ele também pode ser usado para coletar informações a respeito da QoS no interior dos elementos de rede ao longo do caminho do fluxo de dados. Como exemplo deste mecanismo, temos os protocolos de reserva de recursos, como o RSVP e o ST-II, que serão citados na Seção 2.3.
- ***Token Bucket***
  - ☞ Um *token bucket* [Par92b] é uma forma particular de especificação de tráfego consistindo de uma *token rate* **r** e um *bucket size* (ou *bucket depht*) **b**. Essencialmente, o parâmetro **r** especifica a taxa de dados contínua sustentada, enquanto o parâmetro **b** especifica o pico que a taxa de transmissão pode atingir por curtos períodos de tempo. Mais precisamente, durante um período de tempo (T) o tráfego total não deve ultrapassar **rT+b** bits.
- ***Token Bucket Filter***
  - ☞ O *Token Bucket Filter* é um filtro ou uma função de policiamento que diferencia os pacotes que estão de acordo com a especificação de tráfego dos demais pacotes que compartilham o mesmo enlace.
- **Controle de Admissão**
  - ☞ O Controle de Admissão é o processo de decisão sobre as novas solicitações de serviço em um elemento de rede. Esta ação precisa ser executada para qualquer serviço que pretende oferecer limites quantitativos de performance. O critério de decisão do Controle de Admissão é específico de cada serviço.

- **Policiamento**
  - ☞ O policiamento é um conjunto de ações a serem executadas, quando as características momentâneas de um fluxo de dados excedem os valores definidos na especificação do tráfego. Os serviços que requerem policiamento precisam especificar as ações a serem executadas quando ocorrerem as discrepâncias e definir os locais onde serão feitas esta verificação. Como exemplo de ações, podemos relegar os pacotes do tráfego *best-effort*, descartar pacotes, realinhar o tráfego ou outras ações.
- **Interfaces**
  - ☞ Conceituam-se interfaces como a forma com que o módulo de serviço interage com as outras partes do elemento de rede. A especificação de serviço deve definir claramente os dados e os formatos que percorrerão cada interface e ter certeza de que o mapeamento entre as interfaces e os mecanismos do serviço sendo definidos estão claros.

2.1.2 Caracterização das Aplicações

O núcleo do modelo de serviços está no comprometimento com o tempo de transmissão/retransmissão dos pacotes. Assim, o atraso de pacotes é o ponto central do comprometimento com a Qualidade de Serviço. Levando-se em conta o atraso, pode-se definir dois tipos de aplicações:

- **Aplicações de Tempo Real**
  - ☞ Aplicações que necessitam que os dados cheguem no destino em um certo tempo. Estas aplicações são sensíveis ao atraso e às variações do mesmo. Se este tempo for extrapolado, os dados podem não fazer mais sentido para a aplicação.
- **Aplicações Elásticas**
  - ☞ Aplicações que sempre esperam os pacotes de dados, independentemente do tempo de chegada. Estas aplicações geram tráfego *best-effort.*

Para a transmissão das aplicações de tempo real pode-se utilizar o conceito de *playback*, onde em um determinado ponto do caminho do fluxo de dados é feito o

ressincronismo dos pacotes através de *buffers*, para que a variação do atraso (*jitter*) não seja percebido pela aplicação. Quando este é incluído na própria aplicação, as aplicações são chamadas de **Aplicações Adaptativas**. Estas aplicações tem a vantagem de se ajustar às condições em que a rede se encontra, variando sua taxa de transmissão e compensando o atraso.

O fato das aplicações elásticas esperarem sempre pela chegada dos pacotes não significa que elas sejam insensíveis ao atraso. Muito pelo contrário, o aumento do atraso dos pacotes irá reduzir a performance da aplicação. Estas aplicações têm por procedimento básico, utilizar os pacotes no mesmo instante que chegam, sem "buferizá-los" ou prosseguir o processamento sem eles.

As aplicações elásticas podem ser divididas em três tipos de aplicações, de acordo com a sensibilidade do atraso (ou elasticidade):

- **Rajadas Interativas** - *Interactive Burst*
  - ☞ Alta sensibilidade ao atraso (Telnet, X, NFS)
- **Transferência de Volumes de Dados Interativamente** - *Interactive Bulk Transfer*
  - ☞ Sensibilidade média ao atraso (FTP)
- **Transferência de Volumes de Dados Assincronamente** - *Asynchronous Bulk Transfer*
  - ☞ Baixa sensibilidade ao atraso (E. Mail, FAX)

A caracterização dos tipos de aplicações não é exata e nem está completa. Ela é citada apenas como guia para o desenvolvimento do núcleo do modelo de serviços. O julgamento deste modelo não deve ser feito por esta caracterização e sim pela adequação do modelo às necessidades de todo o espectro de aplicações. Por isso, o modelo deve ser robusto o suficiente para incorporar diversos conceitos.

2.1.3 Compartilhamento de Recursos

O compartilhamento de recursos surge da necessidade de compartilhamento de um mesmo enlace por diversas entidades ao mesmo tempo. Este compartilhamento pode ser:

- **Compartilhamento Multi-entidades**

- ☞ Quando mais de uma organização compartilha o mesmo enlace. Deve-se estabelecer um tratamento eqüitativo a todas as entidades e manter a carga da rede sob controle, garantindo o investimento em comum.

- **Compartilhamento Multi-protocolos**
  - ☞ Quando mais de uma família de protocolos compartilham o mesmo enlace. Deve-se prevenir que um protocolo sobrecarregue a rede, em detrimento dos demais.

- **Compartilhamento Multi-serviços**
  - ☞ Em uma família de protocolos, como o IP, um administrador pode desejar reservar uma fração da banda alocada para várias classes de serviço. Por exemplo, reservar banda para o tráfego de tempo real, evitando que o enlace seja sobrecarregado com o tráfego *best-effort*.

A forma de distribuição dos recursos para a efetivação do compartilhamento pode ser feita de diversas formas. Uma forma de gerenciar esta distribuição é a utilização de um modelo hierárquico, onde os recursos são distribuídos em forma de árvore [Par92a]. A forma com que é feita a distribuição e a aceitação de solicitações de recursos é definida pelo Controle de Tráfego (Seção 2.2).

2.1.4 Composição das Especificações de Serviços

As especificações de serviços devem possuir um corpo de apresentação comum. Cada seção é definida e pode ter o caráter **obrigatório** ou **opcional** e ainda, **informativo** ou **normativo**. São definidas para o corpo do documento as seguintes seções:

- **Comportamento fim-a-fim** (obrigatório e informativo)
  - ☞ É a descrição do comportamento resultante, se todos os elementos de rede ao longo do caminho oferecem o mesmo serviço.

- **Motivação** (obrigatório e informativo)
  - ☞ Discute porque o serviço está sendo definido. Descreve que tipos de aplicações podem usar o serviço e porque o serviço pode ser mais apropriado para estas aplicações do que os demais serviços.

- **Propriedades Obtidas dos Elementos de Rede** (obrigatório e normativo)
  - ☞ Descreve as propriedades da qualidade de serviço obtida pelos pacotes de dados processados usando o serviço. A descrição precisa explicar que variáveis são controladas, o grau de controle executado e os aspectos fixados ou assumidos para a obtenção do serviço.
- **Informação de Invocação** (obrigatório e normativo)
  - ☞ Descreve o conjunto de parâmetros para se invocar o serviço e a descrição de como os valores dos parâmetros serão usados pelo módulo do serviço.
- **Informações Exportadas e Parâmetros de Caracterização** (obrigatório e normativo)
  - ☞ Descreve as informações que devem ser coletadas e exportadas pelo módulo do serviço. As informações exportadas estão disponíveis aos demais módulos do elemento de rede e, consequentemente, aos protocolos de controle, protocolos de roteamento, ferramentas de gerenciamento e assemelhados.
- **Policiamento** (obrigatório e normativo)
  - ☞ Descreve a natureza do policiamento usado para manter a conformidade do tráfego. A especificação deve descrever as ações de policiamento esperadas, a legalidade para alternativas de ações de policiamento, a localização das ações de policiamento na rede e as considerações técnicas adicionais aplicáveis.
- **Ordenação e Agrupamento** (obrigatório e normativo)
  - ☞ Descreve as ações a serem tomadas, quando ocorre mais de uma solicitação de serviço para um mesmo fluxo de dados.
- **Guia para Implementação** (opcional e informativo)
  - ☞ Esta seção deve definir um guia com as expectativas do autor da especificação do serviço, com relação às implementações possíveis.
- **Critérios de Avaliação** (obrigatório e informativo)
  - ☞ Define o comportamento esperado da implementação do serviço. São descritos os testes que podem ser usados para avaliar a implementação do serviço.
- **Exemplos de Implementação** (opcional e informativo)
  - ☞ Descreve exemplos referenciados na literatura, que podem ser usados para as implementações.

- **Exemplos de Uso** (opcional e informativo)
  - ☞ Descreve alguns exemplos que podem ser tomados para o uso do serviço.

2.2 Controle de Tráfego

No modelo de roteamento utilizado atualmente, o conjunto de ações para seleção de rotas é muito limitado. A partir da chegada de um pacote, o roteador deve selecionar a rota e retransmitir ou descartar o pacote. É necessário portanto, a extensão do modelo utilizado pelos roteadores para comportar os novos componentes para a Integração de Serviços.

Para a efetivação da implementação do modelo de Integração de Serviços, é definido um mecanismo que possa tratar as diferentes classes de serviço e efetuar as funções necessárias para a aceitação e transmissão dos fluxos de dados, de acordo com cada uma destas classes. Este mecanismo é conhecido como **Controle de Tráfego**. O mecanismo de Controle de Tráfego possui os seguintes componentes:

- **Controle de Admissão**;
- **Classificador de Pacotes** e;
- **Escalonador de Pacotes.**

O Controle de Admissão é o responsável pela aceitação ou não de uma solicitação de recursos para transmissão de um fluxo de dados. Para esta aceitação são definidas regras fixas e outras que podem variar, conforme a classe de serviço solicitada.

O Classificador de Pacotes é responsável pela identificação dos pacotes pertencentes a um determinado fluxo de dados, quando há compartilhamento de recursos. Esta identificação é necessária para que cada fluxo de dados receba o tratamento que solicitou no momento da reserva de recursos.

O mecanismo de Escalonamento de Pacotes tem por finalidade controlar a saída dos pacotes para o enlace, retardando a transmissão e/ou reordenando as filas de saída quando preciso. Este controle se faz necessário para que os fluxos recebam o tratamento solicitado e para que um determinado fluxo de dados não venha utilizar os recursos reservados aos outros fluxos.

Neste documento é proposto um mecanismo de Controle de Tráfego utilizando como base a Integração de Serviços. Neste mecanismo é proposto um tipo de serviço simples, com controle de banda e atraso. Para este controle é proposto um protocolo de gerenciamento dos recursos para redes compartilhadas (no caso, redes ethernet), denominado DCRP (*Distributed Control Reservation Protocol*). O DCRP funcionará em conjunto com um mecanismo de Controle de Tráfego e o protocolo de reserva de recursos (mais precisamente, o RSVP).

As informações coletadas através do protocolo de gerenciamento de recursos são repassadas ao Controle de Tráfego para ser usado pelo Controle de Admissão e no Escalonamento de Pacotes. O protocolo de reserva de recursos irá contactar o Controle de Tráfego para a execução das reservas e transmissão de parâmetros adicionais, como um filtro de pacotes. A Figura 2.2 ilustra o funcionamento do modelo.

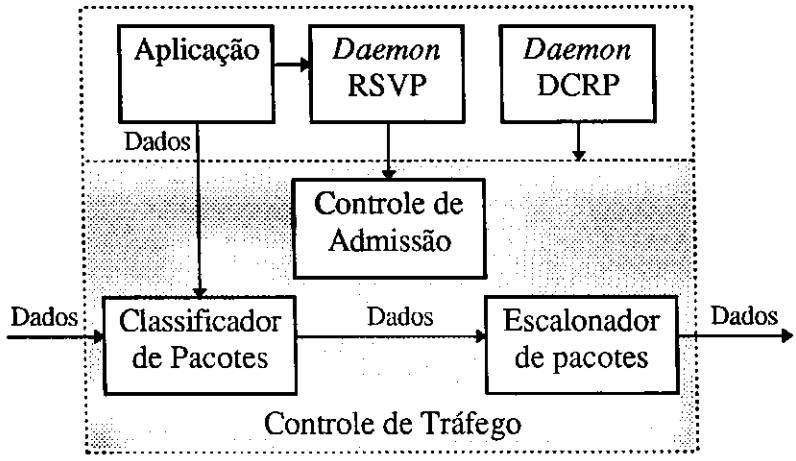


Figura 2.2 - Proposta de Controle de Tráfego

O DCRP e o mecanismo de Controle de Admissão são definidos no Capítulo 4, o mecanismo de Classificação de Pacotes é descrito no Capítulo 5 e o mecanismo de Escalonamento de Pacotes está no Capítulo 6.

2.3 Reserva de Recursos

Além do Controle de Tráfego, detectou-se a necessidade da definição de protocolos de reserva de recursos, que possam efetuar as solicitações de recursos ao longo do caminho a ser

utilizado pelos fluxos de dados das aplicações de tempo real. Atualmente os grupos de trabalho na Internet estudam dois protocolos de reserva de recursos:

- **ST-II** - *Experimental Internet STream Protocol* e;
- **RSVP** - *Resource Reservation Protocol.*

O ST-II [DB95] é derivado do ST (*STream Protocol*) [For79]. Por ser mais antigo, possui além de implementações acadêmicas, algumas implementações comerciais já a certo tempo. Seu modelo de funcionamento é orientado a conexão e as reservas são definidas pelo transmissor. Sua implementação, porém, requer um modelo próprio de comunicação, que difere do modelo IP.

O RSVP [ZDESZ93] é mais recente, ainda está em fase de desenvolvimento e possui implementações acadêmicas e, em breve, implementações comerciais (Capítulo 3). Sua especificação é feita para funcionar com o modelo de Integração de Serviços e incorpora conceitos atuais. Por isso, foi escolhido para funcionar junto com o modelo de Integração de Serviços.

Esta escolha não impede a especificação e uso do ST-II como protocolo de *setup* no modelo de Integração e Serviços. O Grupo que trabalha com o ST-II também vem realizando pesquisas neste sentido.

Numa comparação entre os dois protocolos [MESZ94], o RSVP apresentou menor *overhead*, melhor interação entre transmissor e receptor e um maior leque de opções para o tratamento de reservas, como estilos diferentes de reservas e seleção de transmissores.

Além destes dois protocolos, há outras experiências, como o **RDP** (*Real Time Datagram Protocol*) [MAN94] e o **NRR** (*Network Resource Reservation Program*) [IBM94]. O primeiro faz parte de uma proposição de arquitetura para comunicação em tempo real, adaptando protocolos já consagrados como o UDP e o IP. O segundo é um produto comercial IBM, que faz reserva de recursos para a transmissão de dados multimídia, utilizando o paradigma cliente-servidor, sobre o protocolo IP, em redes locais.

Quando for citado o termo Protocolo de Reserva de Recursos neste documento, isto se refere RSVP. Mais detalhes sobre o RSVP serão apresentados no Capítulo 3.

2.4 Roteamento

Um dos pontos mais sensíveis do modelo de Integração de Serviços no IP é o roteamento a ser utilizado para a descoberta e manutenção dos caminhos entre a(s) origem(ns) e o(s) destino(s). Discutiremos brevemente alguns pontos importantes a respeito do suporte que o roteamento deve prover, para que possa ser implementada a Integração de Serviços em IP (Internet, Intranet, redes corporativas privadas, etc.).

A primeira questão é o tipo de comunicação desejado para as aplicações que farão uso da Integração de Serviços (Figura 2.3). Numa comunicação pessoal, por exemplo, é usado normalmente a comunicação ponto-a-ponto. Nas distribuições de vídeo, por exemplo, é usada a comunicação ponto-a-multiponto. Nas vídeo-conferências, por exemplo, é usada a comunicação multiponto-a-multiponto.

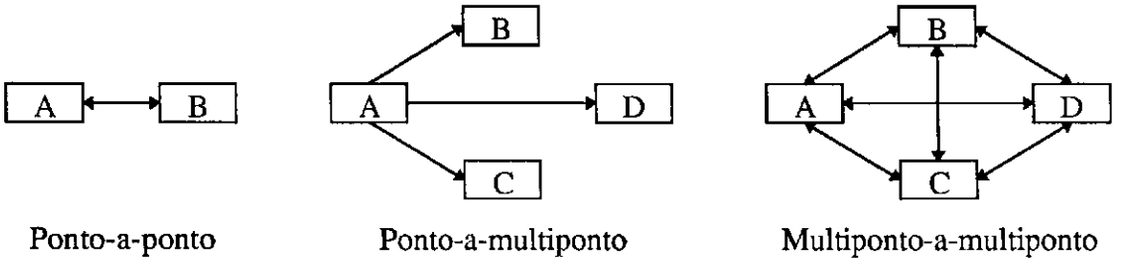


Figura 2.3 - Tipos de comunicação

A utilização de roteamento *unicast* para atender aos três tipos de comunicação é bastante complicada e dispendiosa. Nas comunicações ponto-a-ponto, não há problemas. Mas, nas demais há a necessidade de se abrir e manter múltiplas conexões ponto-a-ponto. Tomamos como exemplo a Figura 2.4, para transmissão de vídeo do *host* H1 para os *hosts* H2, H3, H4 e H5, distribuídos em três subredes e interligados pelos roteadores R1, R2 e R3. Para esta transmissão seria necessário abrir e manter pelo menos quatro conexões ponto-a-ponto (N-1). Se os demais *hosts* do grupo também desejarem transmitir dados, o número de conexões passa para 10 (combinação de N, 2 a 2).

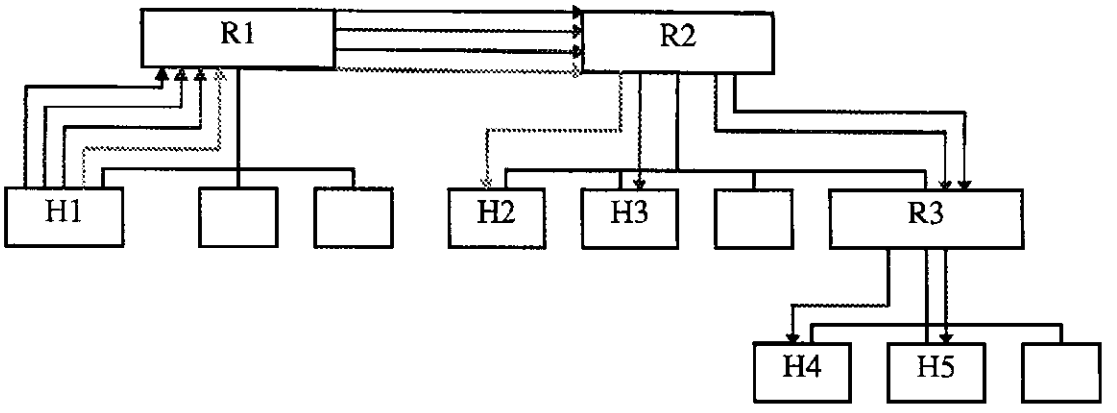


Figura 2.4 - Transmissão ponto-a-multiponto com roteamento *unicast*

Tomamos como exemplo a Figura 2.5, para transmissão de vídeo do *host* H1 para os *hosts* H2, H3, H4 e H5, distribuídos em três subredes, interligadas pelos roteadores R1, R2 e R3. Para esta transmissão seria necessário abrir e manter apenas uma conexão *multicast*. Além disso, poderia ser utilizada a mesma conexão para comunicação multiponto-a-multiponto, onde todos podem transmitir. Para evitar a abertura de múltiplas conexões ponto-a-ponto, o roteamento *multicast* [DC85] é a melhor solução.

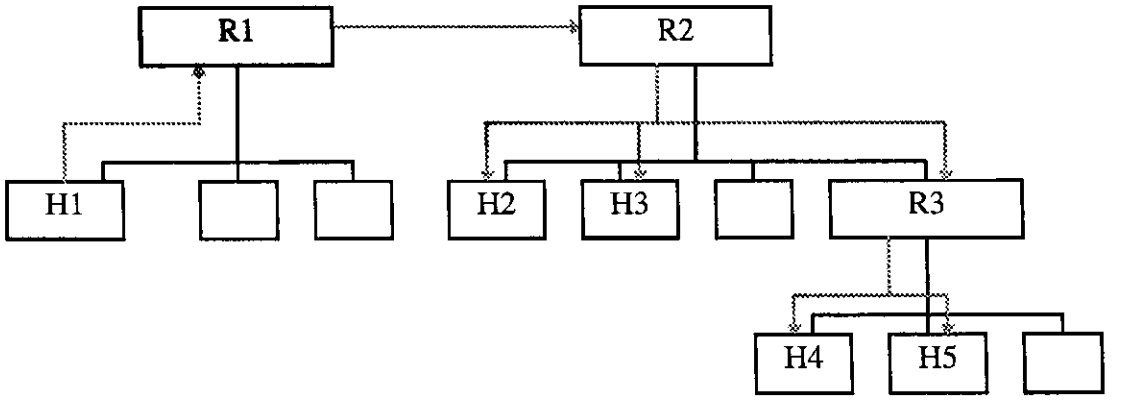


Figura 2.5 - Transmissão ponto-a-multiponto com roteamento *multicast*

Os roteadores atuais funcionam muito bem no roteamento *unicast*. Mas, poucos roteadores têm capacidade *multicast* ou estão corretamente configurados para isso. Na atual estrutura da Internet, o roteamento *multicast* é feito utilizando *hosts* ou roteadores que se comunicam entre si através de túneis [Dee89] e um protocolo de roteamento conhecido como DVMRP (*Distance Vector Multicast Routing Protocol*) [WPD88]. Os *hosts* ou roteadores interligados ficam responsáveis pela distribuição e replicação dos pacotes entre si e nas subredes às quais estão conectados.

Esta estrutura de túneis é conhecida como MBONE (*Multicast Backbone*) [Kum95]. O MBONE é uma estrutura temporária e foi usada inicialmente para testes de aplicações multimídia e de protocolos. As primeiras experiências com transmissão de conferências foram realizadas pelo IETF em 1992 [CD92]. Atualmente o MBONE vem sendo utilizado para as mais diversas finalidades, que vão de conferências pessoais, até transmissão de *shows* de Rock [Kum96].

Outros mecanismos de suporte ao roteamento *multicast* vêm sendo desenvolvidos. Entre eles, se destacam o protocolo MOSPF (*Multicast Open Shortest Path First*) [Moy94a] e o PIM (*Protocol Independent Multicasting*) [DEFJLW95]. O MOSPF é uma extensão do protocolo de roteamento OSPF (*Open Shortest Path First*) para comportar roteamento *multicast* [Moy94b]. O PIM está em fase de especificação, e não é baseado em qualquer outro protocolo de roteamento *unicast*.

A estrutura atual das implementações que fazem uso de integração de serviços está baseada no MBONE. Mas, os outros mecanismos vêm ganhando adeptos, à medida que as especificações avançam. O PIM aparece neste contexto como um forte candidato como futuro padrão para as comunicações *multicast*.

2.5 Especificação do Fluxo

A reserva de recursos no contexto de Integração de Serviços exige que uma série de conjuntos de informações, definidos como objetos, sejam trocados entre os nós ao longo do caminho a ser percorrido pelos fluxos de dados. Estes objetos definem as características destes fluxos, de acordo com a qualidade de serviço necessária para a execução da comunicação.

Para que a troca de objetos seja feita de forma ordenada, são estipuladas regras e formatos que devem ser respeitados por todas as classes de serviço [Wro95]. Os objetos comuns entre as classes de serviço, para a execução de reserva de recursos, são os seguintes:

- TSPEC - Especificação de Tráfego
- RSpec - Especificação de Requisição
- Parâmetros de Caracterização
- Variáveis de Composição

Em [Wro95] é descrito o formato das mensagens a serem transmitidas pelos protocolos que desejam enviar objetos no modelo de Integração de Serviços. O dado contido em um objeto específico é único para o serviço que define o objeto e é descrito na definição do serviço.

A mensagem consiste de um cabeçalho de 32 bits (Figura 2.6), especificando o número da versão (VER) e o tamanho total da mensagem (*OVERALL LENGTH*). Seguindo o cabeçalho estão os dados específicos do serviço.

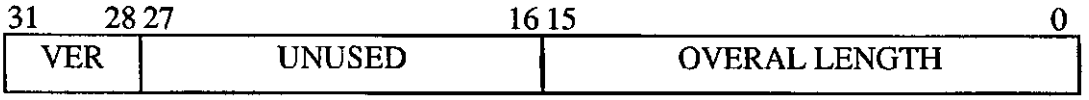


Figura 2.6 - Formato do cabeçalho das mensagens para Integração de Serviços

Cada bloco de dados específico do serviço começa com um cabeçalho de 16 *bitís*, contendo o número (SVC_NUMBER) do serviço e o comprimento do bloco (LENGTH), seguido pelos parâmetros específicos do serviço (Figura 2.7).

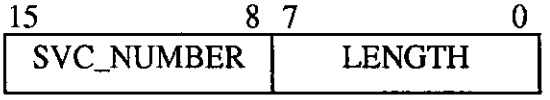


Figura 2.7 - Cabeçalho do campo de serviço

Cada parâmetro no bloco de dados do serviço é composto por um cabeçalho identificador de 16 *bits* (Figura 2.8), contendo o número de identificação do parâmetro (PARAM_NUM) e um campo de *flag* (PARAM_FLAGS).

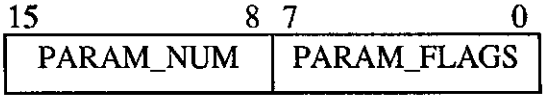


Figura 2.8 - Campo específico do parâmetro

2.6 Especificações de Serviços

Até o momento foram sugeridos pelo grupo de trabalho em Integração de Serviços (Int-Serv) cinco tipos de serviços. Todos estes serviços ainda estão documentados na forma de "internet-drafts" e alguns deles podem em breve ser transformados em padrões.

Dos serviços especificados pelo Int-Serv, dois encontram-se com seus documentos de especificação vencidos. Apesar disso, eles serão apresentados como especificações do Int-Serv. Mesmo vencidos, seus números de identificação continuam válidos e não devem ser substituídos por outros. As especificações do Int-Serv e suas condições atuais são as seguintes:

1. ***Controlled Delay Quality of Service*** - Vencido;
2. ***Guaranteed Quality of Service*** - Válido;
3. ***Predictive Quality of Service*** - Vencido;
4. ***Controlled-Load Network Element Service*** - Válido;
5. ***Committed Rate Quality of Service*** - Válido;

Quando um elemento de rede não implementa determinado serviço, este elemento pode rejeitar a solicitação do serviço não implementado ou possuir alguma forma de adaptar o serviço solicitado para algum outro serviço implementado [BS95]. O mecanismo de Controle de Tráfego proposto neste documento executa adaptações entre os serviços requisitados pelo RSVP e o serviço oferecido.

Existem outros três documentos que sugerem ações para o uso de Serviços Integrados em elementos de rede específicos. O primeiro discute a adaptação de *switches* para o funcionamento em ambiente de Integração de Serviços [Kra96]. O segundo documento [Kim96] descreve um protocolo a ser usado pelos dispositivos de rede baseados no padrão IEEE 802.1 [Tan94] .O terceiro documento se refere ao uso do modelo de Integração de Serviços sobre subredes ATM (*Asynchronous Transfer Mode*) [BG96]. Estes documentos sugerem novas formas de mapeamento de serviços, compatibilização e sinalização de protocolos. As discussões contidas nestes dois documentos são bastante complexas e ainda muito polêmicas, fugindo do escopo deste trabalho. Portanto, estes documentos não serão discutidos neste trabalho

A seguir serão apresentados sucintamente os serviços em definição pelo Int-Serv, na mesma ordem em que foram citados anteriormente nesta seção.

2.6.1 *Controlled Delay Quality of Service*

O serviço *Controlled Delay* [SPW95], ou atraso controlado, foi o primeiro serviço a ser especificado e está atualmente em desuso. Ele seria voltado para aqueles elementos de rede

que estabelecem formas de controle de atraso. O serviço se compromete a controlar os níveis de atraso e perda de pacotes. A banda alocada para o fluxo de dados deve ser na média maior ou igual à banda solicitada pelo serviço.

O serviço pode ser utilizado pelas aplicações que se adaptam aos serviços e ao atraso. Estas aplicações são sensíveis ao atraso, mas estão preparadas para se adaptar às alterações dinâmicas das condições da rede. Com isso, elas podem estar preparadas para substituir os níveis de serviço solicitados, se o nível de serviço oferecido pelo elemento de rede não for adequado.

São especificados três níveis lógicos de serviço diferentes. O elemento de rede pode implementar todos ou apenas alguns dos níveis lógicos, mas precisa mapeá-los na interface do serviço. Eles possuem graus diferentes de controle do atraso, conforme o nível:

1. Maior controle de atraso.
2. Controle médio de atraso.
3. Menor controle de atraso.

O elemento de rede não precisa definir para cada nível atrasos diferentes. Ou seja, os fluxos que solicitam o nível 1 não precisam receber necessariamente um atraso menor do que os que solicitaram o nível 3. Precisa-se apenas garantir que o fluxo com nível 1 de controle não obterá atraso maior que o do nível 3.

O tráfego *best-effort* também pode ser oferecido pelos elementos de rede que implementam este serviço. Porém, quando a carga da rede for alta, o atraso obtido no serviço de Atraso Controlado deve ser significativamente menor que o atraso experimentado pelo tráfego *best-effort*.

A taxa de perda de pacotes oferecida pelo serviço depende da implementação e deve variar entre 10^{-4} e 10^{-8}.

A definição do serviço não requer qualquer controle da variação de atraso na transmissão de pacotes no fluxo (*jitter*). Mas, este controle pode ser implementado, se assim for desejado.

Não é permitida a fragmentação de pacotes. Pacotes maiores que a MTU (*Maximum Transmission Unit*) [Com91] do enlace devem ser tratados pelo mecanismo de policiamento. Os pacotes que estiverem fora das especificações de tamanho ou que extrapolarem os limites de carga definidos no fluxo serão enviados junto com o tráfego *best-effort*.

A definição do fluxo para o elemento de rede é feita na forma de TSpec e a solicitação de serviço na forma de RSpec. O TSpec é formado pelo ***token bucket*** (Seção 2.1.1) mais um tamanho mínimo de pacote **m** e o tamanho máximo dos pacotes **M**.

O *token bucket* é formado pelo *bucket depht* **b** e pelo *bucket rate* **r**. Tanto **b** quanto **r** precisam ser positivos e são representados na forma de ponto flutuante. A taxa **r** é medida em *bytes* de datagramas IP por segundo e pode variar de 1 *byte* a 40 *Tbytes* por segundo. O parâmetro **b** também é medido em *bytes* e pode variar entre 1 e 250 *Gbytes*.

O tamanho mínimo de pacote, **m**, é um inteiro medido em *bytes*. Todos os datagramas IP menores que este valor serão armazenados no *buffer* até que se possa transmitir **m** *bytes*. O tamanho máximo de pacote, **M**, também é medido em *bytes*, sendo o maior tamanho aceito pelo Controle de Tráfego. O fluxo deve ser rejeitado se o valor de **M** for maior que a MTU do enlace. Tanto **m** quanto **M** devem ser positivos e **m** precisa ser menor ou igual a **M**.

O RSpec é o nível de serviço solicitado. O serviço é especificado pelos valores inteiros 1, 2 ou 3. As implementações devem substituir internamente estes valores pelos atrasos correspondentes. A representação pode chegar a até 256 níveis, para extensões futuras. Não é sugerido na especificação valores de atraso para estes níveis, como é feito no Serviço Predito (Seção 2.6.3).

O TSpec pode ser representado por dois números de ponto flutuante com precisão simples no formato IEEE, seguido por dois inteiros de 32 bits em *network byte order*. O primeiro valor é a taxa **r**, o segundo é o *bucket size* **b**, o terceiro o valor de **m** e o quarto o valor de **M**. O RSpec pode ser representado por um inteiro de 16 bits, sem sinal, em *network byte order*.

Cada nível de serviço possui três subníveis de atraso, totalizando nove parâmetros de caracterização. Cada um destes parâmetros é baseado no atraso máximo experimentado na rede por intervalos de tempo (**T**) pré-determinados. Estes intervalos são definidos como 1 segundo, 60 segundos e 3600 segundos.

Os atrasos computados nos elementos de rede e que serão exportados podem variar entre 1 e 2^{28} microssegundos em cada elemento e atualizando até (2^{32})-1 microssegundos em todo o caminho.

O serviço recebe a identificação de serviço número **1**, os parâmetros recebem a seguinte identificação:

1. TSpec.
2. RSpec.

Os parâmetros de caracterização de atraso recebem numeração de 1 a 9, conforme a tabela a seguir.

Identificação do Parâmetro	Nível do Serviço	T
1	1	1
2	1	60
3	1	3600
4	2	1
5	2	60
6	2	3600
7	3	1
8	3	60
9	3	3600

Tabela 2.1 - Parâmetros de caracterização do serviço *Controlled Delay*

2.6.2 *Guaranteed Quality of Service*

A especificação descreve as características necessárias ao elemento de rede para oferecer o *Guaranteed Service* [SPG96], ou Serviço Garantido (atraso garantido e banda). Isto envolve limites rígidos de atraso, alocação de banda, nenhum tipo de perda de pacotes e rotas fixas durante o tempo de vida do fluxo. Algumas aplicações de tempo real, como vídeo e audio do tipo *play-back,* necessitam desta rigidez e é para estas aplicações que é voltado o serviço.

O serviço não controla o atraso mínimo, médio ou a variação dos atraso (*jitter*) dos pacotes, apenas o atraso máximo. Para computar este atraso, é necessário determinar a latência do caminho do fluxo. Um limite conservativo pode ser calculado observando-se o atraso experimentado pelos pacotes.

O nível de serviço é caracterizado em cada elemento de rede pela banda (ou taxa de serviço), **R**, e por um tamanho de *buffer*, **B**. O **R** representa o montante do enlace que pode ser usado pelo Serviço Garantido e o **B** representa o espaço de *buffer* que o fluxo pode consumir no elemento de rede.

O atraso experimentado em um elemento de rede é caracterizado pelos parâmetros **C** e **D**. Estes parâmetros são denominados **termos de erro**. O termo de erro C é dependente da taxa de transmissão, representando o atraso que o datagrama pode experimentar em função da taxa de transmissão do fluxo. O termo de erro D é independente da taxa de transmissão e

Regras específicas de policiamento, composição de fluxos e compartilhamento de banda também são definidos na especificação. Estas regras são bastante complexas e não serão tratadas aqui, devendo o leitor seguir a referência da especificação para maiores detalhes [SPG96].

2.6.3 *Predictive Quality of Service*

A especificação descreve as características necessárias para que o elemento de rede ofereça o *Predictive Service*, ou Serviço Predito [SPDB95]. Este serviço estabelece um limite de atraso para a transmissão dos pacotes ao longo do caminho do fluxo de dados. Este limite é dado para a maioria dos pacotes, diferentemente do Serviço Garantido (Seção 2.6.2), que exige um limite rígido e absoluto para todos os pacotes. Este serviço é voltado para as aplicações que precisam de um limite para os atrasos fim-a-fim, mas que podem tolerar violações ocasionais deste limite.

Cada nível de serviço é associado a um limite de atraso e quase todos os pacotes são transmitidos com este limite. São definidos três níveis de atraso (como no Serviço de Atraso Controlado) e o atraso experimentado pelo serviço não poderá ser pior do que o experimentado pelo tráfego *best-effort*, quando a carga na rede é alta. A perda de pacotes deve ser rara quando o tráfego enviado está de acordo com os recursos solicitados.

O limite de atraso não é absolutamente rígido. Alguns pacotes podem chegar após o limite ter extrapolado ou com perda de pacotes na transmissão. Não é necessário o controle da variação de atraso dos pacotes (*jitter*) para estabelecer o limite. Espera-se que a maioria dos pacotes experimente um atraso menor do que o limite. Conseqüentemente, o atraso médio também deverá ser inferior a este limite.

Este serviço é voltado para as aplicações que desejam uma taxa de reserva com pequena perda de pacotes, um limite máximo de atraso e que sejam tolerantes ao descarte ou atraso nos pacotes. O limite de violação do atraso e de perda de pacotes pode variar conforme a implementação. Sugere-se limites entre 10^{-4} a 10^{-8}.

Associado ao serviço existem parâmetros de caracterização que descrevem o limite de atraso e o atraso atual experimentado nos três níveis de serviço. A diferença entre este serviço e o Serviço de Atraso Controlado está no limite de atraso razoavelmente confiável estabelecido pelo primeiro serviço, enquanto no segundo serviço não há qualquer segurança na

quantificação do serviço. Sem a caracterização, a aplicação passa a não fazer diferenciação entre os dois serviços.

São especificados, por enquanto, três níveis lógicos de serviço diferentes. O elemento de rede pode implementar todos ou apenas alguns dos níveis lógicos, mas precisa mapeá-los na interface do serviço. Eles possuem graus diferentes de controle do atraso, conforme o nível:

1. Maior controle de atraso.
2. Controle médio de atraso.
3. Menor controle de atraso.

Não é permitida a fragmentação de pacotes. Pacotes maiores que a MTU do enlace devem ser tratados pelo mecanismo de policiamento. Os pacotes que estiverem fora das especificações de tamanho ou que extrapolarem os limites de carga definidos no fluxo serão tratados como *best-effort*.

A definição do fluxo para o elemento de rede é feita na forma de TSpec e a solicitação de serviço na forma de RSpec. O TSpec é formado pelo ***token bucket***, mais um tamanho mínimo de pacote **m** e o tamanho máximo dos pacotes **M**. As regras de formatação destes parâmetros são as mesmas aplicadas no serviço de Atraso Controlado (Seção 2.6.1).

O RSpec é o nível de serviço solicitado. O serviço é especificado pelos valores inteiros 1, 2 ou 3. As implementações devem substituir internamente estes valores pelos atrasos correspondentes. A representação pode chegar a até 256 níveis, para extensões futuras. É sugerido para estes níveis os valores de 1 milissegundo para o nível 1, 10 milissegundos para o nível 2 e 100 milissegundos para o nível 3.

Cada nível de serviço possui três subníveis de atraso e um parâmetro de limite de atraso, totalizando doze parâmetros de caracterização. Cada um destes parâmetros é baseado no atraso máximo experimentado na rede por intervalos de tempo **T**, pré-determinados. Estes intervalos são definidos como 1 segundo, 60 segundos e 3600 segundos. Os parâmetros exportados são médias sobre o conjunto destes intervalos de tempo.

Não é necessário que estes parâmetros de caracterização sejam baseados em medidas exatas. Estas medidas podem ser baseadas em estimativas do atraso dos pacotes ou valores agregados da carga nas filas. Os atrasos computados nos elementos de rede e que serão exportados podem variar entre 1 e 2^{28} microssegundos em cada elemento e totalizando até (2^{32})-1 microssegundos em todo o caminho.

O Serviço Predito recebe o número de identificação 3.

Os parâmetros de caracterização de atraso recebem numeração de 1 a 12, conforme a tabela a seguir.

Identificação do Parâmetro	Nível do Serviço	Intervalo
1	1	Limite
2	1	T=1
3	1	T=60
4	1	T=3600
5	2	Limite
6	2	T=1
7	2	T=60
8	2	T=3600
9	3	Limite
10	3	T=1
11	3	T=60
12	3	T=3600

Tabela 2.2 - Parâmetros de caracterização do serviço *Predictive Service*

Regras específicas de policiamento, composição de fluxos e compartilhamento de banda também são definidos na especificação. Estas regras são bastante complexas e não serão tratadas aqui, devendo o leitor seguir a referência da especificação para maiores detalhes [SPDB95].

2.6.4 *Controlled Load Network Element Service*

O *Controlled Load Network Element Service*, ou Serviço de Carga Controlada, oferece às aplicações o mesmo serviço que é dado ao tráfego *best-effort* quando o enlace está com baixa carga [Wro96]. Desde que a rede esteja funcionando corretamente, assume-se que:

- Uma grande percentagem dos pacotes serão transmitidos com sucesso pela rede aos seus destinos. Esta percentagem deve ser semelhante à taxa média de erro esperado do meio de transmissão.
- O atraso experimentado na transmissão por uma grande percentagem dos pacotes não deve exceder o atraso mínimo esperado no meio de transmissão. Este atraso mínimo inclui o atraso da transmissão na velocidade da luz, mais um tempo fixo de processamento nos roteadores ao longo do caminho do fluxo de dados.

Este tipo de serviço tem a intenção de suportar uma grande variedade de aplicações, principalmente aquelas sensíveis à carga da rede, como as aplicações de tempo real adaptativas.

A implementação deste tipo de serviço pode ser feita utilizando uma grande variedade de técnicas de escalonamento e algoritmos de controle de admissão.

O serviço de Carga Controlada não recebe ou utiliza parâmetros de controle, como perda de pacotes ou atraso. Seu comprometimento está com um serviço equivalente ao tráfego *best-effort*, quando a carga no enlace não é excessiva. Um fluxo que utiliza este serviço em um elemento de rede pode experimentar:

- Um pequeno ou nenhum atraso que seja maior do que o tempo de transmissão de uma rajada de dados (*burst time*). Este tempo é definido como o tempo necessário para que a maior rajada de um fluxo seja transmitida com a taxa de transmissão solicitada para o fluxo.
- Uma pequena ou nenhuma perda por congestionamento, que seja maior do que o especificado no *burst time*. Neste contexto, perda por congestionamento pode ser derivada de processamento, falta de espaço no *buffer* ou na banda do enlace. Embora perdas ocasionais possam ocorrer, perdas significativas consecutivas representam uma falha no Controle de Admissão.

Não é permitida a fragmentação de pacotes. Pacotes maiores que a MTU do enlace devem ser tratados pelo mecanismo de policiamento. Os pacotes que estiverem fora das especificações de tamanho ou que extrapolarem os limites de carga definidos no fluxo serão tratados como *best-effort*. Deve-se tomar cuidado para que o tráfego *best-effort* não venha a alterar significativamente as características do enlace e, por conseguinte, prejudicar o tráfego do serviço de Carga Controlada.

Não é necessário que as implementações façam controle da variação do atraso (*jitter*). Porém seu controle pelo algoritmo de escalonamento é permitido.

O serviço é ativado pela especificação dos parâmetros de tráfego (TSpec) ao elemento de rede. O TSpec é formado pelo ***token bucket***, mais um tamanho mínimo de pacote **m** e o tamanho máximo dos pacotes **M**. As regras de formatação destes parâmetros são as mesmas aplicadas no serviço de Atraso Controlado (Seção 2.6.1).

O Serviço de Carga Controlada não requer o uso de parâmetros de caracterização. Porém, implementações individuais podem fazer algum tipo de monitoramento.

O Serviço de Carga Controlada recebe o número de identificação 5.

O parâmetro de especificação TSpec recebe o número de identificação 1.

Regras específicas de policiamento, composição de fluxos, compartilhamento de banda, exemplos de implementação, formas de avaliação e formato de mensagens também são definidos na especificação. Estas regras são bastante complexas e não serão tratadas aqui, devendo o leitor seguir a referência da especificação para maiores detalhes [Wro96].

2.6.5 *Committed Rate Quality of Service*

O serviço *Committed Rate* [FGD96], ou Comprometimento da Taxa de Transmissão, disponibiliza um tipo de serviço onde é comprometida uma taxa fixa de transmissão, sem perda ou com um mínimo de perda de pacotes do fluxo de dados, ao longo do caminho percorrido pelo fluxo. Este serviço não especifica garantias de atraso, mas assume que o atraso experimentado pelos pacotes não deve ser significativamente maior do que aquele experimentado em um enlace dedicado.

Este serviço é voltado para aplicações que precisam da garantia de que será alocada uma certa quantidade de banda na rede para a emulação de circuitos dedicados. Isto inclui as aplicações que geram tráfego com taxas contínuas.

As garantias oferecidas por este serviço são semelhantes às oferecidas pelo Serviço Garantido (Seção 2.6.2). Porém, a diferença é que não é atribuído o mesmo rigor na garantia de atraso usado pelo Serviço Garantido. O Serviço de Comprometimento de Taxa de Transmissão pode ser visto como um serviço intermediário entre o Serviço Garantido e o Serviço de Atraso Controlado (Seção 2.6.4). Ele oferece menos garantias que o primeiro, mas impõe uma habilidade maior dos elementos de rede, do que no segundo.

Como no Serviço Garantido, não são exportadas informações de caracterização, mas as implementações estão livres para fazê-lo.

A definição do fluxo para o elemento de rede é feita na forma de TSpec e a solicitação de serviço na forma de RSpec. O TSpec é formado pelo ***token bucket***, uma taxa de pico do tráfego (*peak rate*) **p**, um tamanho mínimo de pacote **m** e o tamanho máximo dos pacotes **M**.

Estes parâmetros possuem a mesma formatação dos parâmetros usados no Serviço Garantido (Seção 0).

O **RSpec** é formado pela taxa de reserva **R**. O parâmetro **R** pode ser diferente da taxa **r**, para possibilitar maior flexibilidade na solicitação de serviço que o receptor pode fazer .

Não é permitida a fragmentação de pacotes. Pacotes maiores que a MTU do enlace devem ser tratados pelo mecanismo de policiamento. Os pacotes que estiverem fora das especificações de tamanho ou que extrapolarem os limites de carga definidos no fluxo serão tratados como *best-effort.*

O Serviço Garantido recebe o número de identificação 6.

O **TSpec** é identificado com o número 1 e o **RSpec** com o número 2.

Capítulo 3
RSVP

O RSVP (*Resource Reservation Protocol*) é um protocolo de reserva de recursos proposto inicialmente por Zhang et alli [ZDESZ93]. O Protocolo está sendo desenvolvido por um grupo de trabalho do IETF [HC94], denominado *Resource Reservation Setup Protocol Working Group* [BZ96].

Este protocolo visa possibilitar a integração de serviços dentro da Internet de forma confiável e estável, mantendo a Qualidade de Serviço (QoS) através da reserva de recursos ao longo do caminho dos pacotes. O RSVP é usado por um *host*, a fim de solicitar os recursos necessários para o fluxo de dados gerado por uma aplicação local. Ele também é usado pelos roteadores ao longo do caminho a ser percorrido pelo fluxo, a fim de estabelecer a reserva de recursos ao longo de todo este caminho.

O RSVP solicita recursos em uma direção apenas, tratando o transmissor de forma distinta do receptor, embora a mesma aplicação possa estar agindo das duas formas ao mesmo tempo. O RSVP opera no topo do IP (tanto no IPv4, quanto no IPv6), ocupando o lugar do protocolo de transporte na pilha de protocolos. Porém, ele não transporta os dados da aplicação. Na realidade ele age como um protocolo de controle, como o ICMP (*Internet Control Message Protocol*), o IGMP (*Internet Group Management Protocol*) e os protocolos de roteamento [Com91].

Em cada nó, o RSVP passa uma solicitação de reserva a uma rotina de Controle de Admissão, para verificar se há recursos suficientes disponíveis (Capítulo 4). Se existir, o nó reserva estes recursos no mecanismo de Controle de Tráfego. Uma vez feita a reserva, é necessário que o RSVP também indique o filtro de identificação do fluxo de dados que fará uso da reserva. O Classificador de Pacotes (Capitulo 5) do nó fará a identificação dos pacotes pertencentes ao fluxo que deve receber a reserva, verificará a rota a ser seguida e repassará estes pacotes ao Escalonador de Pacotes (Capítulo 6). Este então, tomará as decisões a respeito da transmissão dos pacotes, para obter a QoS solicitada. A Figura 3.1 ilustra a arquitetura do RSVP em um *host*/roteador.

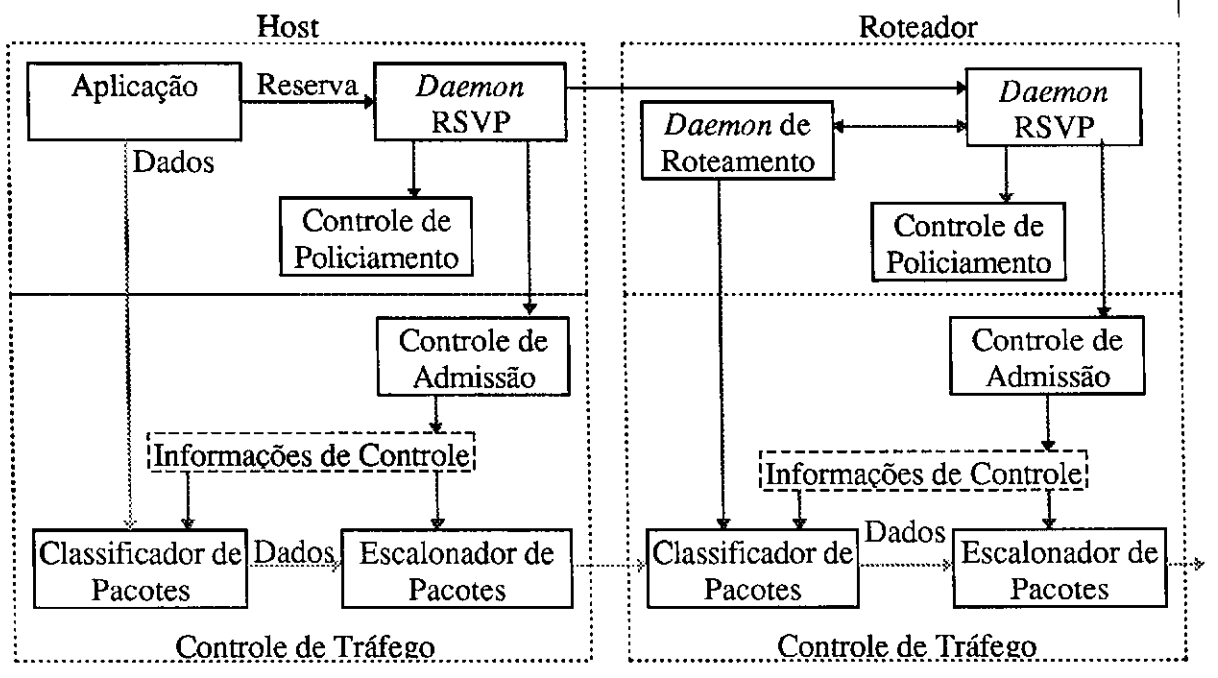


Figura 3.1 - RSVP em *Hosts* e Roteadores

Em cada nó, o RSVP se comunica com dois módulos de decisão locais, o Controle de Admissão e o Controle de Policiamento. O Controle de Admissão verifica se há recursos disponíveis para a solicitação. O Controle de Policiamento determina se o tráfego está preservando o contrato com o nó e se o usuário tem permissão para fazer reservas. Se ambas as pesquisas obtiverem sucesso, o RSVP envia ao Controle de Tráfego as informações necessárias para que o Classificador de Pacotes e o Escalonador de Pacotes possam executar suas atividades. Se a verificação falhar, o RSVP retorna uma notificação de erro à aplicação que gerou a solicitação.

Ele não é por si só um protocolo de roteamento mas, é projetado para operar com os protocolos de roteamento *unicast* e *multicast*. Assim, uma aplicação que deseja obter Qualidade de Serviço se junta a um grupo *multicast* (também pode ser feito com *unicast*) e chama o RSVP para reservar recursos ao longo do caminho usado pelo grupo para a transmissão do fluxo de dados. Diferentemente dos protocolos de roteamento, o RSVP é explicitamente chamado pelas aplicações para a obtenção da QoS desejada.

Para a tradicional comunicação ponto-a-ponto, o remetente, através do RSVP, transmite um aviso de transmissão em direção ao receptor, com os roteadores ao longo do caminho a ser usado pelos dados. Já, nos casos de ponto-a-multiponto, uma forma trivial de estender este paradigma é ter o remetente transmitindo os avisos de transmissão através da árvore de roteamento *multicast*, até cada receptor.

Quando temos multiponto-a-multiponto, cada remetente envia o aviso de transmissão através de sua própria árvore *multicast* até os demais membros do grupo. O problema ocorre quando os receptores possuem características e necessidades que não podem ser satisfeitas com a simples extensão deste paradigma.

Não podemos assumir, por exemplo, que todos os membros de um grupo *multicast* possuem a mesma capacidade de processamento ou que desejam solicitar a mesma qualidade de serviços que os demais (como no caso de transmissões de vídeo para equipamentos com definições diferentes). Ou ainda, o caminho necessário para atingir cada receptor pode não ter a mesma capacidade. Com isso, desejamos que a reserva de recursos seja independente para cada receptor, permitindo a alocação correta de recursos para cada um.

Com isso, podemos identificar algumas das metas que se deseja alcançar ao definir o RSVP [ZDESZ93]:

- Estabelecer a capacidade de receptores heterogêneos solicitarem as reservas de acordo com suas necessidades.
- A troca de membros de grupos de forma simples e segura, sem que se precise reinicializar o grupo de *multicast* a cada troca.
- Permitir ao usuário final especificar o que suas aplicações precisam e como estes recursos podem ser agregados ao longo do caminho da árvore *multicast*, utilizando os recursos da rede de forma mais eficiente.
- Permitir aos receptores a troca dos canais (transmissores) dos quais deseja receber as informações.

- Permitir as trocas de rotas dinamicamente, restabelecendo as reservas de recursos ao longo do novo caminho.
- Controlar o *overhead* para que este não cresça demais a medida que o número de participantes do grupo *multicast* aumente.
- O esquema geral do protocolo deve ser independente das tecnologias (como por exemplo, protocolos de roteamento), sem perder de vista a integração destas tecnologias com o protocolo, para uma maior eficiência do mesmo.

Aplicando estas metas, pode-se chegar aos principais princípios do RSVP:

- **Reserva inicializada pelo receptor**
  - ☞ A reserva é feita pelo receptor que deseja se juntar ao grupo *multicast*, indicando quais recursos deseja, sem afetar os demais membros do grupo. Cabe a este a gerência de sua reserva enquanto permanecer no grupo.
- **Separação da reserva de recurso da filtragem de pacotes**
  - ☞ A reserva de recursos não determina quais pacotes podem usar os recursos, mas simplesmente especifica quais recursos são reservados e por quem. Uma função separada denominada **filtro de pacotes** (*packet filter*) seleciona aqueles pacotes que podem usar os recursos. Isto permite a utilização de estilos de reservas diferentes.
- **Disponibilizar estilos diferentes de reservas**
  - ☞ Para suportar as diferentes necessidades de várias aplicações e usar mais eficientemente os recursos da rede, o RSVP define diferentes estilos de reservas que indicam como os nós intermediários devem agregar as solicitações de reservas dos receptores em um mesmo grupo *multicast*.
- **Manutenção de "*soft state*" na rede**
  - ☞ Durante uma conferência multiponto-a-multiponto, tanto pode ocorrer a inclusão de novos membros, quanto a saída de algum membro. Para ajustar a reserva de recursos de acordo com isto, o RSVP mantém os nós intermediários em ***soft state*** e toma a responsabilidade da reserva de recursos para os usuários finais. Com isso, os nós intermediários podem ser mantidos com dois estados de

informação: um **estado de caminho** e um **estado de reserva**. Cada remetente de dados envia periodicamente uma **mensagem de caminho** que estabiliza ou substitui o estado de caminho, enquanto que cada receptor envia periodicamente uma **mensagem de reserva** que estabelece ou substitui o estado de reserva. Estas mensagens são denominadas **mensagens de *refresh***.

- **Controle de *overhead* do protocolo**
  - ☞ O *overhead* do RSVP é determinado por três fatores: o número de mensagens RSVP enviadas, o tamanho destas mensagens e a frequência de *refresh* das mensagens de caminho e das mensagens de reserva. O RSVP junta as mensagens que serão enviadas pelos enlaces em cada direção, durante cada período de *refresh*. Com isso não mais do que uma mensagem em cada direção é transmitida através dos enlaces. Já, com relação à freqüência de *refresh*, que ajusta os valores de *timeout* levados em cada mensagem, quanto maior o valor de *timeout*, menor será a freqüência de transmissão.
- **Modularidade**
  - ☞ O RSVP deve ser o mais independente dos outros componentes quanto possível. Desta forma, o *flowspec*, o roteamento e o controle de admissão podem ser definidos conforme as condições específicas em cada ambiente.

Assim, temos as principais características gerais que envolvem a operação do RSVP:

- Suporte a *multicast* e *unicast*, com suporte às trocas de membros dos grupos tão bem quanto às trocas de rotas.
- Reserva de recursos para fluxos de dados unidirecionais.
- Ele é orientado ao receptor, ou seja, quem recebe o fluxo de dados é responsável pela inicialização e manutenção da reserva de recursos usada. Este esquema permite ao RSVP acomodar receptores heterogêneos em um grupo *multicast*, onde cada um pode reservar a quantidade de recursos que necessitar, receber diferentes fluxos de dados de um mesmo grupo e trocando de tempos em tempos o transmissor de dados (ou canais) do qual deseja receber os dados, sem alterar suas reservas.

- Ele mantém um "*soft state*" nos roteadores, permitindo o suporte às trocas de membros e adaptação às trocas de roteamento.
- O RSVP não é um protocolo de roteamento, mas depende dos protocolos de roteamento.
- Transporta e mantém as informações do Controle de Tráfego e o Policiamento, sem alterá-las.
- Disponibiliza vários estilos de reservas para permitir uma variedade de aplicações, especificando como as reservas de um mesmo grupo podem ser agregadas nos nós intermediários.
- Disponibiliza operação transparente através dos roteadores que não suportem o protocolo.
- Suporta tanto o IPv4, quanto o IPv6.

Na Seção 3.1 será visto como são definidos os fluxos de dados no RSVP. A Seção 3.2 resume o mecanismo de funcionamento do Protocolo. A Seção 3.3 apresenta os estilos de reserva de recursos definidos. A Seção 3.4 mostra os tipos de mensagens que serão geradas. A Seção 3.5 descreve as interfaces entre o RSVP e os mecanismos complementares para a obtenção da Qualidade de Serviço. E, finalmente, a Seção 3.6 cita as implementações em desenvolvimento e faz uma pequena apresentação da implementação executada pelo ISI (*International Science Institute*).

3.1 Fluxo de Dados no RSVP

Uma solicitação de reserva de recursos do RSVP é descrita por uma definição de fluxo (*flowspec*) junto com uma definição de filtro (*filterspec*). Este conjunto é conhecido como Descritor de Fluxo (*Flow Descriptor*). O *flowspec* é usado pelo Escalonador de Pacotes, enquanto o *filterspec* é usado pelo Classificador de Pacotes.

O controle da Qualidade de Serviço ocorre no transmissor e nos nós intermediários, no lugar onde o fluxo é enviado para o meio físico, embora a reserva RSVP se origine no receptor.

A especificação de fluxo (*flowspec*) em uma solicitação de reserva irá geralmente incluir uma classe de serviço e dois conjuntos de parâmetros numéricos, de acordo com o modelo de Integração de Serviços (Capítulo 2):

- **RSpec**
  - ☞ Define a Qualidade de Serviço desejada.
- **TSpec**
  - ☞ Descreve o fluxo de dados gerado.

Para a identificação dos fluxos de dados o RSVP usa o conceito de **sessão**. Uma sessão é descrita pelo destino e pelo protocolo de transporte usado. O destino da sessão é definido da seguinte forma:

- **DestAddress**
  - ☞ Endereço IP de destino dos pacotes de dados. Pode ser tanto um endereço *unicast*, quanto *multicast*.
- **Identificação do protocolo de transporte**
- **DstPort**
  - ☞ Uma porta genérica de destino do protocolo ou da aplicação. Pode ser definida por um campo de porta TCP ou UDP. No caso de sessões *multicast*, não é obrigatório a definição da porta, já que sessões diferentes podem sempre usar endereços *multicast* diferentes. Mas, como sessões *unicast* podem estar endereçadas a um mesmo *host*, precisam da definição de porta.

Como as portas TCP e UDP são usadas na identificação dos fluxos de dados, cada roteador deve ter a capacidade de examinar estes campos, "abrindo" o datagrama IP. Por isso, geralmente é necessário evitar a fragmentação dos pacotes IP. No Capítulo 5 deste documento, é proposto um mecanismo que identifica os fragmentos de pacotes, sem a necessidade de reagrupá-los, possibilitando aos roteadores a fragmentação de pacotes IP também na reserva de recursos.

São relatados dois casos de problemas que podem surgir na identificação de pacotes a partir das portas:

- O IPv6 insere um número variável de cabeçalhos Internet, também de tamanho variável, antes do cabeçalho de transporte, dificultando a

identificação. A classificação eficiente nestes casos, é a utilização do campo *Flow Label* do cabeçalho IPv6 [Par95].

- O nível de segurança IP sobre o IPv4 ou o IPv6 pode criptografar todo o cabeçalho de transporte, tornando o número da porta ilegível aos nós intermediários. Há porém um estudo em andamento a este respeito.

3.2 Mecanismo de Funcionamento do RSVP

Um grupo *multicast* dentro do contexto *Internet Multicast* [DC85] pode ser formado por um ou mais remetentes de dados que se utilizam de um endereço Internet classe D para enviar os pacotes, enquanto um ou mais receptores recebem os dados neste mesmo endereço. A qualquer instante podem ocorrer inclusões ou saídas de membros de um grupo. Os membros dos grupos podem ser tanto diversos *hosts*, quanto diversos usuários ou aplicações lógicas em um simples *host*.

No RSVP, antes de um remetente iniciar a transmissão dos dados, é enviada uma **mensagem de caminho** (**PATH** *message*), contendo o *flowspec* do remetente. Esta mensagem é usada para o estabelecimento da rota a ser usada pelo fluxo de dados entre a origem e o(s) destino(s).

Ao receber uma mensagem de caminho, os nós intermediários aplicam diversas regras para a verificação das mensagens, e dos mecanismos de controle locais, para então, se necessário, retransmitir a mensagem. O nó somente irá repassar a mensagem de caminho se um transmissor estiver enviando sua primeira mensagem, se houver troca de parâmetros na reserva ou se for preciso executar um *refresh* no caminho, para a manutenção do *soft-state*.

Quando um receptor recebe uma mensagem de caminho de um transmissor do qual deseja receber dados, ele envia uma mensagem de reserva usando o (possivelmente modificado) *flowspec* recebido na mensagem de caminho. A mensagem de reserva é guiada através do caminho inverso que trouxe a mensagem de caminho até atingir o remetente. Se qualquer nó ao longo da rota rejeitar a reserva, uma mensagem de erro é enviada de volta ao receptor e a mensagem de reserva é descartada. Caso contrário, ela é propagada em direção ao transmissor. O receptor pode, se desejar, solicitar uma confirmação da reserva.

Uma vez estabelecida a reserva, o receptor envia periodicamente mensagens de *refresh* idênticas à solicitação original. Como as reservas são retransmitidas através da rota inversa, os nós intermediários combinam as mensagens de um mesmo grupo, eliminando as que carregam reservas de recursos menores ou iguais aqueles já reservados anteriormente.

O modelo de reservas do RSVP é feito em apenas um passo. A máquina receptora envia sua reserva em direção ao transmissor e cada nó no caminho aceita ou rejeita a mensagem. O RSVP suporta um mecanismo conhecido como *One Pass With Advertising* (OPWA) [SB95]. Com ele, o RSVP controla os pacotes que são enviados no sentido dos transmissores, seguindo o caminho dos dados para coletar informações que podem ser usadas para prever a Qualidade de Serviço fim-a-fim. Os resultados (*advertisements*) são enviados pelo RSVP para os *hosts* receptores e às aplicações. Estas informações podem ser usadas pelos receptores para construir ou ajustar dinamicamente a qualidade dos serviços.

Sempre que possível, para reduzir o *overhead* do protocolo, as informações que chegam nas mensagens RSVP para uma determinada sessão são combinadas dentro de uma mensagem que será transmitida. As mensagens que causam uma troca de estado são retransmitidas sem atraso, enquanto que as mensagens de *refresh* podem ser combinadas dentro de poucas mensagens (uma por sessão).

Quando um receptor deseja terminar a conexão, este envia uma mensagem de liberação (*Teardown*) do estado de caminho ou dos recursos reservados. A mensagem é retransmitida nó-a-nó rapidamente, sem atrasos. Não existe qualquer temporizador para esta mensagem. Se algo ocorre com ela, antes de percorrer todo o caminho, os nós intermediários irão automaticamente encerrar a conexão quando o *timeout* expirar sem que um *refresh* tenha sido recebido.

Existem dois tipos de mensagens *teardown*. A mensagem **Path_Tear** é enviada em direção aos receptores e libera o estado de caminho. Já a mensagem **Resv_Tear** é enviada em direção aos transmissores e é usada para a liberação dos recursos. As mensagens somente são retransmitidas para o próximo nó, quando os recursos locais forem totalmente liberados para aquela conexão.

3.3 Estilos de Reservas

Cada solicitação de reserva especifica também um estilo de reserva. Estes estilos consistem do tratamento de reservas para diferentes transmissores numa mesma sessão. Este tratamento pode consistir de reservas distintas para cada um dos transmissores ou reservas compartilhadas por diversos transmissores.

Outra opção de controle de reserva na seleção dos transmissores está na seleção dos transmissores. A reserva pode ser explicita na seleção dos transmissores ou genérica (*wildcard*), que seleciona todos os transmissores de uma mesma sessão. A Tabela 3.3 mostra um esquema comparativo dos estilos de reserva mostrados a seguir, com as opções de controle.

Seleção do Transmissor	Reservas	
	Distintas	Compartilhadas
Explícita	FF	SE
Genérica	(não definido)	WF

Tabela 3.3 - Controle da seleção de estilos de reservas

Os seguintes estilos de reserva estão definidos até o momento:

1. ***Wildcard-Filter (WF) Style***
   - Cria uma ligação simples através de cada *link*, compartilhada pelos pacotes de dados provenientes de todos os transmissores para uma determinada sessão. O tamanho da reserva compartilhada por todos os transmissores será a maior reserva feita por todos os receptores.
2. ***Fixed-Filter (FF) Style***
   - Cria reservas para pacotes de dados provenientes de um determinado transmissor. A solicitação de reservas FF para um determinado receptor $\mathbf{R_j}$ contém uma lista de uma ou mais descrições de fluxo, cada uma consistindo de uma *filterspec*, que especifica vários transmissores $\mathbf{S_i}$ e um *flowspec* (**Q**) correspondente.
3. ***Shared Explicit (SE) Style***
   - Cria uma reserva simples que será compartilhada por diversos transmissores de uma mesma sessão. Diferentemente do WF, o SE

permite a um receptor especificar o conjunto de transmissores que irão compartilhar os recursos.

As reservas compartilhadas WF e SE são apropriadas para as aplicações *multicast* cuja camada de aplicação não permite que todos os transmissores de dados transmitam simultaneamente, como por exemplo, em conferências de audio, onde um número limitado de pessoas fala de cada vez.

As reservas FF podem ser dirigidas às aplicações onde deseja-se receber de cada transmissor uma Qualidade de Serviço definida e que não deve ser compartilhada com os demais transmissores numa mesma sessão, como canais de vídeo.

Podemos exemplificar os estilos da seguinte forma:

- **WF(* {Q})** - Aqui, o "*" representa o filtro *wildcard* (selecionando todos os transmissores) e o "Q" representa o *flowspec* de quantidade "Q" (para simplificar o exemplo).
- **FF(S1{Q1}, S2{Q2}, ...)** - Esta mensagem carrega uma lista de pares (transmissores, *flowspec*).
- **SE((S1, S2, ...) {Q})** - Esta mensagem leva um montante de recursos Q que pode ser compartilhado ao mesmo tempo pelos fluxos de dados provenientes dos transmissores S1, S2, etc.

Tomemos como exemplo a Figura 3.2, onde temos em um roteador quatro interfaces interligando na interface (a) o Transmissor S1, na interface (b) os transmissores S2 e S3, na interface (c) o Receptor R1 e na interface (d) os receptores R2 e R3.

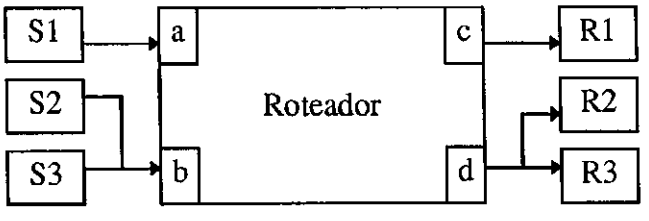


Figura 3.2 - Configuração do roteador

Se utilizarmos o estilo WF na reserva, com R1 solicitando recursos na ordem de 4B, o R2 solicitando 3B e o R3 solicitando 2B, teríamos no final o estado descrito na Tabela 3.4, onde se reservaria em cada interface o valor máximo solicitado.

Transmitido		Reservado		Recebido	
WF(*{4B}) ←	a	c	* {4B}	c	← WF(*{4B})
WF(*{4B}) ←	b	d	*{3B}	d	← WF(*{3B}) ← WF(*{2B})

Tabela 3.4 - Estado de reservas com estilo WF

A Tabela 3.5 mostra o estado do roteador após utilizarmos o estilo FF na reserva, com R1 solicitando recursos na ordem de 4B para S1 e 5B para S2; o R2 solicitando 3B para S1 e B para S3; e o R3 solicitando B para S1.

Transmitido		Reservado		Recebido	
FF(S1{4B}) ←	a	c	S1{4B) S2{5B}	c	← FF (S1{4B}, S2{5B})
FF(S2{5B}, S3{B}) ←	b	d	S1{3B} S3{B}	d	← FF(S1{3B}, S3{B}) ← FF(S1{B})

Tabela 3.5 - Estado de reservas com estilo FF

A Tabela 3.6 mostra o estado do roteador após utilizarmos o estilo FF na reserva, com R1 solicitando recursos na ordem B para S1 e S2; o R2 solicitando 3B para S1 e S3; e o R3 solicitando 2B para S2.

Transmitido		Reservado		Recebido	
SE(S1{3B}) ←	a	c	(S1, S2) {B}	c	← SE ((S1, S2) {B})
SE((S2, S3) {3B}) ←	b	d	(S1, S2, S3){3B}	d	← SE((S1, S3){3B}) ← SE(S2{2B})

Tabela 3.6 - Estado de reservas com estilo FF

As regras do RSVP não permitem a junção de estilos de reservas diferentes para o compartilhamento de recursos de uma mesma sessão. Como resultado disso, os estilos são mutuamente incompatíveis.

3.4 Mensagens RSVP

Uma mensagem RSVP consiste de um cabeçalho comum, seguido por um corpo de objetos tipificados de tamanhos variáveis, mas com regras de formatação bem definidas. Cada mensagem pode transportar um ou mais objetos, de acordo com o tipo de mensagem. O RSVP define os seguintes tipos de mensagens:

- **Mensagem de Caminho** (*Path Message*)
  - ☞ Cada transmissor envia periodicamente uma mensagem de caminho para cada fluxo de dados que irá transmitir no contexto de reserva de recursos. Esta mensagem vai em direção aos nós de destino pelo mesmo caminho que será usado pelo fluxo de dados.
- **Mensagem de Reserva** (*Resv Message*)
  - ☞ As mensagens de reserva carregam as solicitações de reserva de recursos originadas nos receptores em direção aos transmissores, utilizando o caminho inverso da mensagem de caminho.
- **Mensagens de Liberação** (*Teardown Messages*)
  - ☞ As mensagens de liberação são usadas para a liberação dos estados de reserva e de caminho entre a origem e o destino e vice-versa. Existem dois tipos de mensagens de liberação:

    PathTear - Libera o estado de caminho e;

    ResvTear - Libera o estado de reserva.
- **Mensagens de Erro** (*Error Messages*)
  - ☞ Carregam as informações a respeito de problemas que eventualmente tenham ocorrido no estabelecimento do caminho (**PathErr**) e no estabelecimento das reservas (**ResvErr**).
- **Mensagem de Confirmação** (*Confirmation Message*)
  - ☞ Esta mensagem é enviada como confirmação de uma reserva, quando na mensagem de reserva é solicitada tal confirmação. A mensagem de confirmação é enviada diretamente ao *host* usando comunicação *unicast*.

As mensagens RSVP podem ser transmitidas como Raw-IP [Ste94], utilizando a identificação de protocolo número 46, ou encapsulando em datagramas UDP. O encapsulamento UDP é usados pelos sistemas que não implementam Raw ou para atingir subredes conectadas por *firewalls*.

A fragmentação IP é desaconselhável já que pode ocasionar erros característicos. Ao invés, uma fragmentação no RSVP deve ser usada. Isto é, a mensagem com um grande número de descritores deve ser dividida em segmentos que serão colocados em datagramas individuais levando o mesmo cabeçalho. Cada uma destas mensagens serão processadas no destino com um efeito acumulativo no estado local. Nenhuma recomposição da mensagem é necessária no destino.

A especificação do RSVP define o formato e os tipos de objetos que serão transportados pelas mensagens. É definido também o conjunto de regras de transmissão e processamento das mensagens nos *hosts* e nós intermediários. Para maiores detalhes a respeito destes objetos e das regras citadas, consulte a especificação do RSVP [BZBHJ96].

3.5 Interfaces

A especificação do RSVP [BZBHJ96] sugere um conjunto mínimo de interfaces que podem ser utilizadas na implementação do RSVP. Este conjunto de interfaces envolve uma interface com as aplicações (API - *Application Program Interface*), uma interface com o mecanismo de Controle de Tráfego, uma interface com o mecanismo de policiamento, uma interface com o mecanismo de roteamento e uma interface específica para manipulação dos serviços Int-Serv (capítulo 2). Esta manipulação específica dos serviços Int-Serv é usada na comparação de serviços para o compartilhamento de recursos e não será mostrada aqui.

A seguir serão descritas, resumidamente, estas interfaces.

3.5.1 Interface RSVP/Aplicação

A especificação da API RSVP é derivada da especificação da RAPI para SunOS e BSD [BH96]. O RSVP define sua API com 5 (cinco) funções básicas:

- Registro de sessão;
- Definição do transmissor;
- Reserva;
- Liberação e;
- Retorno de eventos.

O registro de sessão cria uma sessão a partir dos parâmetros fornecidos pela aplicação. Os parâmetros são o endereço do destino *unicast* ou *multicast* (DestAddress), o protocolo de transporte usado (ProtocolId), além da definição de um objeto (SESSION_object) a ser repassado a um procedimento que receberá os eventos que retornarem do *daemon* RSVP (Upcall_Proc_addr). A função retorna uma identificação de registro no RSVP (Session-id). O formato da chamada é o seguinte:

SESSION(DestAddress , ProtocolId , DstPort [, SESSION_object] [, Upcall_Proc_addr]) → Session-id

A definição do transmissor irá registrar as características do transmissor e do fluxo de dados. Após esta definição, o RSVP envia a mensagem de caminho. Os parâmetros utilizados são a identificação da sessão (Session-id), o endereço do transmissor, a porta utilizada para transmissão, a especificação do formato dos dados transmitidos e as características do transmissor (Sender_Template), a especificação do tráfego a ser gerado (Sender_Tspec), o TTL (*Time-to-Live*) IP [STE94] máximo a ser utilizado na transmissão (Data_TTL) e as informações a respeito do mecanismo de policiamento empregado (Policy_data). A função não retorna qualquer parâmetro. Os eventuais erros serão enviados assincronamente para o procedimento definido no registro da sessão. O formato da chamada é o seguinte:

SENDER(Session-id [, Source_Address] [, Source_Port] [, Sender_Template] [, Sender_Tspec] [, Data_TTL] [, Policy_data])

A reserva de recursos é ativada através da função RESERV. Esta função possui como parâmetros a identificação da sessão (Session-id), o endereço do receptor (receiver_address), um parâmetro para solicitar a confirmação da reserva (CONF_flag), as informações a respeito do mecanismo de policiamento a ser empregado (Policy_data), o estilo de reserva utilizado

(style) e a especificação de reserva de acordo com o estilo solicitado (style-dependent-params). A função não retorna qualquer parâmetro. Os eventuais erros serão enviados assincronamente para o procedimento definido no registro da sessão. O formato da chamada é o seguinte:

RESERVE(session-id, [receiver_address ,] [CONF_flag ,] [Policy_data ,] style , style-dependent-params)

A liberação de reserva ou de caminho é ativada pela chamada RELEASE, contendo apenas a identificação da sessão (session-id). O formato da chamada é o seguinte:

RELEASE(session-id)

O retorno de eventos é feito através do procedimento definido no registro da sessão. Cada evento além de sua identificação, retorna parâmetros específicos do próprio evento. Como retorno, pode-se ter os seguintes eventos:

- PATH_EVENT
  - ☞ Avisa da chegada de uma mensagem de caminho ou *refresh* de caminho.
- RESV_EVENT
  - ☞ Avisa da chegada de uma mensagem de reserva ou *refresh* de reserva.
- PATH_ERROR
  - ☞ Avisa da ocorrência de um erro no envio da mensagem de caminho. Retorna também as informações referentes ao tipo de erro ocorrido.
- RESV_ERROR
  - ☞ Avisa da ocorrência de um erro no envio da mensagem de reserva. Retorna também as informações referentes ao tipo de erro ocorrido.
- RESV_CONFIRM
  - ☞ Retorna a confirmação de uma reserva.

3.5.2 Interface RSVP/Controle de Tráfego

É definido pelo RSVP a interface a ser utilizada pelas implementações de controle de tráfego para a comunicação com o *daemon* RSVP. A interface deve compreender as seguintes atividades:

- Executar reserva;
- Modificar reserva;
- Excluir reserva;
- Definir filtro à reserva;
- Excluir o filtro da reserva;
- Atualização OPWA e;
- Retorno de eventos.

Para a execução de reserva é utilizada a chamada TC_AddFlowspec. Esta chamada utiliza como parâmetros a interface na qual será feita a reserva (Interface), a especificação dos recursos reservados (TC_Flowspec), a especificação do tráfego (TC_Tspec) e as informações sobre o estado do nó e das reservas (Police_Flags). A chamada retorna ao RSVP a identificação da reserva no *kernel* (RHandle) e, se o Controle de Tráfego atualizou a especificação de fluxo, retorna a nova especificação (Fwd_Flowspec). O formato da chamada é o seguinte:

TC_AddFlowspec(Interface , TC_Flowspec , TC_Tspec , Police_Flags)
→ RHandle [, Fwd_Flowspec]

Para a modificação de uma reserva é usada a chamada TC_ModFlowspec, repassando a interface a ser utilizada (Interface), a identificação da reserva no *kernel* (RHandle), a nova especificação dos recursos reservados (TC_Flowspec), a especificação do tráfego (TC_Tspec) e as informações sobre o estado do nó e das reservas (Police_Flags). Se o Controle de Tráfego atualizou a especificação de fluxo, o RSVP recebe a especificação modificada (Fwd_Flowspec). O formato da chamada é o seguinte:

TC_ModFlowspec(Interface, Rhandle, TC_Flowspec, TC_Tspec, Police_Flags)
→ [Fwd_Flowspec]

Para excluir uma reserva, é usada a chamada TC_DelFlowspec com a interface da reserva (Interface) e a identificação da mesma (RHandle):

TC_DelFlowspec(Interface , RHandle)

A inclusão do filtro na reserva é feito pela chamada TC_AddFilter, repassando a interface (Interface), a identificação da reserva (RHandle), a identificação da sessão (Session) e

a definição do tipo de filtro a ser utilizado (FilterSpec). A chamada retorna ao RSVP uma identificação do filtro (FHandle). O formato da chamada é o seguinte:

TC_AddFilter(Interface , Rhandle , Session , FilterSpec) → FHandle

A exclusão do filtro é feita pela chamada TC_Del_filter, repassando a interface (Interface) e a identificação do filtro (FHandle):

TC_DelFilt(Interface , FHandle)

A coleta de informações do *kernel* para OPWA é feita pela chamada TC_Advertise, repassando a Interface e o formato da informação a respeito do estado do enlace (Adspec). A chamada retorna com as informações sobre o estado da interface para transmissão de pacotes (New_Adspec). O formato da chamada é o seguinte:

TC_Advertise(Interface , Adspec) → New_Adspec

O Controle de Tráfego pode retornar ao RSVP avisos de erro ou eventos através de uma chamada TC_Preempt. Esta função retorna a identificação da reserva onde ocorreu o evento (RHandle) e o código do evento (Reason_code). O formato da chamada é o seguinte:

TC_Preempt( ) → RHandle , Reason_code

3.5.3 Interface RSVP/Policiamento

A especificação do RSVP define um módulo de policiamento que será utilizado pelo RSVP para o controle dos enlaces [Her96c] e pelo mecanismo de Controle de Tráfego [Her96a]. Este módulo é bastante recente e não será discutido neste documento.

A interface entre o RSVP e o mecanismo de policiamento ainda não está definida na especificação RSVP. Porém, há uma especificação separada sugerindo isso [Her96b]. Nesta especificação são descritas as funções de chamada da interface. Tais funções ainda não estão totalmente claras e por isso, não serão apresentadas aqui.

3.5.4 Interface RSVP/Roteamento

O RSVP precisa de suporte do mecanismo de roteamento para a recepção direta dos pacotes destinados ao protocolo número 46 (identificação do RSVP no modelo IP), sem que o *host* tente retransmiti-los.

Além disso, é necessária uma interface, para questionamento de rotas para a atualização das tabelas de estado do protocolo, onde são indicados que pacotes devem seguir para qual rota. As trocas de rotas também devem ser notificadas ao RSVP pelo mecanismo de roteamento.

Outros suportes também são desejáveis, como poder forçar a especificação de uma rota, ignorando o roteamento normal. E também, descobrir e gerenciar as interfaces de saídas, permitindo ou não que os pacotes RSVP saiam por uma determinada interface.

A especificação RSVP sugere uma interface para alguns destes mecanismos. Mas, a maioria ainda está sendo estudada.

3.6 Implementações

Atualmente existem diversas empresas e grupos de pesquisa trabalhando na implementação do RSVP. A seguir estão algumas (reportados à lista de discussão do RSVP até 15/08/96) empresas e pesquisadores que estão fazendo isso (em ordem alfabética):

- Baynetworks
  - ✉ jj@baynetworks.com
- CISCO
  - ✉ rsvp-support@cisco.com
- IBM (Roteadores)
  - ✉ pan@watson.ibm.com
- IBM (*hosts*)
  - ✉ jychu@watson.ibm.com
- Intel
  - ✉ yavatkar@ibeam.intel.com

- ISI (*Information Science Institute*)
  - ✉ isi-rsvp@isi.edu
- Precept
  - ✉ karl@precept.com

Outros fabricantes estão utilizando a implementação feita pelo ISI [ISI96]. Entre eles a Sun Microsystems, que usa o RSVP sobre seu próprio mecanismo de controle de Tráfego [Wak94].

Por ser esta uma implementação aberta (os fontes estão disponíveis) e a única disponível até o momento, todas as referências feitas a respeito da implementação do RSVP serão dirigidas a ela. Além do *daemon* RSVP, a implementação do ISI disponibiliza ferramentas de suporte e teste do protocolo e uma versão dos software de vídeoconferência NV (*Network Video Tool*) [Fre94] e VIC (*Video Conferencing tool*).

A versão atual da implementação ISI é a 4.0a7 e está de acordo com a especificação número 12 do RSVP (a atual é a especificação número 13 [BZBHJ96]). Porém, a versão utilizada na implementação do Controle de Tráfego (Capítulos 4, 5 e 6) e nos testes descritos neste documento (Capítulo 7) é a versão 3.2. Apesar da diferença de versões, a interface com o mecanismo de Controle de Tráfego manteve a mesma estrutura. Com isso, os mecanismos propostos neste documento podem ser portados, com as alterações feitas para a versão 3.2.

O RSVP atualmente implementa apenas os serviços de Carga Controlada (Seção 2.6.4) e Garantido (Seção 0). Para o mapeamento com o Controle de Tráfego o RSVP ainda usa o serviço Predito (Seção 2.6.3), convertendo os parâmetros do serviço de Carga Controlada para este.

3.6.1 Alterações para Comportar o Mecanismo Proposto

Como alteração na implementação do RSVP, é necessário apenas a inclusão da especificação do fluxo usado pelo mecanismo de gerenciamento do Controle de Admissão (Capítulo 4). Esta modificação se dá na especificação do *flowspec*. Na Figura 3.3 estão marcados os locais onde são feitas as alterações.

isps.h

```
80    /* Service specification ("flowspec") - describes quality of service
81     * desired for a flow.
82     */
83
84    /* Basic type of service: determines scheduling algorithm */
85    typedef enum {
86      TOS_GUARANTEED = 1,
87      TOS_PREDICTIVE,
88      TOS_DATAGRAM,
89      TOS_AC,
90    } isps_tos_t;
91
92    /* Per-service RSPEC definitions */
93    typedef struct {
94      u_int32_t g_rate;              /* GUARANTEED: flow rate parameter        */
95    } isps_g_rspec_t;
96
97    typedef struct {
98      u_int32_t p_dly;          /* PREDICTIVE: delay target parameter        */
99                                     /* May also be used as priority..        */
100   } isps_p_rspec_t;
101
102   /* Included for DCRP requests */
103   typedef struct {
104     u_int32_t ac_rate;              /* AC: flow rate parameter              */
105     u_int32_t ac_dly;               /* AC: delay target parameter           */
106   } isps_ac_rspec_t;
107
108
109   typedef union {
110     isps_g_rspec_t rs_g;      /* TOS_ST_GUARANTEED                         */
111     isps_p_rspec_t rs_p;      /* TOS_ST_PREDICTIVE                         */
112     isps_ac_rspec_t rs_ac;    /* ADMISSION CONTROL PARAMETERS              */
113   } isps_rspec_t;
114
115   /* Traffic spec (TSPEC) definitions */
116   typedef struct {
117     u_int32_t tbf_bkt;              /* tbf bucket size: bits                */
118     u_int32_t tbf_rate;             /* tbf rate: bits/sec                   */
119     u_int32_t tbf_buf;              /* tbf reshaping buffer size (currently MBZ)*/
120   } isps_tbf_t;
121
122   typedef struct {
123     isps_tbf_t ts_tb;
124   } isps_tspec_t;
```

Figura 3.3 - Descrição do *flowspec*

3.6.2 Descrição do Fluxo de Dados no *Kernel*

A descrição do fluxo de dados implementada pelo RSVP para o *kernel* consta do TSpec e do RSpec. O Rspec é formado pela estrutura isps_rspec_t e o TSpec pela estrutura isps_tspec_t, conforme mostrado na Figura 3.3.

A estrutura isps_rspec_t é formada pela descrição das estruturas de reserva de cada tipo de serviço implementado:

- Serviço Garantido - isps_g_rspec_t rs_g
  - Taxa de transmissão - g_rate
- Serviço Predito - isps_p_rspec_t rs_p
  - Atraso esperado - p_dly
- Serviço Datagrama (não definido pela estrutura)
  - prioridade - p_dly
- Gerência do Controle de Tráfego - isps_p_rspec_t rs_ac
  - Taxa de transmissão - ac_rate
  - Atraso esperado - ac_dly

A estrutura isps_tspec_t é formada pela estrutura isps_tbf_t, que descreve o *Token Bucket* usado na especificação do fluxo a ser transmitido :

- *Bucket Size* - tbf_bkt
- *Bucket Rate* - tbf_rate
- Específico da implementação do Controle de Tráfego CSZ [CSZ92] e não usado pelo mecanismo proposto - tbf_buf

3.6.3 Especificação do Filtro

A especificação de filtro é feita pela estrutura isps_filt_t, descrita pela estrutura _filt (Figura 3.4). Faz parte desta estrutura a definição do tipo de família [STE95] (f_pf) usada e a identificação da sessão no formato IPv4 (_f_ipv4).

A sessão é formada pelo endereço IP de destino dos pacotes (_f_destaddr), o endereço IP da origem (_f_srcaddr), a porta TCP/UDP do destino (_f_destport), a porta TCP/UDP da origem (_f_srcport) e o protocolo de transporte usado (_f_protocol).

isps.h

```
61    typedef struct _filt isps_filt_t;
62    struct _filt {
63      int f_pf;                   /* Family (only PF_INET now) */
64      union {
65        struct {
66          struct in_addr _f_destaddr; /* IP dest address */
67          struct in_addr _f_srcaddr; /* IP source address */
68          u_int16_t _f_destport;    /* TCP/UDP dest port */
69          u_int16_t _f_srcport;     /* TCP/UDP source port */
70          u_int8_t _f_protocol;     /* Xport protocol (IPPROTO_UDP, etc) */
71        } _f_ipv4;
72      } _f_spec;
73    };
74    #define f_ipv4_destaddr _f_spec._f_ipv4._f_destaddr
75    #define f_ipv4_srcaddr  _f_spec._f_ipv4._f_srcaddr
76    #define f_ipv4_destport _f_spec._f_ipv4._f_destport
77    #define f_ipv4_srcport  _f_spec._f_ipv4._f_srcport
78    #define f_ipv4_protocol _f_spec._f_ipv4._f_protocol
```

Figura 3.4 - Especificação de filtro pela interface do RSVP

3.6.4 Solicitações Entre o RSVP e o *Kernel*

O RSVP implementa uma estrutura chamada ispsreq (Figura 3.5). Esta estrutura é usada para encapsular as solicitações entre o RSVP e o *kernel*, através da função de controle IOCTL [Ste95].

Esta estrutura é formada pelo nome da interface (iq_name), a identificação de uma função a ser executada no *kernel* (iq_function) (Seção 4.2.5), a identificação da reserva (iq_handle), a identificação do filtro (iq_fhdl) e os parâmetros específicos de cada uma das funções a serem executadas (iq_iqu).

isps.h

```
225   struct    ispsreq {
226     char iq_name[IFNAMSIZ];          /* if name, e.g. "en0"     */
227     short iq_function;               /* function code           */
228     short iq_flags;                  /* further info about fcn  */
229     isps_handle_t iq_handle;         /* call/ret - object handle */
230     isps_handle_t iq_fhdl;           /* filter handle           */
231     union {
232           isps_fs_t _iq_fs;          /* ADDFLW: flow spec       */
233           isps_filt_t _iq_filt;      /* SETFILT: filter         */
234           unsigned long _iq_delay;   /* ADDCLS: delay target    */
235           isps_stdesc_t _iq_sdesc;   /* ADDSHARE: share desc    */
236           isps_intdesc_t _iq_idesc;  /* CTL: i'face desc        */
237     } _iq_iqu;
238   };
```

Figura 3.5 - Estrutura ispsreq

Os parâmetros específicos que irão nos interessar são os que descrevem o fluxo de dados (_iq_fs) e o filtro (_iq_filt). O _iq_filt é descrito na Seção 3.6.3 e o _iq_fs é formado pela estrutura isps_fs_t mostrada na Figura 3.6.

isps.h

```
141  typedef struct {
142     isps_tos_t fs_tos;          /* service type                          */
143     isps_rspec_t fs_rspec;      /* rspec parameters for requested svc     */
144     isps_tspec_t fs_tspec;      /* policing parameters                   */
145     isps_plc_t fs_plc;          /* policing algorithm selector           */
146     isps_handle_t fs_handle;    /* handle of share tree node associated  */
147                                 /*  with this flow                       */
148  } isps_fs_t;
```

Figura 3.6 - Estrutura isps_fs_t

A estrutura isps_fs_t é composta do tipo de serviço (fs_tos), da descrição da reserva (fs_rspec), da descrição do tráfego (fs_tspec), do algoritmo de policiamento utilizado (fs_plc) e da identificação de associações do mecanismo CSZ (fs_handle). O fs_plc e o fs_handle não são usadas pelo mecanismo de Controle de Tráfego proposto.

Capítulo 4
Controle de Admissão

Como parte do modelo de reserva de recursos, o protocolo responsável pela reserva de recursos (no caso, o RSVP), invoca um mecanismo conhecido como Controle de Admissão. Este mecanismo é o responsável pelo gerenciamento dos recursos da rede, cabendo a ele a aceitação ou não de uma solicitação de reserva. No modelo utilizado pelo RSVP, o controle de admissão se localiza no *kernel* do Sistema Operacional, como representado na Figura 4.1.

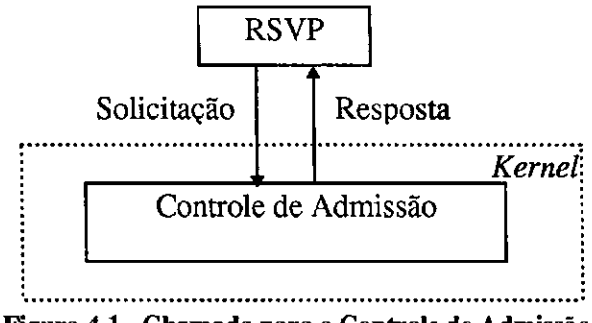


Figura 4.1 - Chamada para o Controle de Admissão

Para o Controle de Admissão funcionar, é preciso que haja um controle rigoroso sobre o meio de transmissão. Não podem ser alocados, por exemplo, mais recursos do que os disponíveis na rede.

Quando os enlaces gerenciados são do tipo ponto-a-ponto, o gerenciamento de recursos se restringe ao controle do tráfego que será transmitido, pela máquina, no enlace. Porém, se a rede que está sendo controlada é do tipo compartilhada, como a Ethernet, é necessário um mecanismo que gerencie não só o tráfego de uma máquina, mas o de toda a rede. A administração dos recursos em uma rede compartilhada como a Ethernet exige que todas as máquinas conectadas à rede utilizem o mesmo mecanismo de controle ou mecanismos compatíveis entre si.

Há atualmente, um grupo de trabalho responsável pela tentativa de padronização de modelos que adaptem as tecnologias de rede mais utilizadas, para que venham a utilizar reserva de recursos na Internet. Este grupo é o ISSLL (*Integrated Services over Specific Link Layers*) [CW96]. Entre as tecnologias de rede sendo estudadas está a Ethernet. Porém, o mecanismo aqui proposto foi idealizado antes da iniciativa do ISSLL e esperamos em breve poder inclui-lo nas discussões deste grupo de pesquisa.

O modelo proposto neste capítulo é composto pelo mecanismo de Controle de Admissão e um mecanismo de gerenciamento dos recursos da rede.

O Controle de Admissão está localizado no *kernel* do Sistema Operacional e é ativado por chamadas de controle, contendo comandos específicos. Existem dois tipos de chamadas. Uma para manutenção das reservas e outra para a inicialização e gerenciamento do próprio Controle de Admissão. Os comandos e rotinas definidos neste modelo são discutidos na Seção **4.2.**

O gerenciamento dos recursos da rede é realizado por um protocolo de controle, chamado **DCRP** (*Distributed Control Reservation Protocol*). O protocolo é executado por um programa chamado **ACD** (*Admission Control Daemon*), que funciona na camada de aplicação e controla de forma distribuída não só a banda, mas também o atraso experimentado na rede. Para monitorar os recursos da máquina, o **ACD** consulta constantemente as informações do Controle de Admissão e toda vez que há alterações, comunica isso através do **DCRP**.

Além do mecanismo proposto, existe um outro sendo atualmente elaborado pela equipe responsável pelo estudo de soluções para as redes Ethernet, dentro do ISSLL. O mecanismo proposto é o **SBM** (*Subnet Bandwidth Manager*) [YPH96].

Neste capítulo será discutido que tipos de recursos devem ser gerenciados na rede (Seção 4.1), será mostrado o modelo de Controle de Admissão e a sua implementação (Seção 4.2), será mostrado como funciona o DCRP e a sua implementação (Seção 4.3) e uma breve apresentação do SBM (Seção 4.4).

4.1 Gerenciamento de Recursos

Em um ambiente compartilhado como o que queremos implementar, existem fluxos provenientes de diversas máquinas. Neste caso, deve haver um mecanismo capaz de manter em suas tabelas o montante de recursos utilizados por todas as máquinas ligadas na rede. Isto significa que uma máquina deve saber as características do tráfego que será gerado na rede pelas demais máquinas.

Existem diversos trabalhos a respeito do desempenho das redes Ethernet. Em [Tan94] há um resumo para o cálculo da eficiência da rede, conforme a carga. São usados como parâmetros de cálculo, o número de estações ligadas à rede, o tamanho (em metros) da rede, o tamanho dos quadros transmitidos, a velocidade de propagação da rede, etc. Também é traçado um gráfico comparando a eficiência do canal com o número de estações e o tamanho dos quadros. Quando o número de estações chega a trinta e duas, a eficiência tende a se estabilizar. Com quadros de 64 *bytes*, a eficiência da rede não chega a 50% no melhor caso. Com mais de 32 estações transmitindo, a eficiência cai para 25%. Mas, com quadros de 1024 *bytes*, a eficiência pode passar dos 90%, ficando em torno dos 85% com mais de 32 estações.

Para estimar o atraso na rede Ethernet, pensou-se inicialmente nas seguintes opções:

- utilização de uma função que descreva o atraso baseada na carga reservada, no número de máquinas conectadas, no tamanho do barramento, etc. Ou;
- restringir a banda disponível e o número de máquinas conectadas à rede a um patamar onde possa ser minimizado o problema das colisões.

A primeira opção pode fornecer uma expectativa de atraso de forma mais exata, de acordo com os recursos alocados. Porém, a execução de cálculos complexos no *kernel* de um Sistema Operacional pode gerar ainda mais atraso. Neste caso, o atraso maior poderia ser acarretado não pelo acesso ao meio, mas pelo próprio cálculo. Contudo, esta opção não deve ser descartada por completo, merecendo um estudo mais aprofundado no futuro.

A segunda opção implica na experimentação da utilização dos recursos da rede com diversos tipos de tráfego, para que se possa concluir qual a limitação ideal para as classes de serviço que se deseja oferecer numa rede deste tipo.

Como ainda não há conclusões seguras a respeito de tal tipo de restrição, optou-se por um modelo simples de controle de atraso, até que um modelo definitivo seja especificado. Para isso, é verificado periodicamente o atraso experimentado na rede e é calculado o atraso médio.

Esta verificação deve ser simples o suficiente para evitar aumento significativo da carga na rede. Utiliza-se então uma simples "regra de três" para se obter uma previsão de atraso na rede. Através de uma equação simples, obtemos um coeficiente de atraso que será utilizado na previsão do atraso quando uma nova reserva é solicitada.

$$Coeficiente\ de\ Atraso = \frac{Atraso\ Médio\ Experimentado}{Banda\ Total\ Reservada\ na\ Rede}$$

Com este coeficiente pode-se estimar o novo atraso a ser experimentado quando um novo fluxo de dados for colocado na rede. Este cálculo também deve ser suficientemente simples para se evitar o comprometimento da capacidade de processamento do equipamento:

*Atraso Estimado = (Banda Total Reservada+ Banda da Nova Solicitação)*Coef. de Atraso*

Esta estimativa de atraso é então utilizada pelo Controle de Admissão para a aceitação ou não de uma reserva, conforme a Classe de Serviço solicitada.

O ponto principal do Controle de Admissão são as regras de admissibilidade do tráfego. Como cada Classe de Serviço define suas próprias variáveis e regras de admissibilidade (Seção 2.6), o RSVP e/ou o controle de admissão devem ser capazes de "mapear" cada classe de serviço solicitada para o tipo de serviço oferecido pela rede. No modelo aqui proposto, utiliza-se apenas duas variáveis para a descrição dos fluxos:

- Banda (*rate*):
  - ☞ Taxa reservada pelo(s) receptor(es) deve ser menor ou igual à taxa especificada pelo transmissor. A soma de todas as banda reservadas não deve ser superior à banda máxima da rede.
- Atraso (*delay*):
  - ☞ O atraso solicitado em uma reserva não pode ser inferior ao atraso médio experimentado na rede. A estimativa de atraso após a aceitação da reserva não deve ser superior ao menor atraso já reservado na rede.

O mapeamento dos serviços deve ser de acordo com o grau de confiabilidade que a rede pode oferecer ao usuário. Nas especificações dos serviços, utiliza-se normalmente o atraso máximo como parâmetro para a aceitação de um fluxo. Nas redes Ethernet este atraso

máximo é difícil de ser controlado e se, for levado em conta, provocará a recusa da maioria dos serviços atualmente definidos.

Em [BS95] é sugerida uma forma de mapeamento dos serviços, estipulando um serviço como **confiável** ou **não confiável** para as redes Ethernet. A proposição inicial é que o serviço oferecido por este modelo seja tratado como **não confiável**. Ou seja, **não haverá garantia plena da Qualidade de Serviço** solicitada. As aplicações deverão levar em conta esta restrição.

A aceitação de um novo fluxo pode implicar num aumento de tráfego e este aumento provocar um atraso maior do que o menor atraso reservado por um fluxo. Isto acarretará a transmissão dos dados de uma reserva já feita com um atraso maior do que o solicitado. Cabe ao Controle de Admissão analisar as informações disponíveis e, mesmo com a restrição de serviço não confiável, se a aceitação de uma nova reserva comprometer o funcionamento das demais, ela deverá ser recusada.

Qualquer fluxo de dados que não possua reserva de recursos é tratado como tráfego *best-effort*. Este tipo de tráfego não pode receber o mesmo tratamento de um fluxo de dados com reservas. Mas, também, não pode deixar de ser transmitido.

Para o tratamento do tráfego *best-effort*, é proposto neste documento um mecanismo denominado ***buffer* elástico**. Este mecanismo atribui uma reserva mínima para o tráfego *best-effort*. Além disso, as rajadas típicas deste tipo de tráfego serão tratadas e enviadas sempre que houver recursos disponíveis e isto não prejudicar os demais fluxos de dados.

Tomou-se como *default*, a banda de 32 Kbps (valor usado como reserva mínima pela interface com o Controle de Tráfego, definida no RSVP), como a reserva inicial do tráfego *best-effort* para redes compartilhadas, no nosso caso, a rede Ethernet. Para os enlaces ponto-a-ponto, optou-se por definir 10% da banda total para este tipo de tráfego. Porém, somente uma experimentação exaustiva poderá comprovar se estes valores são realmente os ideais. Espera-se que em implementações futuras do mecanismo, isso possa ser alterado pelo gerente de rede, de acordo com o tráfego local. No Capítulo 5 o mecanismo de reserva para tráfego *best-effort* será descrito mais detalhadamente.

Tanto para o tráfego com reservas, quanto para o tráfego *best-effort*, há a necessidade da definição do tamanho máximo do *buffer* que será usado em cada uma das reservas. O tamanho do *buffer* é calculado levando-se em conta a banda solicitada e o maior atraso suportado. O *buffer* não poderá ser maior do que o necessário para transmitir a taxa reservada,

sem provocar um atraso maior do que o desejado. O cálculo do tamanho do *buffer* utiliza a seguinte fórmula:

Buffer = (Banda Reservada) * (Atraso Reservado)

O tamanho do *Buffer* é calculado em *bytes*, a Banda Reservada é representada em *bytes* por segundo e o atraso em fração de segundos (atraso em milissegundos / 1000). Esta fórmula é uma variação do cálculo para o tamanho máximo de filas de escalonamento utilizado em [PP88], para um único enlace.

Se o *buffer* encher, isso significa que a taxa de chegada dos pacotes é maior do que a taxa reservada para transmissão. Neste caso, o fluxo não está de acordo com o combinado e medidas de policiamento devem ser tomadas. Isso é atribuição do Escalonador de Pacotes que será apresentado no Capítulo 6.

No mecanismo de ***buffer* elástico** do tráfego *best-effort*, há dois tamanhos de *buffer*. O primeiro é em função do atraso máximo desejável para o tráfego e o segundo em função do atraso máximo permitido. O atraso máximo desejável foi definido inicialmente em 1 (um) segundo e o atraso máximo permitido em 4,096 segundos. O primeiro valor foi escolhido levando-se em conta o bom senso e se mostrou razoável nos testes (Capítulo 7). O segundo valor é definido na implementação do RSVP como o atraso máximo, que pode ser solicitado.

4.2 Mecanismo de Controle de Admissão

As máquinas que se propõem a trabalhar em ambientes com reserva de recursos devem possuir um mecanismo para o controle dos recursos de rede sob sua responsabilidade. Este mecanismo é o Controle de Admissão.

As reservas de recursos podem ser feitas diretamente por uma aplicação, através de um protocolo de transporte como o RTP (*Real-time Transport Protocol*) [SCFJ96] ou através do RSVP.

Cada classe de serviço definida pelo Int-Serv possui características diferentes (Capítulo 2). O Controle de Admissão deve ser capaz de identificar serviços que irá suportar, transportando os parâmetros utilizados por cada serviço para seu próprio formato. Este processo é conhecido como "mapeamento" dos serviços.

Para que o Controle de Admissão possa funcionar no *kernel*, é necessário a inclusão de algumas rotinas e a adaptação de outras do Sistema Operacional. A Tabela 4.1 indica quais arquivos são usados na implementação do Controle de Admissão.

Arquivo	Função
/sys/sys/sockio.h	Definições do *socket*.
/sys/net/if.c	Controle de interrupções.
/sys/net/ac.h	Arquivo com as estruturas e definições do mecanismo de Controle de Tráfego.
/sys/net/isps.h	Arquivo com as estruturas e definições da interface com o Controle de Tráfego.
/sys/net/ac_hash.h	Arquivo com as definições da tabela *hash* de reservas.
/sys/net/ac_isps.c	Rotinas do Controle de Admissão.

Tabela 4.1 - Arquivos utilizados na implementação do Controle de Admissão

A seguir será visto como é definido e implementado o mapeamento dos serviços (Seção 4.2.1), como é feita a adaptação do Sistema Operacional para comportar o Controle de Admissão (Seção 4.2.2), as estruturas de dados utilizadas pelo mecanismo (Seção 4.2.3), como se processa o gerenciamento das informações sobre cada interface (Seção 4.2.4) e finalmente, como é feita a manutenção das reservas de recursos (Seção 4.2.5).

4.2.1 Mapeamento dos Serviços

Na especificação RSVP está definida a interface entre o RSVP e o Controle de Tráfego. Na implementação feita pelo ISI (Seção 3.6), a interface com o Controle de Tráfego estabelece apenas 3 tipos de tráfego: *Guaranteed*, *Predictive* e *Datagram*. Os dois primeiros são bem definidos. Porém, o terceiro não possui qualquer documentação. Levando isto em conta, somente os dois primeiros serviços foram mapeados:

- *Guaranteed*:
  - ☞ Para este serviço, o único parâmetro reservado é a banda. Neste caso, o atraso utilizado deve ser o menor possível para suportar tanto as aplicações de tempo real, quanto o serviço *best-effort*. Como é comum definir como atraso máximo nas aplicações de tempo real o tempo de 100 (cem) milissegundos, será este o atraso padrão reservado para este tipo de serviço.

- *Predictive*:

  ☞ Neste serviço o mapeamento é mais fácil, pois, é passada a banda e o atraso ao Controle de Admissão.

Além destes dois serviços, também são mapeadas as solicitações de reserva executadas por parte do **ACD**. Seu mapeamento é bem mais simples pois, o Controle de Admissão usa os mesmos parâmetros do **ACD**.

A Figura 4.2 representa a implementação deste mapeamento pelo Controle de Admissão. As estruturas usadas na solicitação de reservas são as mesmas usadas pela interface do RSVP com o Controle de Tráfego, apresentadas na seção 3.6.

ac_isps.c

```
216  int tos_test(i_fs, new_flow)
217  isps_fs_t i_fs;
218  flow_spec *new_flow;
219  {
220          switch(i_fs.fs_tos){
221          case TOS_GUARANTEED:
222                  /* It's an unreliable replacement */
223                  new_flow->res_rate=i_fs.fs_rspec.rs_g.g_rate;
224                  new_flow->res_delay = QOS_DELAY;
225                  if (new_flow->res_rate < ISPS_FLOW_MIN_RATE)
226                          new_flow->res_rate = ISPS_FLOW_MIN_RATE;
227                  break;
228          case TOS_PREDICTIVE:
229                  /* It's an unreliable replacement */
230                  new_flow->res_delay = i_fs.fs_rspec.rs_p.p_dly;
231                  new_flow->res_rate = i_fs.fs_tspec.ts_tb.tbf_rate;
232                  if (new_flow->res_rate < ISPS_FLOW_MIN_RATE)
233                          new_flow->res_rate = ISPS_FLOW_MIN_RATE;
234                  if (new_flow->res_delay == 0)
235                          new_flow->res_delay = QOS_DELAY;
236                  break;
237          case TOS_DATAGRAM:
238                  /* not implemented */
239                  return(-1);
240                  break;
241          case TOS_AC:
242                  new_flow->res_rate = i_fs.fs_rspec.rs_ac.ac_rate;
243                  new_flow->res_delay = i_fs.fs_rspec.rs_ac.ac_dly;
244                  break;
245          default:
246                  /* Unknow TOS */
247                  return(-1);
248                  break;
249          }
250          return(0);
251  }
```

tos_test()

Figura 4.2 - Mapeamento dos serviços pelo Controle de Admissão

216-218 A função **tos_test()** tem como entradas a descrição do fluxo e o ponteiro para o fluxo no formato do mecanismo de Controle de Tráfego.

220	Testa cada tipo de fluxo definido na implementação RSVP.
221-227	Se for um *Guaranteed Service*, atribui à taxa o valor reservado e ao atraso o menor atraso para este tipo de serviço (como acabou de ser apresentado). Se a taxa for menor que a menor taxa permitida, atribui a menor taxa permitida.
228-236	Se o serviço for do tipo *Predictive Service*, atribui a taxa e o atraso solicitados ao fluxo a ser reservado. Se a taxa for menor que a menor taxa permitida, atribui a menor taxa permitida. Se o atraso for 0 (zero), pode significar um serviço do tipo *Controlled Load Service*. Neste caso, deve-se assumir que a rede não deve ficar carregada e atribui-se o mesmo atraso do *Guaranteed Service*.
237-240	Para o serviço tipo Datagrama, ainda não há solicitações RSVP. Por isso, não foi implementado.
241-244	Solicitação do DCRP. O mapeamento é direto.
245-248	Serviço desconhecido. Deve ser recusado.

4.2.2 Adaptação do IP

Todos os sistemas operacionais disponibilizam serviços para que os processos da camada de aplicação possam se comunicar com o *kernel*. Estes serviços são executados por funções conhecidas como chamadas do sistema ou *system calls*. A chamada utilizada para o controle de dispositivos de rede é o **ioctl()** [Sun90]. Esta função irá executar no *kernel* os comandos especificados na estrutura de dados contida na chamada.

Na implementação do Controle de Admissão o **ioctl()** é usado tanto para o gerenciamento dos recursos das interfaces de rede, quanto para a manutenção de reservas. Para isso, são definidos dois comandos distintos:

- SIOCIFACCTL
  - ☞ É usado para o gerenciamento das interfaces de rede.
- SIOCIFISPSCTL
  - ☞ É usado para a manutenção das reservas.

A inclusão destes dois comandos implica em uma alteração do arquivo sockio.h (Figura 4.3) e do programa if.c (Figura 4.4) utilizados pelo *kernel* do Sistema Operacional. Os comandos e estruturas utilizadas também precisam ser carregadas por estes programas.

sockio.h

No final do arquivo **sockio.h** são acrescentadas as definições para as chamadas ioctl via socket, conforme a Figura 4.3.

sockio.h

Código original

```
88    #ifndef SIOCISISPSCTL
89    #if defined(sun) && !defined(__GNUC__)
90    #define SIOCIFISPSCTL   _IOWR(i, 70, struct ispsreq)
91    #else
92    #define SIOCIFISPSCTL   _IOWR('i', 70, struct ispsreq)
93    #endif
94    #endif
95
96    #ifndef SIOCIFACCTL
97    #if defined(sun) && !defined(__GNUC__)
98    #define SIOCIFACCTL     _IOWR(i, 90, struct ac_req)
99    #else
100   #define SIOCIFACCTL     _IOWR('i', 90, struct ac_req)
101   #endif
102   #endif
```

Código original

Figura 4.3 - Adaptações no socket para o Controle de Admissão

88-94 Definição da chamada para a reserva de recursos.

96-102 Definição da chamada para a manutenção das interfaces.

ifioctl()

Na função **ifioctl()** é necessário o acréscimo dos testes de comando para as duas novas chamadas, conforme a Figura 4.4.

if.c

Código original

```
577     case SIOCIFISPSCTL:
578             return (isps_ioctl(so, cmd, data, p));
579     case SIOCIFACCTL:
580             return(ac_ioctl(data, ifp));
```

Código original

ifioctl()

Figura 4.4 - Adaptações da chamada ioctl() para o Controle de Admissão

577-578 Chama a função **isps_ioctl()**, responsável pelo gerenciamento das reservas.

579-580 Chama a função **ac_ioctl()**, responsável pelo gerenciamento das interfaces.

4.2.3 Estrutura de Controle das Interfaces

Para o gerenciamento dos recursos de cada interface de rede conectada ao equipamento é necessária uma estrutura de dados que armazene as informações básicas sobre a interface e os dados a respeito dos recursos que serão gerenciados. Esta estrutura de dados é constituída por uma lista onde cada entrada corresponde a uma interface. Esta lista é construída no *kernel* com informações provenientes do *daemon* de controle da rede (**acd**).

A construção e manutenção da lista de interfaces é feita através de interrupções **ioctl()** utilizando a opção SIOCIFACCTL. As solicitações são formatadas dentro da estrutura ac_req, definida no arquivo **ac.h** e representada na Figura 4.5.

ac.h

```
157   /*
158    * Admission Control management ioctl structure
159    */
160   typedef struct ac_req {
161           char acr_name[IFNAMSIZ]; /* Interface name */
162           int acr_cmd;             /* AC kernel command */
163           struct AC_obj acr_obj;   /* AC object */
164   } ac_isps_req;
165
166   /*
167    * Admission Control management commands
168    */
169   #define  IF_ACGET      1       /* Get interface AC data object at kernel*/
170   #define  IF_ACSET      2       /* Set interface AC data object at kernel*/
```

Figura 4.5 - Estrutura ac_req do Controle de Admissão

161-165 A estrutura ac_req tem como campos o nome da interface (acr_name), o comando a ser executado (cmd) e a estrutura da interface que está sendo gerenciada (acr_obj).

170-171 São definidos dois comandos nesta estrutura: o IF_ACGET, que busca no *kernel* as informações referentes à interface acr_name e coloca na estrutura acr_obj; e o IF_ACSET que armazena no *kernel* as informações referentes à interface acr_name trazidas na estrutura acr_obj.

AC_obj

A estrutura mais importante na implementação do controle de tráfego é a AC_obj, que irá montar a lista de interfaces. Ela pode ser vista na Figura 4.6.

ac.h

```
141   struct AC_obj {
142           struct AC_obj     *ac_next_interface; /* next occurrence */
143           char              ac_name[IFNAMSIZ]; /* interface name */
144           struct ifnet      *ac_ifp;  /* ifnet structure pointer */
145           int               ac_interface;      /* interface index */
146           u_char            ac_flags; /* flags */
147           u_int32_t         ac_max_rate;       /* maximum interface rate */
148           struct d_table    ac_datagram;       /* datagram queue */
149           u_int32_t         ac_avg_delay;      /* average delay probed */
150           flow_spec         ac_tot_flow;       /* total reserved flow */
151           flow_spec         ac_loc_flow;       /* local reserved flow */
152           res_list  *ac_list_head;     /* reserve list head */
153           res_list  *ac_list_tail;     /* reserve list tail */
155   } ac_isps_obj;
```

Figura 4.6 - Estrutura AC_obj do Controle de Admissão

142 Aponta para a próxima interface da lista.

143 Identificação da interface, correspondendo ao nome do tipo controladora de rede, seguido pelo número da interface (ex: ed0, sl1, ln0, etc).

144 Estrutura ifnet que contém todos os detalhes sobre a interface. Esta estrutura será usada na classificação e transmissão dos pacotes.

145 Índice da interface. Segue o mesmo índice utilizado na estrutura ifnet.

146 *Flags* utilizados pelo sistema.

147 Taxa máxima de transmissão suportada pela interface.

148 Estrutura contendo informações sobre o tráfego *best-effort*, utilizada na Classificação e Escalonamento dos Pacotes.

149 Atraso médio experimentado na rede.

150 Fluxo total reservado na rede. Isto inclui o tráfego de todas as máquinas ligadas à interface de rede.

151 Fluxo total reservado localmente, indicando o que foi reservado para transmissão.

152-153 Os campos ac_list_head e ac_list_tail apontam respectivamente para o início e o final da lista de reservas da interface. Isto inclui tanto as reservas locais, quanto as remotas. Cada máquina conectada à rede possui uma entrada nesta lista, com o montante de reservas de cada uma delas.

res_list

As reservas são colocadas em uma lista montada pela definição de estrutura res_list, apresentada na Figura 4.7.

ac.h

```
74  typedef struct RES_table {
75          struct    RES_table *res_next;      /* next occurrence */
76          struct    AC_obj *res_obj;  /* AC_obj pointer */
77          isps_fs_t res_spec; /* sender TSpec */
78          isps_filt_t       res_filt;  /* filter spec */
79          flow_spec         res_flow; /* reserved flow spec */
80          short             res_flags;        /* flags */
81          struct sc_queue   res_queue;        /* schedule queue */
82  } res_list;
```

Figura 4.7 - Estrutura res_list do Controle de Admissão

75 Aponta para a próxima reserva na lista.

76 Aponta para a interface a qual pertence a reserva.

77 Especificação da transmissão (Figura 3.6).

78 Filtro do fluxo (Figura 3.4).

79 Fluxo reservado.

80 *Flags* da reserva.

81 Fila de pacotes usada pelo Escalonador de Pacotes (Figura 6.2).

4.2.4 Manutenção da lista de interfaces

A manutenção da estrutura **AC_obj** é relativamente simples. Como apresentado na Figura 4.8, as requisições SIOCIFACCTL indicam o tipo de operação que deve ser executada sobre a estrutura da interface. Existem dois subcomandos: AC_SET para a inicialização e manutenção das informações da interface; e AC_GET para buscar as informações sobre a interface.

ac_ioctl()

A função **ac_ioctl** é chamada a partir da função **ioctl** e executa os comandos pertinentes ao gerenciamento dos recursos da interface de rede.

ac_isps.c

```
606    int
607    ac_ioctl(data, ifp)
608    caddr_t data;
609    struct ifnet *ifp;
610    {
611    register struct ac_req *acr;
612    register struct AC_obj *acp;
613    register struct AC_obj *aco;
614    int s;
615    int error=0;
616
617
618            s = splnet();
619            acr = (struct ac_req *) data;
620            aco = (struct AC_obj *)&acr->acr_obj;
621            acp = ac_find(aco->ac_name);
622            switch (acr->acr_cmd) {
623            case IF_ACGET:
624              if (acp == NULL){
625                      error = ENXIO;
626                      break;
627              }
628              *aco = *acp;
629              break;
630            case IF_ACSET:
631              if (acp == NULL) {
632                      acp = (struct AC_obj *) malloc(sizeof(ac_isps_obj),M_IFADDR, M_WAITOK);
633                      bzero((struct AC_obj *)acp, sizeof(ac_isps_obj));
634                      if (!acp) {
635                        error = ENXIO;
636                        break;
637                      }
638                      store_ac_obj(acp, &AC_LAST, &AC_START);
639                      sprintf(acp->ac_name, "%s", aco->ac_name);
640                      acp->ac_ifp = ifp;
641                      acp->ac_interface = aco->ac_interface;
642                      acp->ac_datagram = aco->ac_datagram;
643                      acp->ac_min_delay = aco->ac_min_delay;
644                      acp->ac_tot_flow = aco->ac_tot_flow;
645                      acp->ac_loc_flow = aco->ac_loc_flow;
646                      acp->ac_list_head = NULL;
647                      acp->ac_list_tail = NULL;
648              }
649              acp->ac_flags = aco->ac_flags;
650              acp->ac_max_rate = aco->ac_max_rate;
651              acp->ac_avg_delay = aco->ac_avg_delay;
652              tc_isps_on = 1;
653              if (!(acp->ac_flags & IS_AC_WAIT))
654                      wakeup((caddr_t)&wakeup_ac_wait);
655              break;
656            }
657            splx(s);
658            return(error);
659    }
```

ac_ioctl()

Figura 4.8 - Função ac_ioctl() para gerenciamento do Controle de Admissão

606-609 A função **ac_ioctl()** recebe como parâmetros a estrutura ac_req representada por data e o apontador para a estrutura ifnet.

611-616	Como variáveis locais, é utilizado: um apontador para a requisição ac_req (acr); dois apontadores para a estrutura de lista AC_obj (acp e aco); uma variável inteira usada para definição de prioridades dos processos do sistema operacional (s); e uma variável inteira onde é armazenado o código de erro (se houver).
618	Muda a prioridade do processo para **splnet()** (*network protocol processing*) [WS95].
619	Coloca em acr a estrutura contida em data.
620	Atribui a aco a estrutura AC_obj da requisição.
621	Procura na lista de interfaces, através de **ac_find()**, uma entrada correspondente à interface da requisição.
622	Processa o comando contido na requisição.
623-629	Se o comando for AC_GET, verifica se há entrada na lista de interfaces e se houver, retorna o conteúdo da entrada.
630-652	Se o comando for AC_SET, verifica se há uma entrada na lista. Se não houver, cria uma entrada para a interface e armazena a estrutura apontada por aco. Se já houver uma entrada, atualiza apenas as variáveis gerenciaveis pelo DCRP.
653-654	Se na estrutura o *flag* IS_AC_WAIT que controla o bloqueio de novas reservas estiver desativado, chama um *wakeup* para que os processos de reservas pendentes sejam reativados.
657-658	Retorna a prioridade do processo ao que era antes e retorna ao **ifioctl()**.

4.2.5 Manutenção das reservas

Como apresentado na seção 3.6, a implementação RSVP define uma interface com o Controle de Tráfego. A seguir é mostrado como foi implementado o controle das reservas, com base nas requisições do RSVP, utilizando a opção SIOCIFISPSCTL do **ioctl()**.

isps_ioctl()

Para facilitar a visualização, a função **isps_ioctl()** será dividida em duas partes. Na primeira (Figura 4.9), são apresentadas as entradas da função, as variáveis utilizadas e os procedimentos iniciais. Em seguida, será visto como cada comando é processado (Figura 4.10).

ac_isps.c

```
531   int isps_ioctl(fp, cmd, data, p)
532   struct socket *fp;
533   int cmd;
534   caddr_t data;
535   struct proc *p;
536   {
537   struct ispsreq *irp;
538   register struct AC_obj *acp;
539   int s;
540   int error=0;
541
542           irp = (struct ispsreq *) data;
543           acp = ac_find(irp->iq_name);
544           if (!acp)
545              return(ENXIO);
546           /* if DCRP init was requested, it lock rew resquests while timer is actived */
547           if ((acp->ac_flags & IS_AC_WAIT) && (irp->iq_fs.fs_tos != TOS_AC))
548              error = tsleep((caddr_t)&wakeup_ac_wait, PRIBIO|PCATCH, "isps_ioctl", 0);
549           s = splnet();
```

isps_ioctl()

Figura 4.9 - Início da função isps_ioctl() para chamadas ao Controle de Admissão

531-535 A função recebe como entrada: o apontador *socket* (fp); o comando que a chamou (cmd); a estrutura contendo a requisição da função; e o apontador do processo (p).

537-540 Como variáveis, o **isps_ioctl()** usa um apontador que receberá a estrutura de requisição ispsreq (irp); um apontador para a interface (arp); uma variável para controle de prioridade do processo (s); e uma variável para erros (error).

542 Atribui a irp o conteúdo da requisição.

543-545 Procura pelo nome, a interface representada na chamada. Se a interface não for encontrada, retorna uma mensagem de erro.

547-548 Se o *bitflag* IS_AC_WAIT estiver ativo, indicando que nenhuma reserva pode ser processada, a chamada é colocada em suspensão, até que o *bitflag* seja novamente desativado (Figura 4.8).

549 Muda a prioridade do processo para **splnet()**.

550 Verifica qual comando foi solicitado.

551-554 Ativa o Controle de Tráfego na interface.

555-557 Libera todas as reservas locais, com exceção das realizadas pelo DCRP.

558-573 Comandos específicos do CSZ e não implementados neste mecanismo.

574-578 Executa a reserva de recursos, retornando a identificação da reserva. Se a reserva não for aceita, retorna 0 (zero) na requisição e uma mensagem de erro.

ac_isps.c

```
550             switch (irp->iq_function) {
551             case IF_IFCTL:
552               if ((acp->ac_flags & IS_AC_ACTIVE) == 0)
553                       acp->ac_flags |= IS_AC_ACTIVE;
554               break;
555             case IF_RESET:
556               ac_reset(acp);
557               break;
558             case IF_ADDCLS:
559               /* not implemented */
560               error = ENXIO;
561               break;
562             case IF_DELCLS:
563               /* not implemented */
564               error = ENXIO;
565               break;
566             case IF_ADDSHARE:
567               /* not implemented */
568               error = ENXIO;
569               break;
570             case IF_DELSHARE:
571               /* not implemented */
572               error = ENXIO;
573               break;
574             case IF_ADDFLW:
575               irp->iq_handle = ac_addflow(irp->iq_fs, acp, irp->iq_flags);
576               if(irp->iq_handle == 0)
577                 error = ENXIO;
578               break;
579             case IF_DELFLW:
580               if (ac_delflow(acp, irp) != 0)
581                 error = ENXIO;
582               break;
583             case IF_MODFLW:
584               if (ac_modflow(acp, irp, irp->iq_flags) != 0)
585                 error = ENXIO;
586               break;
587             case IF_SETFILT:
588               if (ac_setfilt(acp, irp) != 0)
589                       /* error in setfilt */
590                 error = ENXIO;
591               break;
592             case IF_DELFILT:
593               if (ac_delfilt(acp, irp) != 0)
594                       /* error in delfilt */
595                 error = ENXIO;
596               break;
597             }
598             splx(s);
599             return(error);
600     }
```

isps_ioctl()

Figura 4.10 - Comandos da função isps_ioctl() para chamadas so Controle de Admissão

579-582 Apaga uma reserva. Se a identificação da reserva estiver incorreta, retorna erro.

583-586 Faz a alteração de um fluxo já reservado. Se a alteração não for concluída, mantém a reserva anterior e retorna um erro.

587-591 Atribui a uma reserva, o filtro correspondente.

592-596 Retira o filtro da reserva.

Solicitação de reserva

As solicitações de novas reservas são feitas através do comando IF_ADDFLOW. Através do isps_ioctl() é chamada a rotina ac_addflow(), que executará o procedimento de inclusão da reserva, como na Figura 4.11.

ac_isps.c

```
258   isps_handle_t ac_addflow(i_fs, acp, flags)
259   isps_fs_t i_fs;
260   struct AC_obj *acp;
261   short flags;
262   {
263   flow_spec          old_flow, new_flow, c_flow;
264   struct RES_table   *ac_l;
265   u_int32_t buf_len;
266
268             new_flow.res_rate = 0;
269             new_flow.res_delay = ISPS_CLASS_MAX_DELAY*1000;
270             old_flow = new_flow;
271             if (tos_test(i_fs, &new_flow) < 0)
272                       return(0);
273             c_flow = acp->ac_tot_flow;
274             if (calc_flow(acp, old_flow, new_flow, &c_flow, 0) != 0)
275                       return(0);
276             ac_l = (struct RES_table *) malloc(sizeof(res_list),M_IFADDR, M_WAITOK);
277             if (ac_l == NULL)
278                       return(0);
279             acp->ac_tot_flow = c_flow;
280             bzero((struct RES_table *)ac_l, sizeof(res_list));
281             ac_l->res_next = NULL;
282             ac_l->res_obj = acp;
283             ac_l->res_spec = i_fs;
284             ac_l->res_flow = new_flow;
285             ac_l->res_spec.fs_handle = (u_int32_t)ac_l;
286             ac_l->res_queue.qu_head = NULL;
287             ac_l->res_queue.qu_tail = NULL;
288             ac_l->res_queue.qu_len = 0;
289             ac_l->res_queue.qu_rt  = NULL;
290             ac_l->res_queue.qu_time.tv_sec = 0;
291             ac_l->res_queue.qu_time.tv_usec = 0;
292             ac_l->res_queue.qu_bytes = 0;
293             ac_l->res_flags = flags;
294             if (!acp->ac_list_head)
295                       acp->ac_list_head = ac_l;
296             else
297                       (acp->ac_list_tail)->res_next = ac_l;
298             acp->ac_list_tail = ac_l;
299             if (i_fs.fs_tos != TOS_AC) {
300                       c_flow = acp->ac_loc_flow;
301                       if (calc_flow(acp, old_flow, new_flow, &c_flow, TOS_AC) != 0)
302                                 return(0);
303                       acp->ac_loc_flow = c_flow;
304                       if (i_fs.fs_tspec.ts_tb.tbf_bkt > ac_l->res_flow.res_rate)
305                                 buf_len = i_fs.fs_tspec.ts_tb.tbf_bkt;
306                       else
307                                 buf_len = ac_l->res_flow.res_rate;
308                       ac_l->res_queue.qu_max_len = ((u_int64_t)ac_l->res_flow.res_delay *
309                                                 (u_int64_t)buf_len)/8000000;
310             }
311             return(ac_l->res_spec.fs_handle);
312   }
```

ac_addflow()

Figura 4.11- Função ac_addflow() do Controle de Admissão

258-261	O **ac_addflow()** recebe como parâmetros a descrição do fluxo i_fs, o apontador para a interface acp e os *flags*.
263-265	Como variáveis internas, usa-se: três estruturas de fluxo (old_flow, new_flow e c_flow); um apontador para a tabela de reservas (ac_l); e uma variável inteira de 32 bits (buf_len).
268-270	Inicializa as variáveis new_flow e old_flow.
271-272	Executa o mapeamento entre a Classe de Serviço solicitada e o serviço oferecido (Figura 4.2).
273	Atribui a c_flow o total de recursos reservados na rede.
274-275	Solicita o cálculo de recursos (cálculo de admissibilidade) para o novo montante de recursos da rede, baseado no novo fluxo (Figura 4.12).
276-278	Cria uma nova entrada na lista de reservas.
279	Define como novo montante de recursos da rede, o valor retornado pelo **calc_flow()**.
280-298	Inicializa a estrutura que representa a nova entrada na lista de recursos, repassando para ela os parâmetros de fluxo e incluindo-a na lista.
299-303	Se for uma solicitação local (não sendo DCRP), faz também os testes para a aceitação do fluxo local, atualizando a interface.
304-309	Calcula o *buffer* a ser usado pelo fluxo, em função da especificação do tráfego gerado e do atraso solicitado.
311	Retorna a identificação da reserva. Esta identificação é o próprio endereço de memória da reserva. Isto facilita a recuperação da informação.

Cálculo de admissibilidade

Os cálculo para a aceitação ou não de um fluxo seguem as regras definidas na Seção 4.2. Sua implementação pode ser observada na Figura 4.12.

178-182	O **calc_flow()** recebe como parâmetros: o ponteiro da interface (acp); o valor da reserva anterior (old_flow), para o caso de modificação de uma reserva; o valor da nova reserva (new_flow); o total já reservado na interface (c_flow); e a Classe de Serviço (f_tos).

ac_isps.c

```
178  int calc_flow(acp, old_flow, new_flow, c_flow, f_tos)
179  struct AC_obj      *acp;
180  flow_spec          old_flow, new_flow;
181  flow_spec          *c_flow;
182  isps_tos_t         f_tos;
183  {
184  u_int32_t new_delay;
185  struct RES_table *act = 0;
186  u_int32_t t_rate = acp->ac_tot_flow.res_rate;
187
188          c_flow->res_rate = c_flow->res_rate - old_flow.res_rate + new_flow.res_rate;
189          if (c_flow->res_rate > acp->ac_max_rate)
190                  return(-1);
191          if (new_flow.res_delay < c_flow->res_delay)
192                  c_flow->res_delay = new_flow.res_delay;
193          if ((new_flow.res_delay > old_flow.res_delay) &&
194             (old_flow.res_delay == c_flow->res_delay)) {
195                  act = acp->ac_list_head;
196                  c_flow->res_delay = ISPS_CLASS_MAX_DELAY*1000;
197                  while(act){
198                     if ((act->res_flow.res_delay < c_flow->res_delay) &&
199                        (act->res_spec.fs_tos != f_tos))
200                           c_flow->res_delay = act->res_flow.res_delay;
201                     act = act->res_next;
202                  }
203          }
204          if (t_rate < ISPS_FLOW_MIN_RATE)
205                  t_rate = ISPS_FLOW_MIN_RATE;
206          new_delay = (acp->ac_avg_delay * c_flow->res_rate) / t_rate;
207          if (new_delay > c_flow->res_delay)
208                  return(-1);
209          return(0);
210  }
```

calc_flow()

Figura 4.12 - Função calc_flow() do Controle de Admissão

184-186 São usadas como variáveis temporárias: um inteiro de 32 bits para o cálculo do novo atraso (new_delay); um ponteiro para a lista de reservas (act); e um inteiro de 32 bits para o cálculo da taxa total (t_rate).

188 Calcula a nova taxa total.

189-190 Se a nova taxa for maior que a disponível, recusa a reserva.

191-192 Se o novo atraso é menor que o menor atraso solicitado, este passa a ser o menor atraso solicitado.

193-203 Se o novo atraso for maior que o menor atraso e for uma alteração de reserva na qual este era o menor atraso solicitado, é necessário percorrer toda a lista de reservas para achar o menor atraso.

204-205 Se a taxa solicitada for menor que a menor taxa possível para reservas, ela passa a ser a menor possível.

206-208 Calcula o novo atraso médio que um pacote deve sofrer, com base no novo montante solicitado (Seção 4.2). Se o novo valor for superior ao menor atraso solicitado, recusa a reserva.

209 Retorna 0 (zero), aceitando a reserva.

Alteração de uma Reserva

A alteração de uma reserva se faz pelo comando IF_MODFLOW. Através do **isps_ioctl()** é chamada a rotina **ac_modflow()**, que fará o processamento. A figura 4.13 mostra essa rotina.

ac_isps.c

```
318  int ac_modflow(acp, irp, flags)
319  struct AC_obj *acp;
320  struct ispsreq *irp;
321  short flags;
322  {
323  flow_spec old_flow, new_flow;
324  flow_spec c_flow;
325  struct RES_table   *ac_l;
326  u_int32_t buf_len;
327
328          ac_l = (struct RES_table *)irp->iq_handle;
329          if (!ac_l)
330                  return(-1);
331          if(ac_l->res_spec.fs_handle != irp->iq_handle)
332                  return(-1);
333          old_flow = ac_l->res_flow;
334          new_flow.res_rate = 0;
335          new_flow.res_delay = ISPS_CLASS_MAX_DELAY*1000;
336          if (tos_test(irp->iq_fs, &new_flow) < 0)
337                  return(-1);
338          c_flow = acp->ac_tot_flow;
339          ac_l->res_flow = new_flow;
340          if (calc_flow(acp, old_flow, new_flow, &c_flow, 0) != 0){
341                  ac_l->res_flow = old_flow;
342                  return(-1);
343          }
344          acp->ac_tot_flow = c_flow;
345          ac_l->res_flags = flags;
346          if (irp->iq_fs.fs_tos != TOS_AC) {
347                  c_flow = acp->ac_loc_flow;
348                  if (calc_flow(acp, old_flow, new_flow, &c_flow, TOS_AC) != 0)
349                          return(-1);
350                  acp->ac_loc_flow = c_flow;
351                  if (irp->iq_fs.fs_tspec.ts_tb.tbf_bkt > ac_l->res_flow.res_rate)
352                          buf_len = irp->iq_fs.fs_tspec.ts_tb.tbf_bkt;
353                  else
354                          buf_len = ac_l->res_flow.res_rate;
355                  ac_l->res_queue.qu_max_len = ((u_int64_t)ac_l->res_flow.res_delay *
356                                                  (u_int64_t)buf_len)/8000000;
357          }
358          return(0);
359  }
```

ac_modflow()

Figura 4.13 - Função ac_modflow() do Controle de Admissão

318-321 O ac_modflow() recebe como parâmetros a descrição do fluxo i_fs, o apontador para a interface acp e os *flags*.

323-326 Como variáveis internas, são utilizadas três estruturas de fluxo (old_flow, new_flow e c_flow); um apontador para a tabela de reservas (ac_l); e uma variável inteira de 32 bits (buf_len).

328-332 Localiza e verifica a integridade da reserva, colocando-a em ac_l.

333 Atribui a old_flow o fluxo já reservado.

334-335 Inicializa a variável new_flow.

336-337 Executa o mapeamento entre a Classe de Serviço solicitada e o serviço oferecido (Figura 4.13).

338 Atribui a c_flow o total de recursos reservados na rede.

339 Atribui à reserva o novo fluxo solicitado.

340-343 Solicita o cálculo de recursos (cálculo de admissibilidade) para o novo montante de recursos da rede, baseado no novo fluxo (Figura 4.2). Se não for aceita a alteração, retorna o antigo fluxo reservado.

344 Define como novo montante de recursos da rede, o valor retornado pelo **calc_flow()**.

346-350 Se for uma solicitação local (não sendo DCRP), faz também os testes para a aceitação do fluxo local, atualizando a interface.

351-356 Calcula o *buffer* a ser usado pelo fluxo, em função da especificação do tráfego gerado e do atraso solicitado.

Eliminação de uma Reserva

A eliminação de uma reserva se faz pelo comando IF_DELFLOW. Através do **isps_ioctl()** é chamada a rotina **ac_delflow()**, que fará o processamento. A Figura 4.14 apresenta a rotina.

365-367 A função **ac_delflow()** recebe como argumentos o apontador para a interface acp e a solicitação irp.

369-373 Como variáveis internas, são utilizadas: três estruturas de fluxo (old_flow, new_flow e c_flow); dois apontadores para a lista de reservas (ac_l e act); um apontador para a lista de pacotes do Escalonador de Pacotes (scp); e um apontador para o mbuf (m)

ac_isps.c

```
365  int ac_delflow(acp, irp)
366  struct AC_obj *acp;
367  struct ispsreq *irp;
368  {
369  flow_spec old_flow, new_flow;
370  flow_spec c_flow;
371  struct RES_table   *ac_l, *act;
372  struct sc_buf *scb;
373  struct mbuf *m;
374
375          ac_l = (struct RES_table *)irp->iq_handle;
376          if (!ac_l)
377                  return(-1);
378          if(ac_l->res_spec.fs_handle != irp->iq_handle)
379                  return(-1);
380          old_flow = ac_l->res_flow;
381          new_flow.res_rate = 0;
382          new_flow.res_delay = ISPS_CLASS_MAX_DELAY*1000;
383          ac_l->res_flow = new_flow;
384          c_flow = acp->ac_tot_flow;
385          if (calc_flow(acp, old_flow, new_flow, &c_flow, 0) != 0){
386                  ac_l->res_flow = old_flow;
387                  return(-1);
388          }
389          acp->ac_tot_flow = c_flow;
390          if (ac_l->res_spec.fs_tos != TOS_AC) {
391                  c_flow = acp->ac_loc_flow;
392                  if (calc_flow(acp, old_flow, new_flow, &c_flow, TOS_AC) != 0)
393                          return(-1);
394                  acp->ac_loc_flow = c_flow;
395                  isps_del_hash(ac_l);
396          }
397          while (ac_l->res_queue.qu_head) {
398                  scb = ac_l->res_queue.qu_head;
399                  ac_l->res_queue.qu_head = scb->sc_next;
400                  m = scb->sc_m;
401                  m_freem(m);
402                  free(scb, M_IFADDR);
403          }
404          act = acp->ac_list_head;
405          while (act){
406                  if (act->res_next == ac_l)
407                          break;
408                  act = act->res_next;
409          }
410          if (act)
411                  (act)->res_next = ac_l->res_next;
412          if (ac_l == acp->ac_list_head)
413                  acp->ac_list_head = ac_l->res_next;
414          if (ac_l == acp->ac_list_tail)
415                  acp->ac_list_tail = act;
416          free(ac_l, M_IFADDR);
417          return(0);
418  }
```

ac_delflow()

Figura 4.14 - Função ac_delflow() do Controle de Admissão

375-379 Localiza e verifica a integridade da reserva, colocando-a em ac_l.

380 Atribui a old_flow o fluxo já reservado.

380-383 Inicializa a variável new_flow, colocando-a na reserva.

384	Atribui a c_flow o total de recursos reservados na rede.
385-388	Solicita o cálculo de recursos (cálculo de admissibilidade) para o novo montante de recursos da rede, baseado no novo fluxo (Figura 4.12). Se não for aceita a alteração, retorna o antigo fluxo reservado.
389	Define como novo montante de recursos da rede, o valor retornado pelo **calc_flow()**.
390-394	Se for uma solicitação local (não sendo DCRP), faz também os testes para a aceitação do fluxo local, atualizando a interface.
395	Apaga a reserva da tabela *hash* (seção 5.5).
397-403	Limpa todos os pacotes aguardando transmissão no *buffer* do Escalonador de Pacotes.
404-416	Apaga a reserva da tabela de reservas.

Atribuindo um Filtro à Reserva

A Quando um pacote chega ao Classificador de Pacotes (Capítulo 5), é necessário descobrir se este pacote pertence a um fluxo de dados com reserva ou se trata de tráfego *best-effort*. Após uma reserva ser feita, são atribuídas a esta reserva as informações que irão identificar os pacotes que receberão o tratamento especial. Esta identificação é conhecida como **filtro**. O filtro é atribuído à reserva através da função **ac_setfilt()**, representada na Figura 4.15. Maiores detalhes sobre o filtro serão mostrados no Capítulo 5.

ac_isps.c

```
424   int ac_setfilt(acp, irp)
425   struct AC_obj *acp;
426   struct ispsreq *irp;
427   {
428   struct RES_table   *ac_l;
429
430             ac_l = (struct RES_table *)irp->iq_handle;
431             if (!ac_l)
432                       return(-1);
433             if(ac_l->res_spec.fs_handle != irp->iq_handle)
434                return(-1);
435             ac_l->res_filt = irp->iq_filt;
436             irp->iq_fhdl = irp->iq_handle;
437             if (isps_add_hash(ac_l) < 0)
438                       return(-1);
439             return(0);
440   }
```

ac_setfilt()

Figura 4.15 - Função ac_setfilt() do Controle de Admissão

424-426	A função **ac_setfilt()** recebe como parâmetros o apontador para a interface acp e a solicitação irp.
428	Como variável local, é usada o apontador para a tabela de reservas ac_1.
430-434	Localiza e verifica a integridade da reserva, colocando-a em ac_1.
435	Atribui o filtro à reserva.
436	O identificador do filtro será o mesmo identificador da reserva.
437-438	Inclui o filtro na tabela *hash* (Seção 5.5).

Retirando o Filtro da Reserva

Quando não se deseja mais manter a reserva de recursos ou seja necessária a alteração do filtro associada a ela, é necessário eliminar primeiro o filtro relacionado à mesma. A Figura 4.16 mostra como a função **ac_delfilt()** implementa isso.

ac_isps.c()

```
446   int ac_delfilt(acp, irp)
447   struct AC_obj *acp;
448   struct ispsreq *irp;
449   {
450   struct RES_table   *ac_l;
451
452             ac_l = (struct RES_table *)irp->iq_handle;
453             if (!ac_l)
454                       return(-1);
455             if(ac_l->res_spec.fs_handle != irp->iq_handle)
456               return(-1);
457             if (isps_del_hash(ac_l) < 0)
458               return(-1);
459             irp->iq_fhdl = 0;
460             ac_l->res_filt.f_ipv4_destaddr.s_addr = 0;
461             ac_l->res_filt.f_ipv4_srcaddr.s_addr = 0;
462             ac_l->res_filt.f_ipv4_destport = 0;
463             ac_l->res_filt.f_ipv4_srcport = 0;
464             return(0);
465   }
```

ac_delfilt()

Figura 4.16 - Função ac_delfilt() do Controle de Admissão

446-448	A função **ac_delfilt()** recebe como parâmetros o apontador para a interface acp e a solicitação irp.
450	Como variável local, é usado o apontador para a tabela de reservas ac_1.
451-456	Localiza e verifica a integridade da reserva, colocando-a em ac_1.
437-438	Elimina o filtro na tabela *hash* (seção 5.5).

4.3 DCRP - *Distributed Control Reservation Protocol*

O DCRP é o protocolo responsável pela troca de mensagens entre as máquinas conectadas numa rede compartilhada, no nosso caso particular, uma rede Ethernet. Seu objetivo é fazer com que todas as máquinas da rede saibam o montante de recursos alocados por cada uma das demais, comunicando a cada nova reserva o novo montante reservado e verificando o atraso médio experimentado durante a transmissão de suas mensagens.

Para a implementação do mecanismo é necessária a alteração de alguns arquivos utilizados pelo Sistema Operacional e a inclusão de outros novos. A Tabela 4.1 apresenta a relação dos arquivos utilizados pelo DCRP.

Arquivo	Função
/etc/sysconfig	Configuração do sistema.
/etc/netstart	Inicialização da rede.
/etc/acd	Inicialização do Controle de Tráfego.
/var/run/acd.pid	Arquivo com a identificação do processo. É usado na inicialização para verificar se o *daemon* já está avivo.
/var/log/acd.log	Arquivo de log do sistema.
ac.h	Arquivo de *headers* do mecanismo de Controle de Tráfego
acd.h	Arquivo de *headers* do programa ac_main.c.
/sys/net/isps.h	Arquivo com as estruturas e definições da interface com o Controle de Tráfego.
ac_main.c	Programa "C" que gera o **acd**.

Tabela 4.2 - Arquivos utilizados pelo DCRP

4.3.1 Funcionamento do Protocolo

Para que o mecanismo funcione, é necessário que todas as máquinas conectadas à rede tenham conhecimento do montante de reservas confirmadas no enlace (no caso, um segmento de rede ou subrede Ethernet). Ou seja, o *host* deve saber quem está reservando o que. As informações são então armazenadas em uma tabela parecida com o exemplo da Tabela 4.3.

Durante a alocação de recursos, o **ACD** avisa às demais máquinas, que um determinado montante de recursos foi alocado na rede através de uma mensagem **AC_RESERV**. Para cada *host* conectado à rede é necessário armazenar a banda reservada e o

menor atraso solicitado. Assim, é obtida a banda total reservada na rede e o menor atraso reservado.

Host	Banda (kbps)	Atraso (ms)
150.165.2.1	512	120
150.165.2.22	1024	100
150.165.2.46	64	4096
TOTAL	1600	100

Tabela 4.3 - Exemplo dos recursos gerenciados pelo DCRP

Para operacionalizar o protocolo, cada máquina deve manter:

- uma tabela com a lista de recursos para cada interface de saída (**AC_net**);
- uma lista com as informações a respeito de cada interface (**TP_obj**);
- o total de recursos alocados naquela interface e;
- os temporizadores utilizados pelo protocolo.

A lista é inicializada com as informações de cada interface de rede e atualizada constantemente. Estas informações também serão repassadas ao Controle de Admissão.

Por motivos óbvios, nas conexões ponto-a-ponto não é necessária a troca de mensagens para o controle do meio. Como o tipo de conexão é identificado no processo de inicialização da máquina, o protocolo não é utilizado neste caso.

Uma máquina ao ser ligada à rede não sabe de início qual o montante de recursos já alocados. Além disso, uma máquina pode ser desligada sem que os recursos alocados a ela tenham sido liberados. Deve haver portanto, um mecanismo de inicialização, detecção de erros e restauração do estado correto da rede. Para isso, é utilizado um mecanismo semelhante ao *soft state* do RSVP.

O mecanismo de inicialização funciona da seguinte forma:

1. Para cada interface de rede detectada é criada uma entrada no **TP_obj**, com as informações sobre o tipo de interface, a capacidade total da banda, além de outras informações de controle.
2. Se a hora contida no temporizador **tp_next_init** for menor ou igual a hora atual, a máquina envia uma mensagem **AC_INIT**;
3. Cada máquina ao receber a mensagem **AC_INIT** em uma interface, ativa o mecanismo de "tiro aleatório" no processo de inicialização das reservas. O

objeto **AC_net** relacionado a esta interface é "zerado" e o temporizador **tp_next_init** é ativado com um valor aleatório indicando a hora da próxima inicialização. Outro temporizador aleatório **tp_next_send** é usado para definir quando será enviada a mensagem **AC_RESERV** após a inicialização; Este mecanismo é conhecido como "tiro aleatório".

4. É enviado ao *kernel* um aviso para bloquear qualquer nova reserva que seja feita por uma aplicação;
5. Após seu **tp_next_send** expirar, cada máquina (inclusive a que enviou o **AC_INIT**) envia uma mensagem *broadcast* **AC_RESERV** contendo o total de recursos que ela alocou em sua interface de saída. Isso é necessário para evitar que todas as máquinas enviem suas mensagens ao mesmo tempo, provocando congestionamento na rede (colisões, no caso de uma rede Ethernet);
6. Cada mensagem **AC_RESERV** recebida é enviada ao Controle de Admissão no *kernel* e em seguida, armazenada na tabela **AC_net**;
7. Somente após um período máximo **MAX_RES_TIME**, as máquinas desbloqueiam a solicitação de novas reservas;
8. Reporta-se novamente para o procedimento (2).

A Figura 4.17 exemplifica a troca de mensagens durante a inicialização.

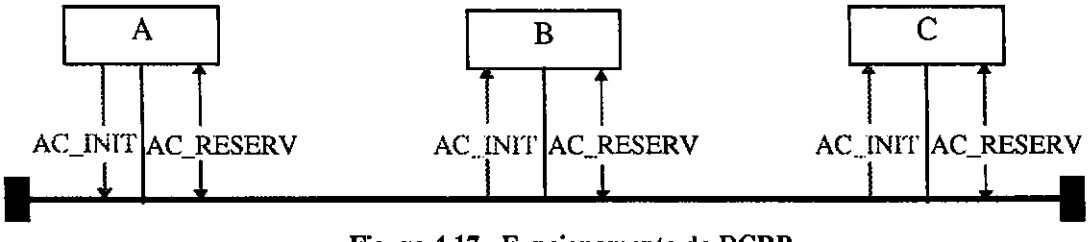


Figura 4.17 - Funcionamento do DCRP

No exemplo, a máquina "A" envia uma solicitação de inicialização que é recebida pelas maquinas "B" e "C" conectadas à mesma rede. Após o **tp_next_init** de uma das máquinas estourar, ela transmite sua reserva (a mesma que tinha anteriormente).

O **MAX_RES_TIME** é usado para evitar que um fluxo reservado antes da inicialização seja sobreposto por um novo fluxo. Isto é importante quando se está trabalhando no limite dos recursos disponíveis, dando tempo para que todas as **AC_RESERV** sejam

enviadas, reduzindo a possibilidade de que uma máquina extrapole os recursos disponíveis. Foi atribuído inicialmente, para fim de testes (não há referências sobre o assunto), um valor de 500 milissegundos. Este valor pode vir a ser alterado após a realização de testes mais abrangentes do que os realizados no Capítulo 7. Também pode-se delegar a atribuição deste valor ao gerende de rede.

Como nenhuma nova reserva é colocada na rede durante o processo de inicialização, o montante de recursos alocados antes do processo é mantido e a transmissão de pacotes de dados pode ocorrer normalmente durante este período.

O temporizador aleatório **tp_next_init** faz com que periodicamente o protocolo seja reinicializado, num processo semelhante ao *soft state* do RSVP (Capítulo 3). Seu tempo é dado aleatoriamente, com valores sugeridos entre 30 segundos e 1 minuto. Este valor foi escolhido levando-se em conta o temporizador de *soft state* do RSVP, que é de 30 segundos.

O temporizador **tp_next_send** é utilizado para reduzir a probabilidade de colisões na rede. Somente após um determinado tempo são enviadas as respostas ao pedido de inicialização. O valor aleatório sugerido para o timer pode variar entre 0 e 100 milissegundos.

4.3.2 Mensagens

Todas as mensagens DCRP são transmitidas em *broadcast*. Para transmissão das mensagens poderia ser utilizado o UDP ou o Raw-IP [WS95], ambos usados pelo RSVP. Seu uso pelo DCRP implicaria a alteração do *socket* para a aceitação de um novo protocolo e a construção de mensagens mais complexas. O UDP foi escolhido para a transmissão das mensagens DCRP pela facilidade de implementação e suporte a conexões *broadcast*, *unicast* e *multicast*. O DCRP utiliza dois tipos de mensagens:

- AC_INIT
  - ☞ Utilizada para a inicialização do protocolo
- AC_RESERV
  - ☞ Utilizada para transmitir o montante de recursos alocados no *host*.

As mensagens possuem 160 bits (20 *bytes*) e são compostas por um cabeçalho comum, dois outros campos com o montante de recursos reservados (**AC_FLOW**) e o tempo decorrido desde que a máquina recebeu a mensagem de inicialização (**AC_WAIT**). Este tempo

é expresso em microssegundos e é usado para ajustes no cálculo do atraso na rede. O formato da mensagem pode ser observado na Figura 4.18.

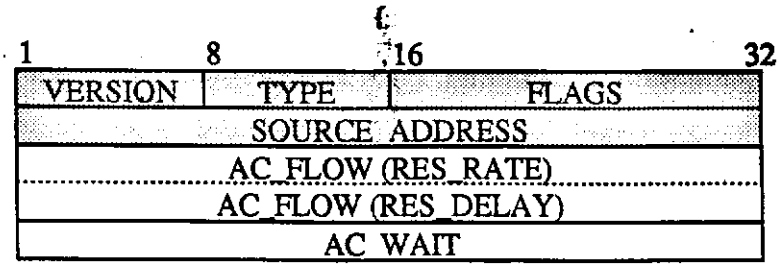


Figura 4.18 - Formato das mensagens DCRP

O campo **VERSION** indica a versão do protocolo que esta sendo utilizada. A versão atual é a 1 (um).

O campo **TYPE** indica o tipo de mensagem (AC_RESERV ou AC_INIT).

O campo **FLAGS** leva informações adicionais.

O campo **SOURCE_ADDRESS** leva o endereço da máquina que está enviando a mensagem. Isto é usado para a autenticação da mesma.

Os campos **AC_FLOW** e **AC_WAIT** somente têm importância nas mensagens AC_RESERV e são ignorados pelo protocolo nas mensagens AC_INIT.

O campo **AC_FLOW** é dividido em dois subcampos: o campo **RES_RATE** contem a taxa reservada (bps); e o campo **RES_DELAY** contém o maior atraso suportado pelo fluxo (Figura 4.19). Ambos os campos são inteiros de 32 bits.

ac.h

```
40    typedef struct {
41            u_int32_t       res_rate; /* Rate */
42            u_int32_t       res_delay;      /* Delay */
43    } flow_spec;
```

Figura 4.19 - Estrutura flow_spec usada pelo DCRP e pelo Controle de Admissão

As estruturas da mensagem são definidas no arquivo **acd.h** e apresentada na Figura 4.20.

124 - 129 Cabeçalho da mensagem, com os campos version, type, flags e source, correspondendo aos campos **VERSION**, **TYPE**, **FLAGS** e **SOURCE_ADDRESS**, respectivamente.

134-142 Estrutura completa da mensagem, com os campos ac_hdr, ac_flow ac_wait. Isso corresponde ao cabeçalho da mensagem, ao campo **AC_FLOW** e ao campo **AC_WAIT**, respectivamente. São colocados também *defines* para os campos do cabeçalho.

acd.h

```
124   typedef struct {
125           char    version;  /* DCRP version */
126           char    type;             /* message type */
127           short   flags;            /* flags */
128           struct in_addr source;    /* message source address */
129   } ac_header;
                          Comentários
134   typedef struct s_message{
135           ac_header ac_hdr;  /* common header */
136           flow_spec         ac_flow; /* reserved flow */
137           u_int32_t         ac_wait; /* wait time since last init */
138   #define acm_version       ac_hdr.version
139   #define acm_type  ac_hdr.type
140   #define acm_flags ac_hdr.flags
141   #define acm_source        ac_hdr.source
142   } ac_message;
```

Figura 4.20 - Estruturas da mensagem DCRP

O DCRP é implementado no programa **ACD** *(Admission Control Daemon)*. Este programa é inicializado tão logo as interfaces de rede tenham sido configuradas e antes que qualquer pacote seja enviado. Para a ativação automática é necessário a alteração de dois arquivos de configuração do Sistema Operacional: o **sysconfig**; e o **netstart**.

No **sysconfig** foi incluída a opção "acd" (Figura 4.21). Ela deve receber o valor "*YES*" se desejarmos utilizar controle de tráfego na máquina ou o valor "*NO*" caso contrário.

sysconfig

```
191   # Set to YES to turn on Traffic Control Protocol. NOTE: It needs ISPS routines
192   # in the kernel.
193   acd=YES
```

Figura 4.21 - Alterações no sysconfig para executar o ACD

No arquivo netstart (Figura 4.22), é colocada a chamada para o programa **acd**, de acordo com o valor atribuído à opção "acd":

netstart

```
46    # Start Trafic Control Protocol
47    if [ "X${acd}" = X"YES" ]; then
48            echo "
49            echo 'Starting Trafic Control Protocol - acd.'
50            /etc/acd -d -l&
51    fi
```

Figura 4.22 - Alterações no netstart para executar o ACD

A Tabela 4.4 apresenta as opções de inicialização do programa **acd**.

Opção	Função
-d	Ativa o modo *debug*. A configuração inicial e todas as mensagens de controles são ecoadas na tela.
-l	As mensagens do sistema são armazenadas no arquivo **acd.log**. Toda vez que o arquivo atingir 1Mbyte, é reinicializado.

Tabela 4.4 - Opções da linha de comando para o ACD

4.3.3 Implementação do DCRP

A implementação do DCRP pelo **ACD** segue o diagrama representado na Figura 4.23. O código de implementação do DCRP está contido no arquivo **ac_main.c** e será apresentado a seguir, na forma de tópicos.

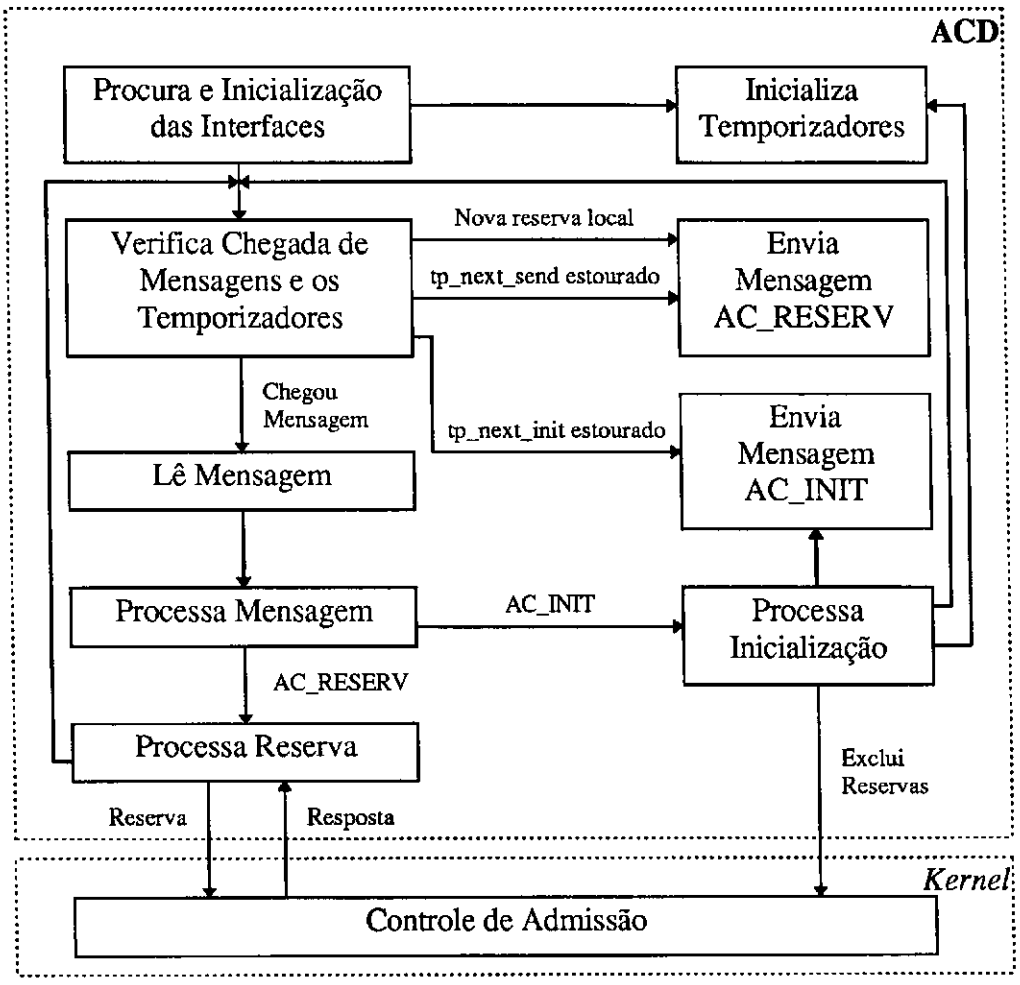


Figura 4.23 - Diagrama de funcionamento do ACD

main()

Uma vez configurado e ativo, o **acd** inicia seu processamento. A Figura 4.24 mostra a implementação da rotina principal do programa.

ac_main.c

```
72    main(argc,argv)
73    int    argc;
74    char  *argv[];
75    {
76    int    i;
77    int on = 1;
78    struct in_addr  addr;
79    int    sin_len;
80    struct sockaddr_in sin;
81    int pid;
82
83          if ((pid=test_acd()) != 0) {
84              printf("acd is already running. PID = %i\n", pid);
85              exit(-1);
86          }
87          pid = getpid();
88          if (save_pid(pid) != 0)
89          exit(-1);
90          if (argc > 1)
91          for (i=1;i<argc;i++) {
92              if(!strcmp(argv[i],"-d")) {
93                 debug=1;
94              }
95              if(!strcmp(argv[i],"-l")) {
96                 debug = 1;
97                 open_log();
98              }
99          }
100         if ((tp_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
101         fprintf(stdout,"ERROR: Socket\n");
102            exit(1);
103         }
104         if (setsockopt(tp_socket, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on)) < 0) {
105            fprintf(stdout,"ERROR: setsockopt SO_BROADCAST\n");
106            exit(1);
107         }
108         memset(&sin, 0, sizeof(sin));
109         sin_len = sizeof(sin);
110         sin.sin_family = AF_INET;
111         sin.sin_port = htons(TP_PORT);
112         sin.sin_addr.s_addr = INADDR_ANY;
113         if (bind(tp_socket, (struct sockaddr *)&sin, sizeof(sin))) {
114             fprintf(stdout,"ERROR: Bind\n");
115             exit(1);
116          }
117         if (getsockname(tp_socket, (struct sockaddr *)&sin, &sin_len) == -1)
118            fprintf(stdout,"ERROR: Bad address for socket\n");
119         srand(pid);
120         init_interfaces();
121         ac_exec(sin);
122         close(tp_socket);
123   }
```

main()

Figura 4.24 - Função principal do ACD

83-89 Testa a existência de outro processo ativo. Se o **acd** não estiver rodando, é gravado o número do processo no arquivo **acd.pid**. Caso contrário, o programa é abortado.

90-99 Testa as opções de configuração do *daemon*.

100-107 Abre e configura o descritor do *socket*, que será utilizado para o envio e recepção das mensagens pelo protocolo. A troca de mensagens é feita através do protocolo UDP, por *broadcast*, conforme a máscara de rede utilizada. A opção SO_BROADCAST é ativada para que se possa enviar e receber mensagens em *broadcast*.

108-118 Atribui à estrutura apontada por sin o valor da porta utilizada pelo protocolo (TP_PORT). O endereço INADDR_ANY é utilizado para a recepção das mensagens em todas as interfaces ativas.

119 O número do processo serve como semente para a geração dos números aleatórios utilizados pelos temporizadores.

120 Configura as interfaces e ativa o módulo de Controle de Admissão no *kernel*.

121 Chama a rotina de manutenção do protocolo. Ela somente será interrompida em caso de erros graves.

122 Fecha o *socket* e encerra o programa.

init_interfaces()

A função init_interfaces é a responsável pela descoberta das interfaces de rede existentes na máquina, sua condição e taxas de transmissão. Também cabe a ela repassar estas informações ao *kernel* para serem usadas pelo Controle de Admissão.

A análise desta rotina será dividida em quatro partes. Primeiro analisaremos o corpo da função (Figura 4.25). Depois, veremos a estrutura TP_obj, que armazenará as informações da interface (Figura 4.26). Também veremos a estrutura que comporta a lista de reservas (Figura 4.27). E, finalmente, a atribuição de valores à esta estrutura (Figura 4.28).

418 - 427 Define as variáveis utilizadas durante a rotina. O init_socket será usado para a localização das interfaces. A estrutura apontada por obj armazena as informações sobre as interfaces. O *buffer* ifbuf recebe do *kernel* as informações lá armazenadas pelo ifconfig [WS95], sobre as interfaces. Também é usado para este fim as estruturas ifc, sdl, ifptr, ifend e a variável ifflags. A variável interface

guarda o índice da interface. A variável request é usada para procura de endereços.

ac_main.c

```
416  init_interfaces()
417  {
418  int        init_socket;
419  struct     TP_obj   *obj;
420  char       ifbuf[MAX_INTERFACES * sizeof(struct ifreq)];
421  struct     ifconf    ifc;
422  int        ifflags;
423  register struct init_cmd *init_p;
424  struct     sockaddr_dl        *sdl;
425  struct ifreq          *ifptr,    *ifend;
426  int        interface;
427  u_long     request;
428
429             TAIL_LIST = HEAD_LIST = NULL;
430             if ((init_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
431               fprintf(fp,"ERROR: socket\n");
432               return(-1);
433             }
434             ifc.ifc_buf=ifbuf;
435             ifc.ifc_len=sizeof(ifbuf);
436             if (ioctl(init_socket, SIOCGIFCONF, (char *) &ifc) <0) {
437               fprintf(fp,"ERROR: SIOCGIFCONF\n");
438               return(-1);
439             }
440             ifend = (struct ifreq *) (ifc.ifc_buf + ifc.ifc_len);
441             ifptr = ifc.ifc_req;
442             ifflags = ifc.ifc_req->ifr_flags;
443             close(init_socket);
444             while (ifptr < ifend) {
445               if (ifptr->ifr_addr.sa_family == AF_LINK) {
446                       sdl = (struct sockaddr_dl *) &ifptr->ifr_addr;
447                       for(init_p = init_cmds; init_p->type; init_p++)
448                         if (sdl->sdl_type == init_p->type){
```

Figura 4.28

```
493                       }
494               }
495               if (ifptr->ifr_addr.sa_len)
496                       ifptr = (struct ifreq *) ((caddr_t) ifptr +
497                               ifptr->ifr_addr.sa_len - sizeof(struct sockaddr));
498               ifptr++;
499             }
500             if (debug)
501               show_config(HEAD_LIST);
502  }
```

init_interfaces()

Figura 4.25 - Visão geral da função init_interfaces() do ACD

429 Inicializa a lista de interfaces. HEAD_LIST aponta para o início da lista e TAIL_LIST aponta para o final da lista.

430-433 Inicializa o *socket* init_socket que será usado para buscar a lista de interfaces configuradas no *kernel* pelo ifconfig [WS95].

434-435 Inicializa a estrutura ifc que irá solicitar ao *kernel* as interfaces.

436-439 Busca no *kernel* através do **ioctl()**, a configuração de todas as interfaces de rede existentes, armazenando-as em ifc.

440-442 ifend aponta para o final da lista de interfaces. ifptr aponta para o início da lista e ifflags armazena os *flags* da solicitação.

443 Fecha o *socket* que buscou as configurações.

444 Executa busca pelas interfaces até que chegue ao fim da estrutura de dados.

445 Verifica se o endereço configurado para a interface pertence a família AF_LINK.

446 Faz a conversão do endereço para a estrutura de endereços apontada por sdl.

447-493 Procura na tabela init_cmds (Figura 4.27), uma entrada correspondente ao tipo de interface encontrada, configurando a nova interface como será visto na Figura 4.28.

495-499 Atribui a ifptr o endereço da próxima ocorrência no *buffer* de dados. Se for o final do *buffer*, encerra a configuração.

500-501 Se foi solicitada a opção *debug* ao ativar o **acd**, é apresentada a configuração das interfaces.

A estrutura que comportará as informações sobre a interface e os dados do protocolo está contida no arquivo **acd.h** e faz menção à estrutura **AC_obj**, armazenada no *kernel*.

188 Aponta para a próxima interface de rede.

189 Aponta para uma cópia da estrutura AC_obj, armazenada no *kernel* e utilizada pelo Controle de Admissão. Diversas entradas comuns entre as duas estruturas são resumidas na forma de *defines* para o TP_obj (Figura 4.26).

190-191 Aponta para o nome e o índice da interface.

192 Tipo da interface.

193 *Flags* da estrutura.

194 Tipo de conexão usada na rede (ponto-a-ponto, compartilhada ou *switch*).

195-197 Endereços de *broadcast* da rede, endereço IP da interface e máscara da rede.

198 Maior taxa de transmissão possível na rede.

199-206 Informações sobre o tráfego *best-effort*. Estas informações são usadas para o Escalonamento de Pacotes do tipo Datagrama (Seção 6.2).

acd.h

```
187  typedef struct TP_obj {
188          struct TP_obj     *tp_next_interface;      /* next occurrence */
189          struct AC_obj     tp_ac_obj;               /* AC_obj pointer */
190  #define tp_name                tp_ac_obj.ac_name
191  #define tp_interface           tp_ac_obj.ac_interface
192          u_char            tp_type;          /* interface type */
193          u_char            tp_flags;         /* interface status */
194          u_char            tp_connection;           /* connection type */
195          struct in_addr    tp_to_addr;              /* broadcast address */
196          struct in_addr    tp_local_addr;           /* local address */
197          struct in_addr    tp_netmask;              /* netmask */
198  #define tp_max_rate            tp_ac_obj.ac_max_rate
199  #define tp_datagram_len        tp_ac_obj.ac_datagram.dt_len
200  #define tp_datagram_bytes      tp_ac_obj.ac_datagram.dt_bytes
201  #define tp_datagram_min_len    p_ac_obj.ac_datagram.dt_min_len
202  #define tp_datagram_max_len    tp_ac_obj.ac_datagram.dt_max_len
203  #define tp_datagram_time tp_ac_obj.ac_datagram.dt_time
204  #define tp_datagram_rate       tp_ac_obj.ac_datagram.dt_rate
205  #define tp_datagram_min_rate   tp_ac_obj.ac_datagram.dt_min_rate
206  #define tp_datagram_max_buf    tp_ac_obj.ac_datagram.dt_max_buf
207          struct timeval    tp_last_init;     /* last init time */
208          struct timeval    tp_next_init;     /* next init time */
209          struct timeval    tp_next_send;     /* next send time */
210          char              tp_who_init;      /* who sent INIT */
211          int               tp_machine_on;    /* network attached hosts account */
212          u_int32_t         tp_res_act;       /* received RESERV account */
213  #define tp_avg_delay           tp_ac_obj.ac_avg_delay
214          u_int32_t         tp_delay_table[MAX_TIME_ACCT]; /* delay table */
215          u_int32_t         tp_acct_delay_number;    /* delay account */
216          int               tp_acct_delay_point;     /* delay table pointer */
217  #define tp_tot_flow            tp_ac_obj.ac_tot_flow
218  #define tp_loc_flow            tp_ac_obj.ac_loc_flow
219          struct    AC_net  *tp_list_head;    /* reserves list head */
220          struct    AC_net  *tp_list_tail;    /* reserves list tail */
221  } tp_obj;
```

Figura 4.26 - Estrutura TP_obj do ACD

207-209 Estruturas de tempo, armazenando a última inicialização (tp_last_init), a próxima inicialização (tp_next_init) e quando enviar a resposta ao AC_INIT (tp_next_send).

211-212 O tp_machine_on contabiliza o número de máquinas ligadas à rede e o tp_res_act a quantidade de mensagens de reserva de recursos recebidas pela interface desde a última inicialização.

213 Guarda o atraso médio experimentado na rede.

214 Tabela com os *probes* feitos, a fim de calcular o atraso médio na rede.

215 Contabiliza o número de *probes* recebidos desde que o processo foi ativado.

216 Como a tabela de *probes* tem um tamanho máximo, esta variável é usada para apontar o ponto na tabela a ser incluído ou substituído, quando estiver cheia.

217-218 Estruturas contendo o montante de recursos reservados na rede e localmente.

219-220 Apontadores para a lista de reservas feitas pelas máquinas na rede. Estes apontadores indicam o início e o fim da lista. Estas estruturas estão detalhadas na Figura 4.27.

A lista de reservas é construída a partir da estrutura AC_net, apresentada na Figura 4.27.

acd.h

```
176  struct AC_net {
177          struct AC_net    *net_next;     /* next occurrence */
178          struct in_addr   net_addr;      /* source address */
179          isps_handle_t    net_handle;    /* isps handle */
180          flow_spec        net_flow;      /* reserved flow */
181  };
```

Figura 4.27 - Estrutura AC_net do ACD

177 Aponta para a próxima reserva da lista.

178 Endereço da máquina que enviou a reserva.

179 Identificação da reserva no *kernel*.

180 Fluxo reservado pela máquina.

A localização das informações sobre a interface e sua atribuição para a inserção na lista de interfaces é apresentada na Figura 4.28.

449-450 Aloca espaço de memória para obj, "zerando" todos os seus campos.

451-452 Copia o nome da interface e seu número para obj.

453-461 Verifica o tipo de interface. Se for Ethernet, atribui o valor ACC_SHARE ao tipo. Se for um enlace ponto-a-ponto, atribui o valor ACC_PPP. Isso será usado para saber se o DCRP deve ou não ser executado na interface.

462-466 Verifica se a interface está ou não ativa.

467-476 Se a interface for do tipo ACC_PPP, busca no *kernel* a taxa de transmissão configurada; calcula a taxa mínima para reserva do tráfego de datagramas como sendo de 10% do total possível (Seção 4.1), armazenando-o em init_cmds; e busca o endereço do outro ponto do enlace. Se for do tipo ACC_SHARE, pega o endereço *broadcast*.

477-478 Busca no *kernel* o endereço IP da interface e a máscara de rede.

479-481 Atribui a obj a taxa máxima da rede, o valor mínimo para transmissão *best-effort* e a taxa inicial deste tipo de pacote.

ac_main.c

```
449                     obj = (struct TP_obj *)malloc(sizeof(struct TP_obj));
450                     bzero((char *)&(*obj),sizeof(struct TP_obj));
451                     strcpy(obj->tp_name, ifptr->ifr_name);
452                     obj->tp_interface = sdl->sdl_index;
453                     switch(sdl->sdl_type){
454                       case IFT_PPP:
455                       case IFT_SLIP:
456                               obj->tp_type = ACC_PPP;
457                               break;
458                       case IFT_ETHER:
459                               obj->tp_type = ACC_SHARE;
460                               break;
461                     }
462                     if (if_up(ifptr->ifr_name)) {
463                       obj->tp_flags |= IS_IF_UP;
464                       obj->tp_ac_obj.ac_flags |= IS_AC_ACTIVE;
465                     } else
466                       obj->tp_flags |= IS_IF_DOWN;
467                     if (obj->tp_type == ACC_PPP) {
468                       if((init_p->max_rate = get_baudrate(obj->tp_interface)) < 0){
469                               fprintf(fp,"ERRO no baudrate");
470                               return(-1);
471                       }
472                       init_p->datagram_rate = init_p->max_rate/10;
473                       request= SIOCGIFDSTADDR;
474                     } else
475                       request= SIOCGIFBRDADDR;
476                     obj->tp_to_addr = get_addr(request, *ifptr);
477                     obj->tp_local_addr = get_addr(SIOCGIFADDR, *ifptr);
478                     obj->tp_netmask = get_addr(SIOCGIFNETMASK, *ifptr);
479                     obj->tp_max_rate = init_p->max_rate;
480                     obj->tp_datagram_min_rate = init_p->datagram_rate;
481                     obj->tp_datagram_rate = obj->tp_datagram_min_rate;
482                     obj->tp_datagram_max_len = (obj->tp_datagram_rate)/8;
483                     obj->tp_datagram_max_buf = (((u_int64_t)obj->tp_datagram_rate/8)
484                                                     * ISPS_CLASS_MAX_DELAY) / 1000;
485                     gettimeofday(&obj->tp_datagram_time,0);
486                     obj->tp_avg_delay = init_p->hardware_delay;
487                     obj->tp_tot_flow.res_rate = obj->tp_datagram_rate;
488                     obj->tp_tot_flow.res_delay = ISPS_CLASS_MAX_DELAY*1000;
489                     obj->tp_loc_flow = obj->tp_tot_flow;
490                     obj->tp_machine_on = 1;
491                     store_interface(obj, &TAIL_LIST, &HEAD_LIST);
492                     break;
```

init_interfaces()

Figura 4.28 - Continuação da função inti_interfaces() do ACD

482 Estipula o tamanho máximo do *buffer*, de acordo com a fórmula definida na Seção 4.1.

483-484 Calcula o tamanho máximo do *buffer* (em *bytes*), que receberá os pacotes *best-effort* para obter um atraso máximo de ISPS_CLASS_MAX_DELAY, definido pelo RSVP (figura 3.XXX). Se o *buffer* estourar, os pacotes que chegarem e não encontrarem espaço, serão descartados.

485 Atribui o valor da hora atual ao *timer* para datagramas. Esta variável é usada no Escalonamento de Pacotes.

486 Como ainda não foi feita nenhuma coleta de tempo na rede, é atribuído ao atraso médio o menor atraso possível na interface.

487 - 489 Inicialmente, o único fluxo reservado é para o tráfego *best-effort*. Este valor então é atribuído ao fluxo reservado localmente e ao fluxo total reservado na interface. O atraso esperado neste caso é definido por ISPS_CLASS_MAX_DELAY.

490 Atribui o valor 1 (um) ao número de máquinas conectadas à rede. Esta variável tem finalidade apenas estatística. Mas, pode vir a ser usada no futuro para a previsão do atraso na rede.

491 Constrói a lista de interfaces e envia para o Controle de Admissão no *kernel*, a configuração da interface.

492 Como foi localizada a interface na tabela init_cmds, encerra a busca e retorna o processamento, conforme a Figura 4.25.

init_cmds

A tabela init_cmds armazena de forma estática as informações sobre os tipos de interfaces, conforme a Figura 4.29. Nas próximas versões, isto pode ser alterado para ser carregado a partir de um arquivo de configuração. Optou-se por enquanto, manter estes valores constantes. Estes valores estão definidos no arquivo **acd.h** e pode ser alterado pelo gerente antes da compilação do programa.

acd.h

```
90   struct init_cmd {
91       u_char        type;          /* interface type */
92       u_int32_t     max_rate;      /* interface maximum rate */
93       u_int32_t     datagram_rate; /* datagram minimum rate */
94       u_int32_t     hardware_delay; /* minimum hardware delay */
95   };
96
97   /*
98    * Interfaces information table
99    */
100  struct init_cmd init_cmds[] = {
101          { IFT_ETHER, ETHER_RATE, ETHER_DATAGRAM, ETHER_DELAY },
102          { IFT_PPP,      0,       0,       0 },
103          { IFT_SLIP,     0,       0,       0 },
104  };
```

Figura 4.29 - Estrutura de identificação e informações sobre as interfaces usada pelo ACD

90-95 Estrutura de dados init_cmd, contendo: o tipo de interface; a taxa máxima suportada; a taxa inicial que será usada para a transmissão de pacotes em *best-effort*; e o atraso mínimo esperado durante a transmissão de um pacote.

100-104 Na variável init_cmds são colocados os valores (constantes), a respeito de cada interface.

ac_exec()

A função **ac_exec()** é o núcleo do DCRP. Através dela são verificadas as recepções de mensagens pelas interfaces, verificados os recursos alocados no *kernel* e mantidos os temporizadores do protocolo.

A análise desta rotina será dividida em duas partes. Primeiro, será visto brevemente as variáveis utilizadas na rotina e em seguida a própria rotina. As variáveis podem ser vistas na Figura 4.30.

ac_main.c

```
523  ac_exec()
524  {
525  fd_set fds;
526  int nb, fdval;
527  struct timeval fd_time ;
528  struct timeval now;
529  double delta;
530  struct TP_obj *start;
```

ac_exec()

Figura 4.30 - Início da função ac_exec() do ACD

526-528 As variáveis fds, nb, fdval e fd_time são usadas pelo *File Descriptor* (fd) para o controle das chamadas com o *socket*. Toda vez que chega uma nova mensagem pela rede, o sistema é avisado.

529 O now armazena a hora atual do sistema, em forma de segundos e microssegundos.

530 A variável de ponto flutuante delta é usadas no cálculo dos temporizadores do protocolo.

531 A variável start aponta para a lista de interfaces.

O procedimento utiliza um laço for sem fim para manter o processo ativo até que algum erro ocorra no processo. Os passos a seguir estão ilustrados na Figura 4.31.

ac_main.c

```
533             for(;;){
534               FD_ZERO(&fds);
535               FD_SET(tp_socket, &fds);
536               fd_time.tv_sec = 0;
537               fd_time.tv_usec = 1000;
538               nb = select(FD_SETSIZE, &fds, (fd_set *) 0, fd_set *) 0, &fd_time);
539               if (nb < 0) {
540                       fprintf(fp,"ERROR: Select\n");
541                       exit(1);
542               }
543               gettimeofday(&now,0);
544               delta=now.tv_sec + (double)now.tv_usec/1000000;
545               if(FD_ISSET(tp_socket, &fds)) /* New message arrived */
546                       receive_ac();
547               for (start = HEAD_LIST; start; start=start->tp_next_interface) {
548                       if ((start->tp_flags & IS_IF_UP) &&
549                         (start->tp_type == ACC_SHARE)){
550                          verify_resources(start);
551                          if (start->tp_next_send.tv_sec > 0){
552                                  if (delta >= (start->tp_next_send.tv_sec +
553                                    (double)start->tp_next_send.tv_usec/1000000)){
554                                     /* next_sent timer expired */
555                                     send_reserv(start);
556                                  }
557                          }
558                          if (start->tp_ac_obj.ac_flags & IS_AC_WAIT) {
559                                  u_int32_t w_delta;
560                                  w_delta = (now.tv_sec - start->tp_last_init.tv_sec) * 1000000
561                                          +(now.tv_sec - start->tp_last_init.tv_sec);
562                                  if (w_delta >= MAX_RES_TIME){
563                                     /* Unlock new reserves in kernel */
564                                     start->tp_ac_obj.ac_flags &= ~IS_AC_WAIT;
565                                     if (ac_kern(start, IF_ACSET) < 0){
566                                             fprintf(fp,"ERROR: AC_kern\n");
567                                             return(-1);
568                                     }
569                                  }
570                          }
571                          if (delta >= (start->tp_next_init.tv_sec +
572                            (double)start->tp_next_init.tv_usec/1000000))
573                                  /* Init timer expired */
574                                  send_init(start);
575                       }
576                 }
577             }
578     }
```

ac_exec()

Figura 4.31 - Continuação da função ac_exec() do ACD

534 Zera a variável fds, utilizada para busca de chamadas nos descritores de arquivos.

535 Atribui a fds o valor tp_socket, utilizado para a transmissão e recepção das mensagens pelo **acd**.

536-537 Atribui um tempo de 0,001 (um milésimo) segundos ao descritor de arquivos, para a revisão de suas chamadas.

538-542 Seleciona a busca de chamadas, verificando os descritores de arquivos. Se algum erro ocorrer durante esta operação, o procedimento é interrompido.

543-544 Obtém a hora do sistema, armazenando-a em now e converte em segundos para a variável delta.

545-546 Verifica se há uma chamada no descritor tp_socket. Se houver, significa que chegou uma nova mensagem para o protocolo. Então, é chamada a rotina que irá ler a mensagem e envia-la para processamento.

547-549 Faz a varredura entre as interfaces para verificar se houve alteração nos temporizadores ou alguma nova solicitação de reserva no Controle de Admissão. Isso será feito apenas nas interfaces ativas e que não são ponto-a-ponto.

550 Verifica se houve alteração nos recursos reservados no *kernel*, comparando os totais reservados. Se houver alteração, uma nova mensagem e reserva é enviada à rede para a atualização do protocolo nas demais máquinas.

551-557 Se houve inicialização ou reinicialização do protocolo, a variável tp_next_send recebe a hora que deve enviar o montante reservado localmente no *kernel*. Neste caso, delta é comparado com o valor na variável, para saber se é o momento da transmissão. Se for o momento, envia a mensagem com a reserva.

558-570 Também, se houve inicialização ou reinicialização do protocolo, é ativado o *bit-flag* IS_AC_WAIT no *kernel*, para bloquear as novas reservas até que se passem MAX_RES_TIME microssegundos, desde a última inicialização. Quando isso ocorre, o *bit-flag* é desativado no *kernel*, desbloqueando as reservas.

571-574 Verifica se o temporizador de inicialização do protocolo expirou. Se estiver expirado, chama o processo de inicialização do protocolo.

receive_ac()

Após receber uma chamada no descritor de arquivo tp_socket, o **ac_exec()** chama a rotina **receive_ac()** para buscar a mensagem e testa-la. A Figura 4.32 mostra a rotina.

ac_main.c

```
584   receive_ac()
585   {
586   int cc;
587   struct sockaddr_in from;
588   struct s_message msg;
589   int f_len = sizeof(from);
590   struct timeval now;
591
592           gettimeofday(&now,0);
593           cc = recvfrom(tp_socket,(struct s_message *)&msg, sizeof(msg), 0,
594                   (struct sockaddr *)&from, &f_len);
595           if (cc <= 0)
596             fprintf(fp,"ERROR: recvfrom\n");
597           else
598             if (from.sin_addr.s_addr != msg.acm_source.s_addr)
599                   fprintf(fp,"Source address error\n");
600             else
601                   msg_process(&msg, now);
602   }
```

receive_ac()

Figura 4.32 - Função receive_ac() do ACD

586-590 A variável cc é usada para armazenar o número de *bytes* recebidos. O from corresponde ao endereço da origem da mensagem. A estrutura msg é a própria mensagem. A variável f_len armazena o tamanho do endereço e é usada pelo comando recv from no recebimento da mensagem. Finalmente, a estrutura de tempo now guarda a hora do sistema.

592 Obtém a hora do sistema e armazena em now.

593-594 Recebe a mensagem.

595-600 Verifica a autenticidade da mensagem. Se o endereço contido na mensagem não corresponder ao de quem a enviou, ela é descartada.

601 Se for uma mensagem válida, chama a rotina **msg_process()** para processa-la.

msg_process()

A rotina msg_process faz a verificação do tipo de mensagem recebida e envia para a rotina correspondente. Estas rotinas são o **init_process()** (AC_INIT) e o **resv_process()** (AC_RESERV). Seu código está representado na Figura 4.33.

694-696 A rotina tem como entrada a mensagem (msg) e a estrutura de tempo com a hora de recepção da mensagem (now).

698-699 A estrutura start aponta para a lista de interfaces e a variável de endereço net_addr será usada para a localização da interface.

702-707 Procura na lista de interfaces uma que tenha endereço de rede correspondente ao endereço da máquina que enviou a mensagem.

ac_main.c

```
694    msg_process(msg, now)
695    struct s_message *msg;
696    struct timeval now;
697    {
698    struct TP_obj *start = HEAD_LIST;
699    struct in_addr net_addr;
700
701            /* find interface from message address */
702            while(start){
703              net_addr.s_addr = msg->acm_source.s_addr & start->tp_netmask.s_addr;
704              if (net_addr.s_addr == start->tp_to_addr.s_addr)
705                      break;
706              start=start->tp_next_interface;
707            }
708            if (start == NULL) {
709              fprintf(fp,"Source address error\n");
710              return;
711            }
712            /* Verify local address */
713            if (msg->acm_source.s_addr != start->tp_local_addr.s_addr){
714              if (debug)
715                      fprintf(fp,"\nReceived %s message from %s\n",
716                              T_message[msg->acm_type], inet_ntoa(msg->acm_source.s_addr));
717              switch(msg->acm_type){
718                      case AC_INI:
719                        init_process(start, now, REMOTE_INIT);
720                        break;
721                      case AC_RESERV:
722                        if (msg->acm_source.s_addr != start->tp_local_addr.s_addr)
723                                resv_process(start, msg, now);
724                        break;
725              }
726            }
727    }
```

msg_process()

Figura 4.33 - Função msg_process() do ACD

708-711 Se não achou a interface, avisa que houve um erro no endereço da mensagem e retorna ao **ac_exec()**.

713 Como as mensagens enviadas em *broadcast* são repassadas de volta através da interface *loopback*, deve-se testar se a mensagem que chegou não é a mesma que acabou de ser transmitida.

714-716 Avisa (se as opções **-p** ou **-l** forem usadas no **acd**), que tipo de mensagem chegou e quem a transmitiu.

717-725 Testa o tipo de mensagem. Se for AC_INIT, chama a rotina **init_process()**, repassando a interface, a hora e uma constante indicando que se trata de inicialização remota (REMOTE_INIT). Se for AC_RESERV, chama a rotina **resv_process()**, repassando a interface, a mensagem e a hora.

init_process()

A rotina **init_process()** (Figura 4.34) irá executar todos os procedimentos de inicialização do protocolo, liberando todas as reservas remotas e restaurando o estado da rede.

ac_main.c

```
754    init_process(t_obj, now, w_init)
755    struct TP_obj *t_obj;
756    struct timeval now;
757    char       w_init;
758    {
759    struct AC_net       *acrn;
760    struct ispsreq ir;
761
762            /* Lock new reserves in kernel */
763            t_obj->tp_ac_obj.ac_flags |= IS_AC_WAIT;
764            if (ac_kern(t_obj, IF_ACSET) < 0){
765               fprintf(fp,"ERROR: init_process AC_kern\n");
766                        return(-1);
767            }
768            sprintf(ir.iq_name,"%s",t_obj->tp_name);
769            ir._iq_iqu._iq_fs.fs_tos = TOS_AC;
770            ir.iq_flags = NULL;
771            ir.iq_fhdl = NULL;
772            acrn = t_obj->tp_list_head;
773            while(acrn){
774               if (acrn->net_handle) {
775                        ir.iq_function = IF_DELFLW;
776                        ir.iq_handle = acrn->net_handle;
777                        if (kern_ctl(ir) <= 0){
778                           fprintf(fp,"Error: Handle assign\n");
779                           return(-1);
780                        }
781               }
782               t_obj->tp_list_head = acrn->net_next;
783               free(acrn);
784               acrn = t_obj->tp_list_head;
785            }
786            set_timer(t_obj, now);
787            set_d_timer(t_obj);
788            t_obj->tp_machine_on = 1;
789            t_obj->tp_who_init = w_init;
790    }
```

init_process()

Figura 4.34 - Função init_process() do ACD

754 -757 A rotina recebe como parâmetro a interface gerenciada, a hora da inicialização e quem solicitou a inicialização.

759-760 Como variáveis internas são utilizadas as estruturas acrn e ir. A estrutura acrn irá apontar para a lista de reservas do protocolo. Já a estrutura ir é usada para a solicitação de reserva ao Controle de Admissão no *kernel*.

763-767 Ativa o *bit-flag* IS_AC_WAIT, para bloquear as novas reservas no Controle de Admissão e envia a informação para o *kernel*.

769-771 Inicia a construção da estrutura ir com o nome da interface, *flags* e *handle* do filtro usado pela reserva (usado pelo RSVP).

772 Atribui a acm o início da fila de reservas do protocolo.

773-785 Percorre toda a lista de reservas, solicitando a liberação do fluxo no *kernel* e liberando a entrada na fila. Isso irá limpar todas as reservas feitas pelo protocolo, restaurando-o ao estado original de inicialização. Apenas as reservas solicitadas localmente pelas aplicações ou pelo RSVP são mantidas.

786-787 Ativa os temporizadores de inicialização e envio de reserva após a inicialização.

788 Atualiza o número de máquinas na rede como sendo 1 (um). A medida que as reservas forem chegando, esta variável é acrescida.

789 Guarda quem solicitou a inicialização (local ou remoto).

resv_process()

Após ter recebido uma mensagem de reserva, o **msg_process()** chama o **resv_process()** que irá executar o cálculo do atraso na rede, criar ou alterar a entrada na tabela de reservas do protocolo e enviar a reserva ao Controle de Admissão no *kernel*.

Para facilitar a visualização, a rotina será dividida em duas partes. Primeiro será analisado seu início, com as entradas, as variáveis e o cálculo do atraso (Figura 4.35). Em seguida será analisada a entrada na tabela de reservas e o envio ao *kernel* (Figura 4.36).

828-831 O **resv_process()** recebe como parâmetro o apontador da interface de rede, a mensagem recebida e a hora de chegada da mensagem.

833-838 Como variáveis do processo, temos dois apontadores para a lista de reserva (acrn e acrnl), um descritor de fluxo (new_flow), uma variável para cálculo de tempo (delta), uma variável para cálculo do atraso (msg_delay) e uma variável usada como indicador da reserva no *kernel*.

840 Atualiza o número de reservas recebidas.

841-842 Calcula a diferença de tempo entre a chegada da reserva e a inicialização.

843 Calcula o atraso da mensagem, subtraindo de delta o tempo contido na mensagem. Este tempo é calculado pela outra máquina da mesma forma com que foi calculado delta, abstraindo daí o atraso.

844-845 O horário de inicialização de quem enviou o AC_INIT será maior pois leva em conta o tempo de ida e volta da mensagem. Por isso dividimos por 2 (dois) o atraso.

ac_main.c

```
828    resv_process(t_obj, msg, now)
829    struct TP_obj *t_obj;
830    struct s_message *msg;
831    struct timeval now;
832    {
833    struct AC_net       *acm, *acmnl;
834    flow_spec new_flow;
835    struct ispsreq ir;
836    u_int32_t delta;
837    int32_t msg_delay;
838    isps_handle_t t_handle;
839
840            t_obj->tp_res_act++;
841            delta = ((now.tv_sec - t_obj->tp_last_init.tv_sec)*1000000
842                    +(now.tv_usec - t_obj->tp_last_init.tv_usec));
843            msg_delay = (int32_t)delta - (int32_t)msg->ac_wait;
844            if (t_obj->tp_who_init == LOCAL_INIT)
845                    msg_delay = msg_delay/2;
846            if ((double)msg_delay < 0){
847                    fprintf(fp,"Network time error. Delay adjusted to 0\n");
848                    msg_delay = 0;
849            }
850            net_calc(t_obj, msg_delay);
851            if (msg->acm_flags & FL_NULL)
852                    memset(&new_flow, 0, sizeof(new_flow));
853            else {
854                    new_flow.res_rate = msg->ac_flow.res_rate;
855                    new_flow.res_delay = msg->ac_flow.res_delay;
856            }
857            if (debug) {
858                    fprintf(fp,"Received Flow: Rate=%d | Delay=%.3fms | Wait=%.3fms\n",
859                            new_flow.res_rate, (double)new_flow.res_delay / 1000,
860                            (double)msg->ac_wait / 1000);
861                    fprintf(fp,"Network probe #%d: delay=%.3fms / Average=%.3fms\n",
862                            t_obj->tp_res_act, (double)msg_delay / 1000,
863                            (double)t_obj->tp_avg_delay / 1000);
864            }
```

resv_process()

Figura 4.35 - Início da função resv_process() do ACD

846-849 Se houver um atraso muito grande no envio da mensagem de inicialização, a máquina que receber uma mensagem de reserva de quem inicializou pode obter um valor negativo para o atraso. Neste caso, o *probe* é rejeitado.

850 Chama a rotina **net_calc()** que irá incluir o novo valor na tabela de atrasos e calcular o novo atraso médio. Esta rotina também irá atualizar a tabela do Controle de Admissão no *kernel*, com o novo valor obtido.

851-856 Se no *flag* da mensagem tiver sido colocado o valor FL_NULL, isto significa que deve-se atribuir 0 (zero) ao fluxo.

857-864 Mostra detalhes sobre a mensagem e o atraso, se a opção de *debug* (**-d** ou **-l**) for solicitada.

solicitados e bloqueia o fluxo para que ele não possa ser apagado pelo comando de *reset* do controle de admissão. Este comando é executado pelo RSVP.

899-905 Executa o envio de solicitação ao *kernel*, com a estrutura ir construída. Se a solicitação for aceita, o *kernel* deve retornar a identificação do fluxo. Este valor então é armazenado na lista de reservas do protocolo.

4.4 SBM - *Subnet Bandwidth Manager*

O SBM está atualmente na sua primeira versão *internet-draft* [YPH96], sendo discutido no ISSLL. Ele foi apresentado pela primeira vez no dia 25 de junho de 1996, durante a reunião do ISSLL, em Montreal, Canadá.

No SBM, há uma entidade lógica, que fica responsável pelo Controle de Admissão. O controle do segmento de rede é gerenciado por uma ou mais entidades chamadas DSBM (*Designated SBM*). O SBM é uma entidade da Camada de Aplicação que usa o UDP como protocolo de transporte e entende mensagens RSVP. A figura 4.1 ilustra o funcionamento do modelo.

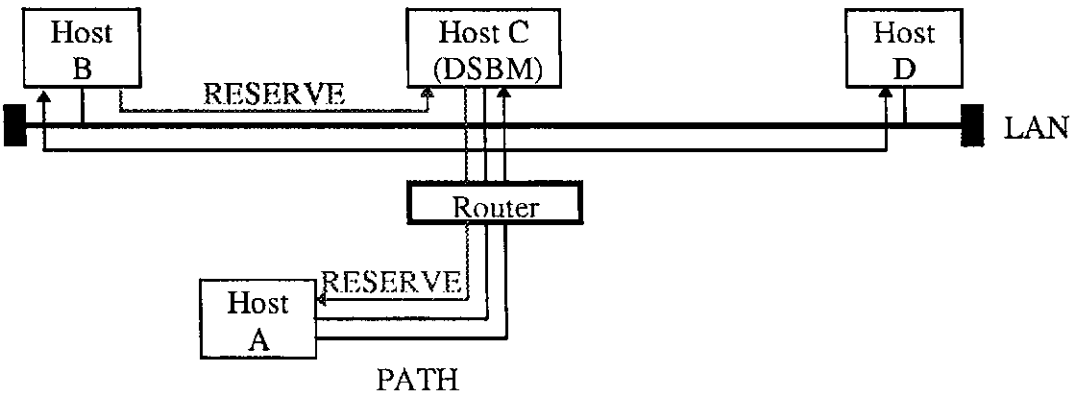


Figura 4.37 - Funcionamento do SBM

Como parte de sua configuração inicial, o DSBM obtém informações sobre a banda máxima que pode ser reservada em cada segmento de rede sob seu controle. Cada nó RSVP descobre seu DSBM, utilizando o *SBM Discovery Protocol*. Como no processamento normal do RSVP, as mensagens PATH são enviadas de um Transmissor (*Host* A) e retransmitidas por todo o caminho até atingir os potenciais Receptores (*Hosts* B e C). Quando um Receptor

(*Host* B, por exemplo) deseja fazer uma reserva, ele envia uma mensagem RESERVE através do RSVP ao DSBM, que irá processa-la e retransmiti-la, se for aceita.

Por se tratar da versão preliminar, ainda estão faltando diversos detalhes sobre seu funcionamento. Espera-se mais detalhes a medida que as discussões avancem. Segundo sua especificação, o mecanismo de localização dos DSBM e a identificação/delimitação das subredes exige equipamentos de controle do meio (*NICs, bridges, hubs e switches*) inteligentes para que possam repassar ao DSBM as informações que este necessita. A implementação deste modelo está em fase de preparação, não havendo assim versões disponíveis no momento que permitam uma comparação mais precisa com nossa proposição.

Capítulo 5
Classificação de Pacotes

No modelo de reserva de recursos associado à operação do protocolo IP, quando um pacote chega ao IP, vindo de uma interface de entrada ou de um protocolo superior deve ser classificado para que seja enviado à interface de saída correta e com as características de reserva que lhe foram atribuídas. O Classificador de Pacotes tem então a finalidade de identificar o fluxo de dados para que este possa receber o tratamento adequado.

No modelo IP tradicional, o roteamento atribui ao pacote a interface de saída e a rota de destino. A rota e a interface de saída podem variar, de acordo com o estado do caminho (mudanças de parâmetros, congestionamentos, queda de enlaces, etc.). No modelo IP com reserva de recursos é necessário que os pacotes sigam sempre uma mesma rota, independente de seu estado (Capítulo 3). Para isso, cada fluxo de dados deve ser identificado.

O cabeçalho IP não possui campos para identificação de fluxo, como ocorre com o STII [DB95] e como está previsto na nova versão do IP (IPv6) [DH96], onde a opção de identificação de fluxos de dados é contemplada [Par95]. Como a estrutura atual da Internet é baseada no IPv4, todas as colocações feitas neste capítulo são baseadas nele. E a forma encontrada para a identificação dos fluxos de dados no IPv4 é a utilização de filtros.

Quando uma reserva chega ao Controle de Admissão (Capítulo 4), ela vem com a interface de saída, seguida de um filtro contendo a identificação do fluxo de dados que utilizará tais recursos. Para isso é necessário que os pacotes sejam "abertos" e seus cabeçalhos lidos.

Se o filtro do pacote não for encontrado na lista de reservas, o pacote é classificado como *best-effort*. Neste caso é processada uma nova classificação para atribuir ao pacote uma prioridade de transmissão (Seção 5.6). Para otimizar o processamento na localização das reservas, os filtros são armazenados em uma tabela *hash* e uma função irá processar a pesquisa na tabela (Seção 5.5).

A forma ideal de classificação dos pacotes seria através de algoritmos de roteamento. Porém, os algoritmos ainda não estão preparados para funcionar em conjunto com o modelo de Integração de Serviços. No modelo aqui apresentado (Figura 5.1), a classificação ocorre após o roteamento ser efetuado. A rota utilizada por todos os pacotes de um fluxo de dados será a mesma utilizada pelo primeiro pacote transmitido. Se por algum motivo a rota for trocada e o pacote enviado para outra interface, o classificador retorna um erro de roteamento. Cabe ao RSVP solicitar a alteração da interface de transmissão e não ao Classificador de Pacotes.

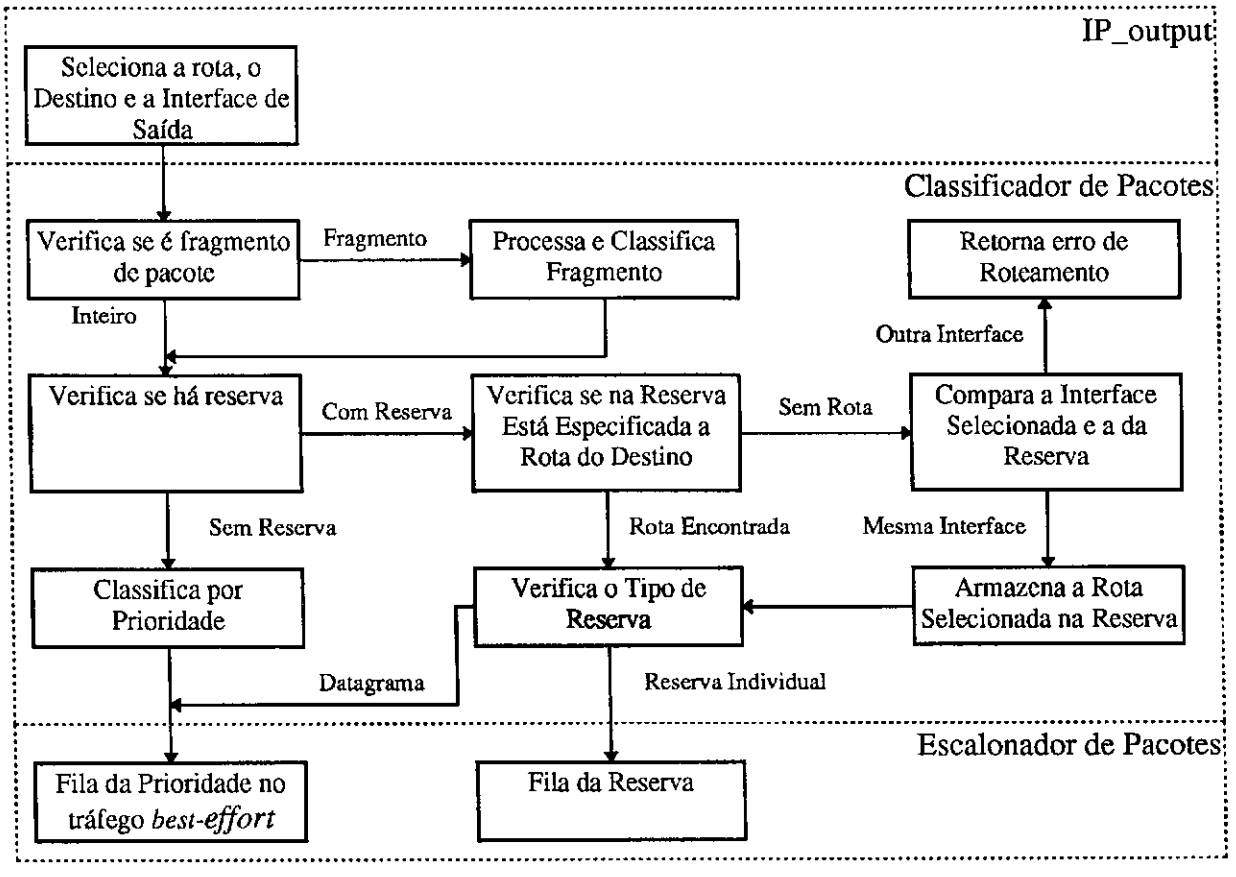


Figura 5.1 - Funcionamento do mecanismo de Classificação de Pacotes

O mecanismo do IP não está preparado para a classificação dos pacotes. Por isso se faz necessário a adaptação do mecanismo atual para o funcionamento com este tipo de modelo de operação (IP + RSVP).

Neste capítulo será vista a adaptação do IP para receber reserva de recursos (Seção 5.2), como é feita a identificação das reservas (Seção 5.3), como é processado um pacote fragmentado (Seção 5.4), o uso de tabela *hash* para identificação de reservas (Seção 5.5) e como se dá a classificação dos fluxos sem reserva (Seção 5.6).

5.1 Arquivos Referenciados

Na Tabela 5.1 estão os arquivos alterados ou criados para que o IP funcione com Classificação de Pacotes.

Arquivo	Função
/sys/net/ac.h	Arquivo com as estruturas e definições do mecanismo de Controle de Tráfego.
/sys/net/isps.h	Arquivo com as estruturas e definições da interface com o Controle de Tráfego.
/sys/net/ac_hash.h	Arquivo com as definições da tabela *hash* de reservas.
/sys/net/ac_isps.c	Rotinas do Controle de Admissão.
/sys/net/cl_isps.c	Rotinas usadas na Classificação de Pacotes.
/sys/netinet/ip_output.c	Rotina de processamento da saída da camada IP para a interface de rede.

Tabela 5.1 - Arquivos referenciados na Classificação de Pacotes

5.2 Adaptação do IP Para Classificação de Pacotes

Todos os protocolos acima da camada IP e os mecanismos de retransmissão enviam os pacotes para a função **ip_output()**. Esta função seleciona a rota de saída e seguindo a tabela de roteamento definida no momento, envia o pacote para a fila da interface de rede correspondente. A Figura 5.2 mostra um resumo deste modelo [WS95].

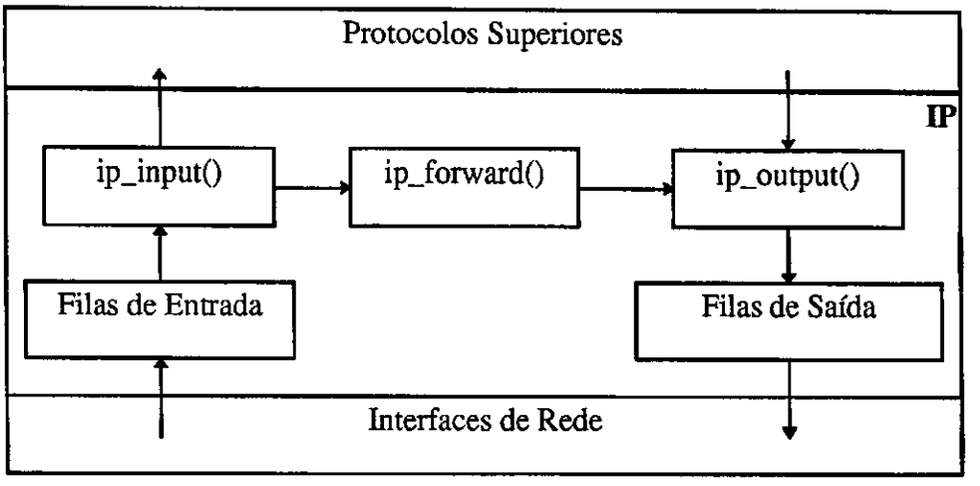


Figura 5.2 - Mecanismo básico de funcionamento do IP

Para incluir o mecanismo de classificação, altera-se a localização do envio dos pacotes para as filas de saída, incluindo em seu lugar a rotina de classificação de pacotes **isps_classifier()**. A Figura 5.3 ilustra a nova seqüência de operação do IP.

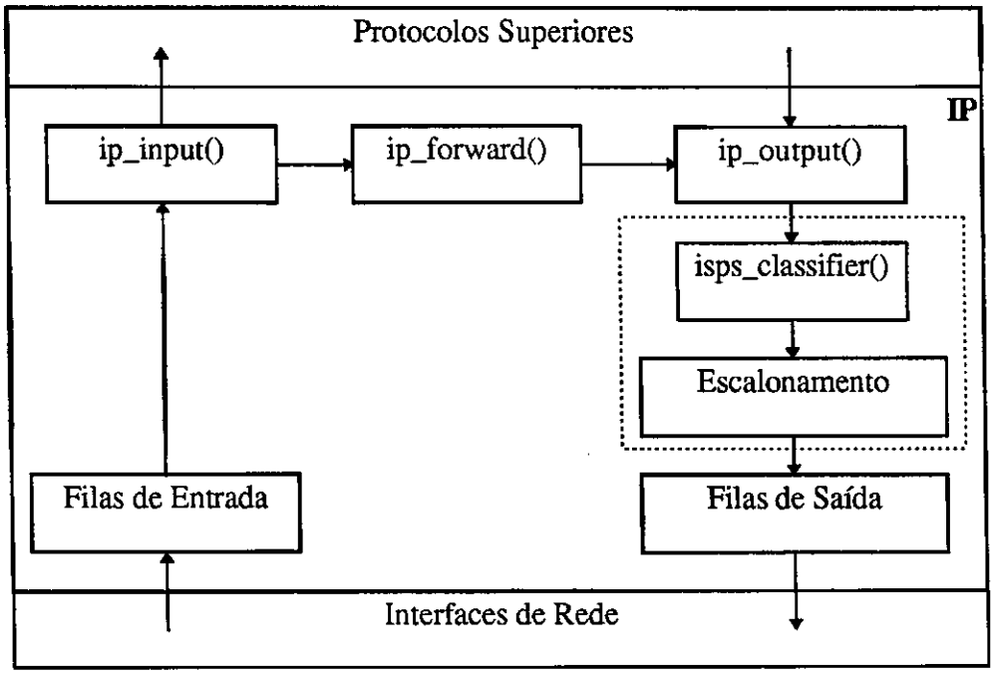


Figura 5.3 - Adaptação do IP para Classificação de Pacotes

A rotina de classificação executa os seguintes passos:

1. Verifica se é ou não um fragmento. Se for um fragmento, tenta identifica-lo;
2. Verifica se há reserva para o pacote;
3. Se houver reserva, chama a rotina que irá colocar o pacote na fila de reserva correspondente.

4. Se não for localizada uma reserva para o pacote, ele é definido como *best-effort* e lhe é atribuída uma prioridade.

Na Figura 5.4 pode ser observada a seqüência de operação proposta.

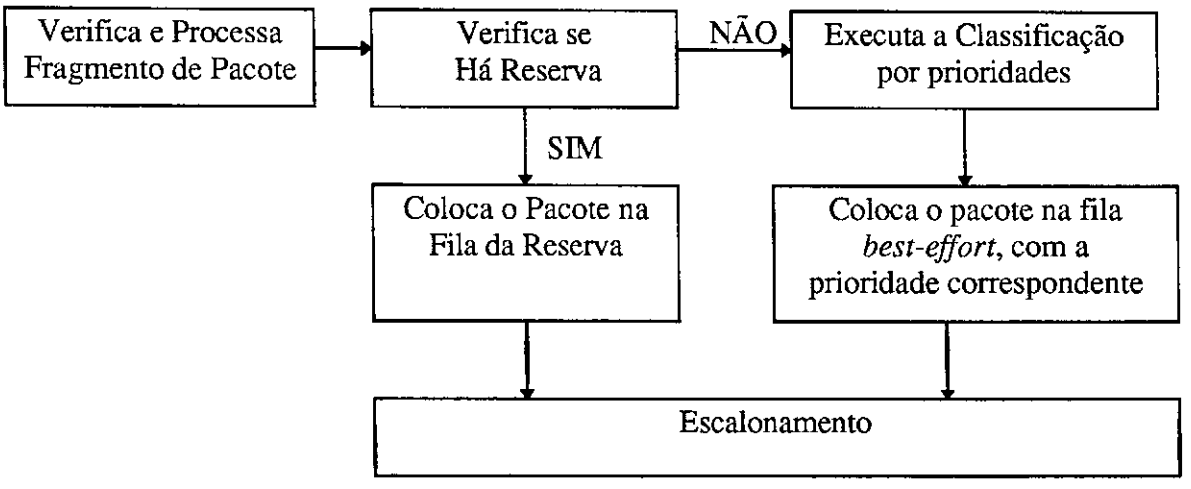


Figura 5.4 - Funcionamento do mecanismo de classificação

A implementação da adaptação consiste em substituir a chamada da rotina de saída da interface pela chamada ao Classificador de Pacotes, **isps_classifier()**. Isto ocorre em três pontos distintos da função, conforme indicado na Figura 5.5.

ip_output.c

```
                                    Código Original
 71   int isps_classifier __P((register struct ifnet *, struct mbuf *,
 72                                  struct sockaddr *, struct rtentry *));
                                    Código Original
329                     /* Call packet classifier subroutine */
330                     error = isps_classifier(ifp, m,
331                                  (struct sockaddr *)dst, ro->ro_rt);
332                     /* if don't exist ISPS
333                     error = (*ifp->if_output)(ifp, m,
334                                  (struct sockaddr *)dst, ro->ro_rt);
335                     */
                                    Código Original
418                          /* Call packet classifier subroutine */
419                          error = isps_classifier(ifp, m,
420                                  (struct sockaddr *)dst, ro->ro_rt);
421                          /* if don't exist ISPS
422                          error = (*ifp->if_output)(ifp, m,
423                             (struct sockaddr *)dst, ro->ro_rt);
424                          */
                                    Código Original
```

ip_output()

Figura 5.5 - Alterações na função ip_output()

70 - 73 Protótipo da função **isps_classifier()**, que substitui a chamada original de envio do pacote para a fila interface de rede.

329-335 Chamada da função **isps_classifier()**. São repassados os mesmos parâmetros que seriam passados à rotina apontada por ifp->if_output, como é mostrado no comentário.

418-424 Chamada da função **isps_classifier()**. São repassados os mesmos parâmetros que seriam passados à rotina apontada por ifp->if_output, como é mostrado no comentário.

5.3 Identificação de Reservas

Como foi visto na Seção 4.2.5, as reservas são precedidas de um filtro que irá identificar qual o tipo de tratamento cada pacote (dados) deve receber. Este filtro é formado pelo endereço IP do destino, pelo endereço IP da origem, pela porta do destino, pela porta da origem e pelo protocolo. A Figura 5.6 mostra a estrutura utilizada pelo RSVP e pelo Controle de Admissão para armazenar o filtro de identificação.

Além dos serviços definidos pelo Int-Serv (Capítulo 2), há na implementação da interface entre o RSVP e o Controle de Tráfego a definição de um serviço denominado *Datagram*. Pelo que pôde ser entendido, é atribuído reserva de prioridade para um fluxo de dados *best-effort*. Apesar de não haver especificações a respeito, a classificação deste tipo de tráfego também foi implementada para testes (Figura 5.9).

isps.h

```
61  typedef struct _filt isps_filt_t;
62  struct _filt {
63    int f_pf;                    /* Family (only PF_INET now) */
64    union {
65      struct {
66        struct in_addr _f_destaddr; /* IP dest address */
67        struct in_addr _f_srcaddr;  /* IP source address */
68        u_int16_t _f_destport;      /* TCP/UDP dest port */
69        u_int16_t _f_srcport;       /* TCP/UDP source port */
70        u_int8_t _f_protocol;       /* Xport protocol (IPPROTO_UDP, etc) */
71      } _f_ipv4;
72    } _f_spec;
73  };
74  #define f_ipv4_destaddr _f_spec._f_ipv4._f_destaddr
75  #define f_ipv4_srcaddr  _f_spec._f_ipv4._f_srcaddr
76  #define f_ipv4_destport _f_spec._f_ipv4._f_destport
77  #define f_ipv4_srcport  _f_spec._f_ipv4._f_srcport
78  #define f_ipv4_protocol _f_spec._f_ipv4._f_protocol
```

Figura 5.6 - Estrutura de filtros

61 Define o tipo de dado isps_filt_t, como sendo a estrutura _filt (Seção 3.6.3).

63 Família utilizada pelo filtro.

66 Endereço IP do destino.

67 Endereço IP da origem.

68 Porta TCP/UDP do destino.

69 Porta TCP/UDP da origem.

80 Tipo de protocolo do pacote.

74-78 Atalhos para as estruturas.

Os endereços IP de origem e destino são facilmente identificados no cabeçalho IP (Figura 5.7) [POS81]. Mas, as portas TCP/UDP precisam ser retiradas do campo de dados do pacote que, por sua vez, encapsula os segmentos TCP/UDP.

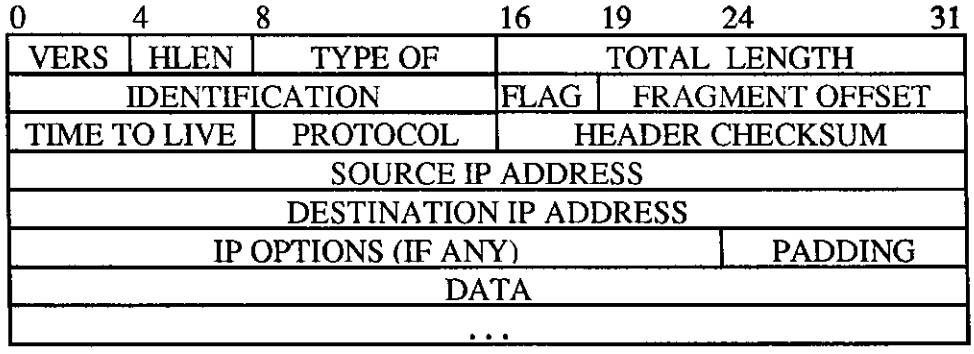


Figura 5.7 - Mensagem IP

Tanto o protocolo UDP, quanto o protocolo TCP colocam em seu cabeçalho suas portas de destino e de origem numa posição fixa em relação à origem dos segmentos. Como isso é padrão, facilita a identificação das portas. A Figura 5.8 mostra como são armazenadas as portas após sua identificação.

ac.h

```
174   /* UDP/TCP source and dest ports */
175   struct ports {
176           u_short sport; /* source port */
177           u_short dport; /* destination port */
178   };
```

Figura 5.8 - Estrutura de portas

176 Porta de origem

177 Porta de destino.

Os problemas passam a ser então, a fragmentação dos pacotes e a criptografia. A fragmentação pode ser resolvida facilmente e será descrita na Seção 5.4. Quanto à criptografia,

se esta for feita pela camada de aplicação, não há problema. Pois, os cabeçalhos TCP/UDP não serão criptografados. Porém, se a criptografia for realizada após o encapsulamento TCP/UDP, não haverá como descobrir as portas. Já é comum a criptografia de pacotes na camada IP. Em [Atk95] e [Bel94] são feitas considerações a respeito dos mecanismos de segurança para o IPv4 e do IPv6.

O processo de classificação de pacotes é executado pela função **isps_classifier()**. Esta função irá tratar o problema da fragmentação (Seção 5.4) e chamar as rotinas para a localização de reservas ou tratamento do fluxo *best-effort* (Seção 5.6). A Figura 5.9 mostra a implementação do mecanismo de Classificação de Pacotes.

346-350 A rotina **isps_classifier()** recebe como parâmetros: o apontador para a interface (ifp); a estrutura de mbuf (m0); o endereço do destino (dst); e a rota para o destino (rt0).

352-363 A lista de variáveis internas é composta por: um apontador de reservas (rtp); uma estrutura IP (ip); uma estrutura de portas (tports); três inteiros de 16 bits (ip_id, iplen e off); uma estrutura de tempo (now); duas estruturas de fragmentos (ip_ft e ip_ft0); um apontador para a estrutura de interfaces do Controle de Admissão (acp); um inteiro de 32 bits (delta); e duas variáveis inteiras (priority e error).

365-366 Se o Controle de Tráfego estiver desativado, envia o pacote diretamente para a interface.

367 Atribui a ip o conteúdo do mbuf.

368 Inicializa o apontador de reservas.

369 Atribui a tports as portas contidas no pacote.

370 Atribui a iplen o tamanho do pacote.

371-422 Verifica se o pacote é um fragmento e processa-o, se for o caso (Figura 5.12).

423-424 Se não é um fragmento, pesquisa o filtro do pacote (Figura 5.10).

425-433 Se não for encontrada uma reserva, o pacote é processado como *best-effort* (Seção 5.6). Sua prioridade é calculada e ele é colocado na fila de escalonamento (Seção 6.2). Se o pacote pertencer ao protocolo DCRP (Seção 4.3), ele é enviado diretamente à interface de rede.

434-439 Se foi localizada reserva para o pacote, é verificado se esta pertence à Classe de Serviço *Datagram* (Seção 3.6.2). Se for, envia o pacote como tráfego *best-effort*, conforme a prioridade solicitada na reserva.

ac_isps.c

```
346  int isps_classifier(ifp, m0, dst, rt0)
347          register struct ifnet *ifp;
348          struct mbuf *m0;
349          struct sockaddr *dst;
350          struct rtentry *rt0;
351  {
352  struct RES_table *rtp;
353  struct ip  *ip;
354  struct ports          *tports;
355  u_short   ip_id;
356  short                 iplen;
357  short                 off;
358  struct timeval        now;
359  struct ip_frag        *ip_ft, *ip_ft0;
360  struct AC_obj         *acp;
361  u_int32_t             delta;
362  int                   priority;
363  int                   error=0;
365              if (!tc_isps_on)
366                 return((*ifp->if_output)(ifp, m0, (struct sockaddr *)dst, rt0));
367              ip = mtod(m0, struct ip *);
368              rtp = NULL;
369              tports = (struct ports *) ((caddr_t) ip + (ip->ip_hl << 2));
370              iplen = ntohs(ip->ip_len);
```

Figura 5.12

```
423              } else
424                 rtp = isps_search_filter(ip, tports);
425              if (!rtp) {
426                 priority = classify_datagram(ip->ip_tos, ip->ip_p, tports);
427                 if (priority == 8)
428                        /* DCRP package */
429                        error = (*ifp->if_output)(ifp, m0, (struct sockaddr *)dst, rt0);
430                 else
431                        /* best-effort package */
432                        error = cl_datagram(priority, ifp, m0,
433                                                (struct sockaddr *)dst, rt0, iplen);
434              } else {
435                 if (rtp->res_spec.fs_tos == TOS_DATAGRAM){
436                        /* Reserved datagram flow */
437                        priority = rtp->res_spec.fs_rspec.rs_p.p_dly;
438                        error = cl_datagram(priority, ifp, m0,
439                                                (struct sockaddr *)dst, rt0, iplen);
440                  } else {
441                        /* reserved flow */
442                        if (!rtp->res_queue.qu_rt) {
443                           acp = rtp->res_obj;
444                           if (acp->ac_ifp != ifp)
445                                  return(ENETUNREACH);
446                           rtp->res_queue.qu_rt = rt0;
447                           rtp->res_queue.qu_dst = *dst;
448                        }
449                        error = cl_enque(m0, rtp, iplen);
450                  }
451              }
452              return(error);
453  }
```

isps_classifier()

Figura 5.9 - Função isps_classifier()

440-449 Se for localizada uma reserva para outras Classes de Serviço, verifica se já há uma rota cadastrada na reserva. Se não, cadastra e aplica a rota selecionada

pelo IP. Se a interface de destino não corresponde à interface da reserva, retorna erro de roteamento. Finalmente, coloca o pacote na fila do Escalonador, com as informações da reserva.

Para verificar se há reserva para o pacote, é necessário pesquisar na lista de reservas, o filtro correspondente aos parâmetros do pacote. Isso é feito através da rotina **isps_search_filter()**, apresentada na Figura 5.10.

cl_isps.c

```
66    struct RES_table *isps_search_filter(ip, tports)
67    struct ip *ip;
68    struct ports *tports;
69    {
70    struct RES_table *rtp = NULL;
71    ihs       *ihsp;
72
73            if (ip->ip_p != IPPROTO_RSVP){
74              ihsp = isps_find_hash((struct in_addr)ip->ip_src,
75                                    (struct in_addr)ip->ip_dst,
76                                    tports->sport, tports->dport, ip->ip_p);
77              if (ihsp)
78                    rtp = ihsp->hs_res;
79            }
80            return(rtp);
81    }
```

isps_search_filter()

Figura 5.10 - Função isps_search_filter()

66-68 A função **isps_search_filter()** recebe como parâmetros o pacote IP (ip) e a estrutura com a identificação das portas (tports).

70-71 As variáveis da função são um apontador para a lista de recursos (rtp) e a da tabela *hash* (ihsp).

73 A estrutura da mensagem RSVP é diferente das estruturas TCP/UDP. Por isso, não há identificação de portas na mensagem. A tentativa de localização de reservas em um pacote RSVP causa erro de perda de descritores no *kernel*. Como não existe reservas de recursos para o tráfego RSVP, não é realizada a pesquisa e envia-se as mensagens como *best-effort*.

74-76 Procura a reserva a partir dos valores contidos no pacote (Figura 5.16).

77-78 Se for encontrada um filtro correspondente aos valores procurados na tabela, atribui o apontador contido na tabela ao apontador de reserva.

80 Retorna o apontador para a reserva.

5.4 Fragmentação

Cada tipo de interface de rede possui um limite para o tamanho dos pacotes que irão trafegar no meio físico. Este valor é denominado MTU (*Maximum Transmission Unit*) [Com91]. Quando uma aplicação gera mensagens maiores do que a MTU da interface, é necessário a fragmentação das mesmas para que possam ser transmitidas com sucesso. Quando os pacotes chegam ao seu destino são novamente reagrupados antes de serem repassados às camadas superiores.

Além das aplicações, pode ser necessário a fragmentação ou refragmentação de pacotes ao longo do caminho entre a origem e o destino destes. Isso ocorre devido à heterogeneidade dos enlaces físicos existentes na Internet. Os pacotes saem de um enlace ou rede com um MTU grande e são repassados para outro com MTU menor. Essa fragmentação de pacotes é feita pelo IP (rotina **ip_output**()) antes de repassar os pacotes para a interface de rede.

O mecanismo de fragmentação de pacotes pelo IP utiliza os campo IDENTIFICATION, FLAGS e FRAGMENT OFFSET do cabeçalho IP (Figura 5.7). Quando é necessário a fragmentação, é desativado o *flag* IP_DF, que indica ser este um pacote inteiro e não um fragmento. Cada fragmentação recebe uma identificação única (IDENTIFICATION). Cada fragmento de pacote também é identificado seqüencialmente, de acordo com seu tamanho (FRAGMENT OFFSET). Essa identificação será usada mais tarde para a recomposição dos mesmos [WS95].

Na fragmentação de pacotes, somente o primeiro pacote carrega as informações (portas) necessárias para a identificação dos pacotes pelo Classificador de Pacotes.

Apesar das especificações que tratam do assunto de reserva de recursos e Serviços Integrados definirem, que a fragmentação de pacotes de dados e de controle deva ocorrer apenas no momento da transmissão em sua origem, a convivência com múltiplas fragmentações ainda deve ser longa.

Para evitar a fragmentação é necessário descobrir a menor MTU ao longo do caminho do fluxo de dados e a aplicação deve gerar mensagens que sejam menores ou iguais à esta MTU. Para que se possa utilizar enlaces com MTUs diferentes em ambiente com reserva de recursos, sem a necessidade de pesquisar a menor MTU ao longo do caminho, é sugerido neste documento a utilização de um mecanismo para o tratamento dos pacotes fragmentados para que eles possam ser corretamente identificados e transmitidos de acordo com as reservas de recursos solicitadas.

Os pacotes fragmentados ao chegarem em um roteador são identificados e transmitidos de acordo com a reserva do fluxo. Caso o pacote não consiga ser identificado ou não haja reserva de recursos para ele, este é transmitido como *best-effort*.

Para a correta transmissão dos pacotes com reserva de recursos, é necessário guardar em uma tabela a identificação dos pacotes fragmentados (IDENTIFICATION), que possuam reserva de recursos. Esta informação ficará armazenada durante algum tempo e se após este tempo não chegar o final do pacote, a informação é descartada. A temporização utilizada não é a mesma usada pelo IP. O tempo de armazenamento será o mesmo definido na implementação do RSVP como o maior atraso possível para reserva de recursos (ISPS_CLASS_MAX_DELAY) [ISI96]. A Figura 5.11 mostra como a tabela com as informações sobre fragmentos é formada.

cl_isps.c

```
53  struct ip_frag {
54          struct ip_frag *f_next;
55          u_short f_ip_id;
56          struct timeval f_time;
57          struct ports f_ports;
58          struct RES_table *f_res;
59  };
```

Figura 5.11 - Estrutura para armazenamento de informações sobre fragmentos de pacotes

54 Próxima identificação de fragmento.

55 Identificação de fragmento IP [WS95].

56 Hora de inclusão da identificação. Isto será usado para indicação de estouro de temporização (*time-out*) da identificação.

57 Identificação das portas de origem e destino TCP/UDP.

58 Apontador para a reserva.

A Figura 5.12 mostra a continuação da função **isps_classifier()** apresentada na Figura 5.9, onde pode ser observado o processamento dos fragmentos de pacotes.

372 A variável off armazena o *offset* do protocolo e seus *flags*.

373 Verifica se é um fragmento.

374 Retira de off os *flags*. Resta então, o valor com a identificação de cada fragmento, de um mesmo pacote.

375 Verifica se é o primeiro fragmento.

377-392 Verifica se há reserva para o pacote. Se houver, inclui os dados do pacote na lista de fragmentos.

393-422 Se é continuação de um pacote fragmentado, procura na lista de fragmentos a identificação do pacote. Se achar alguma identificação a muito tempo armazenada, elimina esta identificação da lista. Se achar a identificação e for o último fragmento, também elimina a identificação da lista.

ac_isps.c

```
371          /* package fragment test */
372          off = ntohs((u_short)ip->ip_off);
373          if (off &~ IP_DF) {
374             off = off & IP_OFFMASK;
375             ip_id = ntohs((u_short)ip->ip_id);
376             if (off == 0) {
377                   rtp = isps_search_filter(ip, tports);
378                   if (rtp) {
379                      ip_ft = (struct ip_frag *) malloc(sizeof(struct ip_frag),
380                                                          M_IFADDR, M_WAITOK);
381                      if (ip_ft){
382                            ip_ft->f_ip_id = ip_id;
383                            ip_ft->f_time = time;
384                            ip_ft->f_next = NULL;
385                            if (!FT_HEAD)
386                               FT_HEAD = ip_ft;
387                            else
388                               (FT_TAIL)->f_next = ip_ft;
389                            FT_TAIL = ip_ft;
390                            ip_ft->f_res = rtp;
391                      }
392                   }
393             } else {
394                   ip_ft = FT_HEAD;
395                   now = time;
396                   ip_ft0 = 0;
397                   while (ip_ft) {
398                      if ((ip_ft->f_ip_id == ip_id) &&
399                         (rtp->res_filt.f_ipv4_destaddr.s_addr == ip->ip_dst.s_addr)&&
400                            (rtp->res_filt.f_ipv4_srcaddr.s_addr == ip->ip_src.s_addr) &&
401                            (rtp->res_filt.f_ipv4_protocol == ip->ip_p)) {
402                               rtp = ip_ft->f_res;
403                               break;
404                      }
405                      delta = ((now.tv_sec - ip_ft->f_time.tv_sec)*1000000 +
406                                 (now.tv_usec - ip_ft->f_time.tv_usec));
407                      if ((delta > ISPS_CLASS_MAX_DELAY*1000) || !(off&IP_MF)) {
408                            if (ip_ft == FT_HEAD)
409                               FT_HEAD = ip_ft->f_next;
410                            if (ip_ft == FT_TAIL)
411                               FT_TAIL = ip_ft->f_next;
412                            if (ip_ft0)
413                               (ip_ft0)->f_next = ip_ft->f_next;
414                            ip_ft0 = ip_ft->f_next;
415                            free(ip_ft, M_IFADDR);
416                            ip_ft = ip_ft0;
417                      } else {
418                            ip_ft0 = ip_ft;
419                            ip_ft = ip_ft->f_next;
420                      }
421                   }
422             }
423          } else
```

isps_classifier()

Figura 5.12 - Tratamento de fragmentos

5.5 Tabela *Hash*

Para agilizar a busca de reservas a partir da identificação do pacote é usada uma tabela *hash*. Esta tabela possui 64 (sessenta e quatro) entradas, apontando para uma fila de identificadores (Figura 5.13).

ac_hash.h

```
55    struct t_hash    isps_hash_table[64];    /* hash table */
```

Figura 5.13 - Tabela *hash*

A função *hash* utilizada é bastante simples, sendo calculada a partir do somatório dos campos de identificação do filtro (Figura 5.14).

ac_isps.c

```
64    u_int32_t isps_calc_hash(s_addr, d_addr, s_port, d_port, protocol)
65    struct in_addr    s_addr;
66    struct in_addr    d_addr;
67    u_short  s_port;
68    u_short  d_port;
69    u_char   protocol;
70    {
71    u_int32_t         hash_val;
72
73            hash_val = ((u_int32_t)s_addr.s_addr + (u_int32_t)d_addr.s_addr +
74                        (u_int32_t)s_port + (u_int32_t)d_port +
75                        (u_int32_t)protocol) & ISPS_HASH_MASK;
76            return(hash_val);
77    }
```

isps_calc_hash()

Figura 5.14 - Cálculo do valor *hash*

64-69 A função **isps_calc_hash()** recebe como parâmetros o endereço de origem (s_addr), o endereço de destino (d_addr), a porta de origem (s_port), a porta de destino (d_port) e o protocolo (protocol).

71 A única variável é a de cálculo da função (hash_val).

73-75 Calcula o valor através do somatório de todos os parâmetros e no final atribui uma máscara de 32 (trinta e dois) bits, com os 6 (seis) bits menos significativos ativos (0x0000003f).

76 Retorna o valor calculado.

A tabela *hash* é formada pela estrutura isps_hash é mostrada da Figura 5.15.

ac_hash.h

```
39  typedef struct isps_hash {
40          struct  isps_hash       *hs_next;       /* Next occurrence */
41          struct  RES_table       *hs_res;        /* ISPS reserve pointer */
42  #define hs_protocol         ish_res->res_filt.f_ipv4_protocol /* filter protocol */
43  #define hs_srcaddr ish_res->res_filt.f_ipv4_srcaddr  /* filter source address */
44  #define hs_srcport ish_res->res_filt.f_ipv4_srcport  /* filter source port */
45  #define hs_destaddr         ish_res->res_filt.f_ipv4_destaddr /* filter dest address */
46  #define hs_destport         ish_res->res_filt.f_ipv4_destport /* filter source port */
47  } ihs;
```

Figura 5.15 - Estrutura da tabela *hash*

40 Aponta para a próxima entrada com o mesmo valor *hash*.

41 Apontador para a reserva.

42-46 Atalhos para a descrição do filtro na reserva.

Além da função de cálculo também há as seguintes rotinas:

- uma função de pesquisa a partir do conteúdo do pacote (Figura 5.16);
- uma função de inclusão de um filtro na tabela *hash* (Figura 5.17) e;
- uma função de exclusão de um filtro da tabela (Figura 5.18).

A Figura 5.16 mostra a implementação da função de pesquisa de reservas na tabela *hash*, a partir dos parâmetros de identificação dos pacotes.

ac_isps.c

```
147 ihs *isps_find_hash(s_addr, d_addr, s_port, d_port, protocol)
148 struct in_addr     s_addr;
149 struct in_addr     d_addr;
150 u_short  s_port;
151 u_short  d_port;
152 u_char   protocol;
153 {
154 struct RES_table *rtp = 0;
155 ihs      *ihsp;
156 u_int32_t hash_val;
157
158
159         hash_val = isps_calc_hash(s_addr, d_addr, s_port, d_port, protocol);
160         ihsp = isps_hash_table[hash_val].ht_head;
161         while(ihsp) {
162                 rtp = ihsp->hs_res;
163                 if ((rtp->res_filt.f_ipv4_destaddr.s_addr == d_addr.s_addr) &&
164                    (rtp->res_filt.f_ipv4_srcaddr.s_addr == s_addr.s_addr) &&
165                    (rtp->res_filt.f_ipv4_destport == d_port) &&
166                    (rtp->res_filt.f_ipv4_srcport == s_port) &&
167                    (rtp->res_filt.f_ipv4_protocol == protocol))
168                         break;
169                 ihsp = ihsp->hs_next;
170         }
171         return(ihsp);
172 }
```

isps_find_hash()

Figura 5.16 - Pesquisa na tabela *hash*

147-152 A função **isps_find_hash()** recebe como parâmetros o endereço de origem (s_addr), o endereço de destino (d_addr), a porta de origem (s_port), a porta de destino (d_port) e o protocolo (protocol).

154-156 As variáveis utilizadas são a estrutura de reservas (rtp), a estrutura da tabela *hash* (ihsp) e o valor *hash* (hash_val).

159 Calcula o valor *hash*.

160 Atribui a ihsp o topo da lista de filtros com valores iguais, na tabela *hash*.

161-170 Pesquisa na lista uma entrada que seja exatamente igual à solicitada.

171 Retorna a estrutura da tabela *hash*.

As funções **isps_add_hash()** (Figura 5.17) e **isps_del_hash()** (Figura 5.18) são chamadas a partir do Controle de Admissão (Seção 4.2.5), durante a inclusão e exclusão de filtros, respectivamente. A Figura 5.17 mostra como funciona a função **isps_add_hash()**.

ac_isps.c

```
83   int isps_add_hash(rtp)
84   struct RES_table *rtp;
85   {
86   u_int32_t          hash_val;
87   ihs        *ihsp;
88
89             ihsp = (ihs *) malloc(sizeof(ihs), M_IFADDR, M_WAITOK);
90             if (!ihsp)
91                       return(-1);
92             hash_val = isps_calc_hash(rtp->res_filt.f_ipv4_srcaddr,
93                                       rtp->res_filt.f_ipv4_destaddr,
94                                       rtp->res_filt.f_ipv4_srcport,
95                                       rtp->res_filt.f_ipv4_destport,
96                                       rtp->res_filt.f_ipv4_protocol);
97             ihsp->hs_res = rtp;
98             ihsp->hs_next = NULL;
99             if (!isps_hash_table[hash_val].ht_head)
100                      isps_hash_table[hash_val].ht_head = ihsp;
101            else
102                      (isps_hash_table[hash_val].ht_tail)->hs_next = ihsp;
103            isps_hash_table[hash_val].ht_tail = ihsp;
104            return(0);
105  }
```

isps_add_hash()

Figura 5.17 - Adição de filtros na tabela *hash*

83-84 A função **add_hash()** recebe como parâmetro a reserva feita.

86-87 É usada uma variável para o valor *hash* e um apontador para a estrutura *hash*.

89-91 Aloca para ihsp um endereço de memória, retornando em caso de erro.

92-96 Calcula o valor *hash*.

97 Atribui à estrutura um apontador para a reserva.

98-103 Inclui a estrutura ihsp na tabela *hash*.

A exclusão de filtros chamada através da função **isps_del_hash**() (Figura 5.18) ocorre tanto em uma solicitação de exclusão de filtro, quanto em uma solicitação de exclusão de reserva.

ac_isps.c

```
111  int isps_del_hash(rtp)
112  struct RES_table *rtp;
113  {
114  u_int32_t          hash_val;
115  ihs        *ihsp, *ihsp0;
116
117            hash_val = isps_calc_hash(rtp->res_filt.f_ipv4_srcaddr,
118                                      rtp->res_filt.f_ipv4_destaddr,
119                                      rtp->res_filt.f_ipv4_srcport,
120                                      rtp->res_filt.f_ipv4_destport,
121                                      rtp->res_filt.f_ipv4_protocol);
122            ihsp = isps_hash_table[hash_val].ht_head;
123            ihsp0 = NULL;
124            while(ihsp){
125                      if (ihsp->hs_res == rtp)
126                                break;
127                      ihsp0 = ihsp;
128                      ihsp = ihsp->hs_next;
129            }
130            if (ihsp) {
131                      if (ihsp == isps_hash_table[hash_val].ht_head)
132                                isps_hash_table[hash_val].ht_head = ihsp->hs_next;
133                      if (ihsp == isps_hash_table[hash_val].ht_tail)
134                                isps_hash_table[hash_val].ht_tail = ihsp0;
135                      if (ihsp0)
136                                (ihsp0)->hs_next = ihsp->hs_next;
137                      free(ihsp, M_IFADDR);
138            } else
139                      return(-1);
140            return(0);
141  }
```

isps_del_hash

Figura 5.18 - Eliminação de filtros da tabela *hash*

111-112 A função **del_hash**() recebe como parâmetro a reserva feita.

114-115 É usada uma variável para o valor *hash* e um apontador para a estrutura *hash*.

117-121 Calcula o valor *hash*.

122-139 Elimina da tabela *hash*, o filtro usado na reserva (se encontrar).

5.6 Priorização de Pacotes

Como não foi localizado um documento que defina as regras a serem utilizadas na classificação do tráfego *best-effort* em ambiente de reserva de recursos, foram propostas e implementadas algumas regras para a Classificação de Pacotes.

Primeiramente, atribui-se 8 (oito) níveis de prioridade para o tráfego *best-effort*. Os pacotes de mais alta prioridade, recebem a classificação 7 (sete). Os pacotes de mais baixa prioridade recebem a classificação 0 (zero).

Define-se então as regras para a priorização dos pacotes. Foram detectadas quatro formas de identificar a prioridade dos pacotes. Todas as quatro formas de identificação foram implementadas e a procura segue a seguinte ordem:

1. utilizar o subcampo PRECEDENCE do cabeçalho IP;
2. utilizar os bits do subcampo TOS do cabeçalho IP e;
3. identificar os pacotes através das portas UDP/TCP conhecidas.

O pacote/datagrama IP (Figura 5.7) possui um campo chamado TYPE OF SERVICE [Pos81]. Este campo é usado pelos algoritmos de roteamento para a seleção do caminho ideal para a transmissão do pacote/datagrama entre as rotas possíveis, conforme o tipo de dado que o pacote carrega.

A Figura 5.19 expande o formato do campo TYPE OF SERVICE, onde o campo PRECEDENCE estabelece a prioridade do datagrama entre 0 (normal) e 7 (controle de rede), permitindo ao remetente estabelecer a importância de cada datagrama. Há também os bits D, T, R e C, que indicam quando ativados, baixo atraso, alto *throughput*, confiabilidade e custo, respectivamente [Ste94].

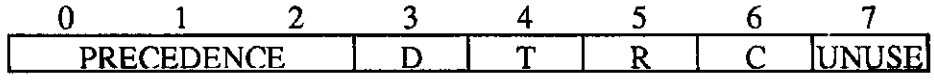


Figura 5.19 - Campo TYPE OF SERVICE

No roteamento IP normal os bits **D, T, R** e **C** são utilizados para a definição da melhor rota. Este conjunto de bits é conhecido como campo TOS. O campo PRECEDENCE comporta valores entre 0 (zero) e 7 (sete) e é atualmente muito pouco usado.

O verificação do campo PRECEDENCE como primeira escolha de priorização é uma sugestão deste documento. Por ser um campo pouquíssimo usado e comportar 8 (oito) níveis de prioridades, acredito que este campo seja o ideal para classificar este tipo de tráfego.

Se nenhum valor for atribuído ao campo PRECEDENCE, os quatro bits seguintes (campo TOS) são usados na priorização, conforme a Tabela 5.2. O uso do campo TOS como segunda opção é devido à sua divulgação. Este campo possui regras claras de utilização [Alm92] e já vem sendo largamente utilizado no roteamento de pacotes. Protocolos como o ICMP e aplicações como Telnet e FTP usam estes bits para priorizar seus tráfegos durante o roteamento.

A atribuição de valores ao campo TYPE OF SERVICE, por parte da aplicação, é extremamente simples, através da utilização da opção de comando IP_TOS do *socket*. Para a utilização do campo PRECEDENCE não há regras definidas. Mas, para o campo TOS existe um documento sugerindo regras de utilização por parte das aplicações e protocolos [Alm92].

Bit	Prioridade
Low Delay	7
Throughput	6
Reliability	5
Minimum Cost	4

Tabela 5.2 - Tabela de prioridades para TOS

Se não houver qualquer valor no campo TYPE OF SERVICE, é sugerido que se identifique o tipo de aplicação que gerou o pacote através da porta TCP/UDP conhecida [BCS94]. Para isso são definidos três tipos de prioridades:

- *Interactive burst* (prioridade 7):

  Telnet, X e NFS.

- *Interactive bulk transfer* (prioridade 6):

  FTP.

- *Asynchronous bulk transfer* (prioridade 5):

  E.MAIL e FAX.

As aplicações que não se encaixarem em nenhuma dessas três classes, são enviadas com prioridade 0 (zero). Aplicações de som e vídeo também podem ser identificadas através de suas portas. A Figura 5.22 mostra como é implementada esta identificação.

Os pacotes do protocolo de controle DCRP (Seção 4.3.2) também são classificados. Porém, após sua identificação, o pacote é transmitido diretamente, sem atrasos ou escalonamento. Isso se faz necessário pois a disputa ou o atraso nas filas de saída do Escalonador poderiam levar a uma inconsistência no estado das reservas na rede.

A Figura 5.20 ilustra a implementação do mecanismo de classificação.

cl_isps.c

```
253  int classify_datagram(tos, protocol, tports)
254  u_char tos;
255  u_char protocol;
256  struct ports *tports;
257  {
258  u_char precedence;
259  int priority = 0;
260
261          if (tos){
262            precedence = tos >> 5;
263            if (precedence)
264                    priority = precedence;
265            else
266                    switch (tos) {
267                      case IPTOS_LOWDELAY:
268                              priority = 7;
269                              break;
270                      case IPTOS_THROUGHPUT:
271                              priority = 6;
272                              break;
273                      case IPTOS_RELIABILITY:
274                              priority = 5;
275                              break;
276                      case IPTOS_MINCOST:
277                              priority = 4;
278                              break;
279                    }
280          } else
281            switch(protocol) {
282                    case IPPROTO_TCP:
283                      priority = classify_udp_tcp(tports);
284                      break;
285                    case IPPROTO_UDP:
286                      priority = classify_udp_tcp(tports);
287                      break;
288                    case IPPROTO_RSVP:
289                      priority = 7;
290                      break;
291            }
292          return(priority);
293  }
```

classify_datagram()

Figura 5.20 - Classificação de datagramas

253-256 A rotina **classify_datagram()** recebe como parâmetros: o campo TOS (Figura 5.19) do pacote IP (tos); o protocolo usado (protocol); e a estrutura de portas (tports).

258-259 As variáveis utilizadas são o valor de PRECEDENCE (precedence) e a prioridade (priority).

261 Verifica se existe algum valor em tos.

262-264 Se houver precedence, atribui este valor à prioridade.

265-279 Se houver valor em tos, testa o tipo de prioridade (Tabela 5.2).

280-291 Se não houver valor em tos, verifica o protocolo. Se o protocolo for UDP ou TCP, chama a rotina **classify_udp_tcp()** e se for RSVP, atribui prioridade 7 (mais alta).

292 Retorna a prioridade selecionada.

Se não for necessária a classificação do pacote pelo número da porta, é chamada a função **classify_udp_tcp()**. Essa função fará a pesquisa tanto pela porta TCP, quanto pela UDP. Na maioria das aplicações identificadas as portas são as mesmas em ambos os protocolos. Primeiro, pesquisa-se pela porta de destino. Depois pela porta de origem. A Figura 5.21 mostra como é feita essa classificação.

cl_isps.c

```
233   int classify_udp_tcp(tports)
234   struct ports *tports;
235   {
236   u_short s_port, d_port;
237   int priority;
238
239           d_port = ntohs(tports->dport);
240           s_port = ntohs(tports->sport);
241           /* Veriy Source port */
242           priority = port_classify(ntohs(tports->sport));
243           /* Verify Destination port */
244           if (!priority)
245                   priority = port_classify(ntohs(tports->dport));
246           return(priority);
247   }
```

classify_udp_tcp()

Figura 5.21 - Definição do protocolo usado nos datagramas

233-234 A função recebe como parâmetro a estrutura com as portas.

236-237 As variáveis locais consistem de duas portas (s_port e d_port) e a prioridade selecionada (priority).

239-240 Atribui às variáveis s_port e d_port os valores das portas na ordem de bits correta.

242 Verifica se a porta de origem é conhecida.

244-245 Se a porta de origem não for conhecida, verifica a porta de destino.

246 Retorna a prioridade. Se nenhuma das portas for conhecida, a prioridade será 0 (zero).

O teste de portas é feito pela função **port_classify()** (Figura 5.22).

cl_isps.c

```
194  int port_classify(port)
195  u_short port;
196  {
197  int priority = 0;
198
199          switch(port) {
200                  /* Distributed Trafic Control Protocol */
201                  case 9000:
202                          priority = 8; /* out of band */
203                  /* Interactive Burst */
204                  case 23:        /* Telnet */
205                  case 177: /* X */
206                  case 2049:      /* NFS */
207                  case 2232:      /* IVS Video */
208                  case 2241:      /* IVS daemon */
209                  case 3456:      /* VAT */
210                  case 4444:      /* NV - same of krb524 (BUG?) */
211                  case 7100:      /* X */
212                          priority = 7;
213                          break;
214                  /* Interactive Bulk Transfer */
215                  case 20:  /* FTP */
216                  case 21:  /* FTP */
217                          priority = 6;
218                          break;
219                  /* Asynchronous Bulk Transfer */
220                  case 25:  /* Mail */
221                  case 4557:       /* Fax */
222                          priority = 5;
223                          break;
224          }
225          return(priority);
226  }
```

port_classify()

Figura 5.22 - Definição das prioridades das aplicações

194-195 A função recebe como parâmetro a estrutura com as portas.

197 Como variável local há somente a prioridade (priority).

199-224 Verifica o número da porta, classificando-a conforme o tipo de aplicação. O protocolo DCRP recebe a prioridade 8 (oito), para ser enviado sem escalonamento.

225 Retorna a prioridade.

Capítulo 6
Escalonamento de Pacotes

Quando vários fluxos de dados compartilham uma mesma interface de rede, deve haver um mecanismo capaz de separar cada um destes fluxos e garantir sua transmissão com a qualidade de serviço solicitada. Este mecanismo é conhecido como Escalonador de Pacotes.

O modelo aqui proposto, procura atender dois tipos de enlaces diferentes: os enlaces ponto-a-ponto, como o SLIP (*Serial Line Internet protocol*) [Rom88] e o PPP (*Point-to-Point Protocol*) [Sim93]; e as redes Ethernet.

Existem diversos mecanismos de escalonamento. Entre estes mecanismos está o CSZ [CSZ92]. Ele parte do pressuposto que diversas aplicações possuem uma certa flexibilidade com relação ao atraso. Com isso, o escalonador pode "atrasar" alguns pacotes ao invés de transmiti-los imediatamente e em seu lugar colocar outros cuja necessidade de transmissão seja imediata. Para isso é desenvolvido um algoritmo chamado WFQ (*Weighted version of the Fair Queuing algorithm*).

Em nossa proposição não há pretensão de se implementar um mecanismo como CSZ, ou qualquer outro mecanismo que tenha uma grande complexidade. O mecanismo para o escalonamento de pacotes (*Dispatch*) aqui proposto é bastante simples e se baseia em filas FIFO (*First In First Out*), que são verificadas continuamente (Seção 6.3).

O envio dos pacotes para escalonamento é feito após a Classificação dos Pacotes (Capítulo 5). Isto deve ocorrer tanto para os fluxos com reservas, quanto para o tráfego *best-effort.*

A forma de tratamento dos dois tipos de tráfego após a classificação é diferente e requer cuidados especiais para o tráfego *best-effort.* A Seção 6.1 explica como são construídas as filas de escalonamento para os fluxos com reserva de recursos. A Seção 6.2 mostra como as filas são formadas para o tráfego *best-effort.* E finalmente, a Seção 6.3 explica como funciona o mecanismo de transmissão dos pacotes nas filas.

A implementação do mecanismo de Escalonamento de Pacotes implica na adaptação do IP para a inclusão do *Dispatch* no *kernel* do Sistema Operacional. Na Tabela 6.1 estão os arquivos alterados ou criados para que o IP funcione com o Escalonador de Pacotes.

Arquivo	Função
/sys/net/ac.h	Arquivo com as estruturas e definições do mecanismo de Controle de Tráfego.
/sys/net/isps.h	Arquivo com as estruturas e definições da interface com o Controle de Tráfego.
/sys/kern/init_main.c	Rotinas de inicialização do *kernel.*
/sys/net/cl_isps.c	Rotinas usadas na Classificação de Pacotes.
/sys/net/sc_isps.c	Rotinas usadas pelo Escalonador de Pacotes

Tabela 6.1 - Arquivos referenciados pelo Escalonador de Pacotes

6.1 Filas de Escalonamento de Reservas

Cada reserva de recursos possui sua própria fila de pacotes. Nela são colocados todos os pacotes pertencentes ao fluxo classificado. Cada fila possui um tamanho máximo. Este tamanho é medido em *bytes* armazenados e é calculado pelo Controle de Admissão (seção 4.2.5).

6.1.1 Policiamento

Quando uma aplicação transmite mais do que o reservado ou ocorre um congestionamento, o Escalonador de Pacotes aciona um mecanismo de Policiamento. O mecanismo de Policiamento deve ser definido pela aplicação ou pelo RSVP durante o processo de execução da reserva. Porém, ainda não há uma padronização neste sentido. Por isso, é proposto um mecanismo próprio.

Nas aplicações de tempo real, é desejável que o Controle de Tráfego identifique os pacotes prioritários quando há a necessidade de descarte de pacotes [WHD94]. Porém, quando isso não é possível, é preferível o descarte dos pacotes mais antigos do que recebe-los com um atraso muito grande [Alb96]. No mecanismo proposto, se o tamanho da fila extrapolar o máximo permitido, o pacote mais antigo armazenado na fila é descartado, liberando espaço para o novo pacote que acabou de chegar.

Como ainda não há códigos de erro próprios para o Controle de Tráfego, na ocasião do descarte de pacotes é atribuído um código de erro indicando que a interface está cheia.

6.1.2 Estruturas das Filas de Reserva

São usadas duas estruturas para o escalonamento. A estrutura sc_buf (Figura 6.1) é o *buffer* onde são colocados os pacotes após a classificação. A estrutura sc_queue (Figura 6.2) é colocada na reserva de recursos e contém além do sc_buf, os campos que armazenarão as informações necessárias para o escalonamento.

ac.h

```
49  typedef struct sc_buf {
50          struct sc_buf   *sc_next;/* next package */
51          struct mbuf     *sc_m;          /* mbuffer */
52          int             sc_len;         /* data length */
53  } sc_isps_buf;
```

Figura 6.1 - Estrutura de *buffers* para reservas

50 Aponta para o próximo pacote na fila.

51 Aponta para a estrutura mbuf, que contém o pacote.

52 Contém o tamanho do pacote, em *bytes*.

ac.h

```
59  struct sc_queue {
60          struct sc_buf       *qu_head;       /* output queue head */
61          struct sc_buf       *qu_tail; /* output queue tail */
62          u_int32_t           qu_len;         /* queue length */
63          u_int32_t           qu_max_len;     /* queue maximum length */
64          struct timeval      qu_time;        /* queue init time */
65          u_int32_t           qu_bytes;/* transmited bytes after init */
66          struct sockaddr     qu_dst;         /* destination */
67          struct rtentry      *qu_rt;         /* destination route */
68  };
```

Figura 6.2 - Estrutura de fila para reservas

60-61 Aponta, respectivamente, para o início e para o final da lista de pacotes aguardando transmissão.

62 Armazena o tamanho da fila de pacotes. Este tamanho corresponde ao total de bytes armazenados em todos os pacotes na fila.

63 Tamanho máximo que a fila pode atingir.

64 Armazena o horário de inicialização da fila. De tempos em tempos a fila é inicializada (Seção 6.3).

65 Quantidade de bytes transmitidos desde a última inicialização da fila.

66 Endereço do destino, definido pelo algoritmo de roteamento.

67 Rota do destino.

6.1.3 Colocação dos pacotes nas filas

O Classificador de Pacotes após verificar a qual reserva o pacote pertence, coloca o mesmo na fila apontada na reserva. Isso é feito pela função **cl_enque**(), apresentada na Figura **6.3.**

299-302 A rotina **cl_enque**() recebe como parâmetros o pacote (m), a rota (rt) e o tamanho do pacote (iplen).

304-308 São usadas como variáveis internas, três apontadores para a fila de pacotes (head, tail e buf), um apontador para o pacote (m0), um apontador para a estrutura da interface usada pelo Controle de Tráfego (acp), e dois inteiros (error e s).

310-311 Atribui às variáveis head e tail o início e o final da fila de pacotes, respectivamente.

313-321 Se o tamanho do *buffer* for maior do que o tamanho máximo permitido, o primeiro pacote do *buffer* é descartado e o tamanho recalculado. Um código de erro é atribuído para retornar aos protocolos superiores.

322-334 Inclui o novo pacote no *buffer*.

335 Atribui à variável usada para o *wakeup*, o endereço da interface onde foi colocado o novo pacote.

336-337 Altera a prioridade do processo para ativar o *wakeup*, que irá avisar ao escalonador que há um novo pacote na fila, repassando o endereço da interface junto.

339 Retorna algum erro, se houver.

cl_isps.c

```
299  int cl_enque(m, rtp, iplen)
300  struct mbuf *m;
301  struct RES_table *rtp;
302  short iplen;
303  {
304  struct sc_buf *head, *tail, *buf;
305  struct mbuf *m0;
306  struct AC_obj *acp;
307  int error = 0;
308  int s;
309
310          head = rtp->res_queue.qu_head;
311          tail = rtp->res_queue.qu_tail;
312          /* if buffer is full, drop the oldest package */
313          if (rtp->res_queue.qu_len >= rtp->res_queue.qu_max_len) {
314                  rtp->res_queue.qu_head = head->sc_next;
315                  rtp->res_queue.qu_len -= head->sc_len;
316                  m0 = head->sc_m;
317                  if (m0)
318                          m_freem(m0);
319                  free(head, M_IFADDR);
320                  error = ENOBUFS;
321          }
322          buf = malloc(sizeof(struct sc_buf), M_IFADDR, M_WAITOK);
323          if (!buf)
324                  return(ENOMEM);          /* new error code must be made */
325          (buf)->sc_next = NULL;
326          buf->sc_m = m;
327          buf->sc_len = iplen;
328          rtp->res_queue.qu_len += iplen;
329          if (!rtp->res_queue.qu_head)
330                  rtp->res_queue.qu_head = buf;
331          else
332                  (tail)->sc_next = buf;
333          rtp->res_queue.qu_tail = buf;
334          acp = rtp->res_obj;
335          sc_wakeup = (u_int32_t)acp;
336          s = splimp();
337          wakeup((caddr_t)&sc_wakeup);
338          splx(s);
339          return(error);
340  }
```

cl_enque()

Figura 6.3 - Enfileiramento de pacotes com reserva

6.2 Filas de Escalonamento do Tráfego *Best-effort*

O escalonamento de pacotes *best-effort* é bastante diferente do escalonamento dos fluxos com reservas. Como visto no Capítulo 4, o Controle de Admissão define no momento da sua inicialização uma banda mínima para o tráfego *best-effort* e um tamanho de *buffer* para comportar atrasos de até 1 (um) segundo. Como a banda reservada inicialmente é muito pequena para comportar as rajadas típicas deste tipo de tráfego, foi proposto um mecanismo de reavaliação da banda reservada, denominado de **Buffer Elástico**.

Toda vez que o *buffer* estiver cheio ou for menor que um mínimo indicado, é chamada a rotina **calc_datagram_rate**(), que irá recalcular a banda necessária para transmissão. O cálculo de banda utiliza os mesmos mecanismos do Controle de Admissão, podendo aumentar ou diminuir o montante da reserva. Se houver recursos disponíveis e que não prejudique os demais fluxos, o montante da reserva é modificado.

A reserva de banda cresce e diminui por degraus de 32 kbits/s (trinta e dois kilobits por segundo). A taxa inicial é definida como menor banda possível para reservas na implementação do RSVP [ISI96]. Este valor é o mesmo que foi reservado inicialmente e é suficiente para que diversas máquinas compartilhem a mesma rede, não permitindo que uma única máquina monopolize toda a rede.

Quando não houver a possibilidade de reservar mais banda, o *buffer* irá crescer até atingir um limite calculado pelo Controle de Admissão. A partir deste limite, todos os novos pacotes que chegarem serão descartados. Isto é diferente do que ocorre nas filas com reservas, onde os pacotes mais antigos são descartados. Pois, aqui, não há a necessidade de controlar o atraso dos pacotes, apesar disso ser desejável. Além disso, as aplicações e protocolos orientados a conexão receberiam mensagens de erro indicando que o pacote foi descartado e ele na verdade foi corretamente colocado na fila do escalonador. O pacote descartado seria o mais antigo e não o que a aplicação tentou enviar.

O tráfego *best-effort* utiliza não uma, mas oito filas de escalonamento. Cada fila tem uma prioridade diferente e o pacote será colocado na fila selecionada pelo Classificador de Pacotes (seção 6.3). Como no fluxo com reservas, somente um pacote será transmitido a cada ciclo do mecanismo de *dispatch*. O pacote transmitido será aquele com maior prioridade, entre os pacotes que se encontrarem nas filas.

6.2.1 Estruturas das Filas *best-effort*

A estrutura datagram_queue (Figura 6.4) comporta o *buffer* de cada fila e é composta de dois ponteiros para a estrutura de *buffer* datagram_buffer (Figura 6.5), onde é armazenado cada pacote e as informações para a transmissão.

ac.h

```
108  struct datagram_queue {
109          struct datagram_buffer    *dq_head; /* queue head pointer */
110          struct datagram_buffer    *dq_tail;  /* queue tail pointer */
111  };
```

Figura 6.4 - Fila do tráfego *best-effort*

109-110 Aponta respectivamente para o início e o final da lista de pacotes aguardando transmissão.

ac.h

```
96   struct datagram_buffer {
97           struct datagram_buffer    *d_next; /* next occurrence */
98           struct mbuf               *d_m;    /* mbuffer */
99           struct sockaddr           d_dst;   /* destination */
100          struct rtentry            *d_rt;   /* destination route */
101          int                       d_len;   /* datagram length */
102  };
```

Figura 6.5 - *Buffer* do tráfego *best-effort*

97 Aponta para a o próximo pacote na fila.

98 Aponta para o pacote.

99 Endereço do destino, definido pelo algoritmo de roteamento.

100 Rota do destino.

101 Tamanho do pacote.

Todas as filas de prioridades são colocadas na estrutura d_table (Figura 6.6), que é inicializada pelo DCRP, através do Controle de Admissão (Seção 4.3.3). No momento da inicialização são atribuídos valores para as taxas de transmissão e tamanho das filas. Esta estrutura é incluída na estrutura de identificação das interfaces AC_obj.

ac.h

```
117  struct d_table {
118          struct datagram_queue dt_queue[DATAGRAM_P_MAX]; /* priorities table */
119          u_int32_t       dt_len;         /* total queue length */
120          struct timeval  dt_time;        /* init time */
121          u_int32_t       dt_bytes; /* transmited bytes after init */
122          u_int32_t       dt_min_len;     /* minimum datagram queue length */
123          u_int32_t       dt_max_len;     /* maximum datagram queue length */
124          u_int32_t       dt_min_rate;    /* datagram minimum rate */
125          u_int32_t       dt_rate;        /* datagram maximum rate */
126          u_int32_t       dt_max_buf;     /* datagram maximum queue length */
127  };
```

Figura 6.6 - Estrutura para informações sobre o tráfego *best-effort*

118 Armazena as filas de escalonamento.

119 Armazena a quantidade de *bytes* em todas as filas do tráfego *best-effort*.

120 Armazena o horário de inicialização da estrutura.

121 Quantidade de *bytes* transmitidos desde a última inicialização da estrutura.

122 Menor tamanho do *buffer* para que seja solicitada a redução da reserva.

123 Maior tamanho do *buffer* para que seja solicitada a ampliação da reserva.

124 Menor taxa reservada para o tráfego *best-effort*. Esta taxa também é usada como degrau para o aumento da reserva.

125 Taxa máxima reservada para o tráfego *best-effort*. Atualmente é limitado pela taxa máxima de transmissão na interface.

126 Tamanho máximo do *buffer* antes que seja feito o descarte.

6.2.2 Colocação dos pacotes nas filas

O Classificador de Pacotes após verificar qual a prioridade do pacote, coloca o mesmo na fila da prioridade. Isso é feito pela função **cl_datagram**(). A Figura 6.7 apresenta resumidamente a função (apenas alguns trechos da rotina são apresentados para facilitar a visualização).

125-131 A rotina **cl_datagram**() recebe como parâmetros a prioridade do pacote (priority), o apontador para a interface de saída (ifp), o pacote (m), o endereço de destino selecionado pelo roteamento (dst), a rota de destino (rt) e o tamanho do pacote.

133-137 As variáveis locais são: três apontadores para a estrutura de *buffer* (dtb, dt_head e dt_tail); o apontador para a interface do Controle de Tráfego (acp), uma *string* para armazenar o nome da interface (name); um inteiro de 32 bits

para cálculo do tamanho do *buffer* (new_len); e dois inteiros para priorização de processos e erros (s e error).

cl_isps.c

```
125  int cl_datagram(priority, ifp, m, dst, rt, iplen)
126  int priority;
127  register struct ifnet *ifp;
128  struct mbuf *m;
129  struct sockaddr *dst;
130  struct rtentry *rt;
131  short iplen;
132  {
133  struct datagram_buffer        *dtb, *dt_head, *dt_tail;
134  struct AC_obj *acp;
135  char name[IFNAMSIZ];
136  u_int32_t new_len;
137  int s, error = 0;
140          sprintf(name,"%s%d", ifp->if_name, ifp->if_unit);
141          acp = ac_find(name);
142          if (!acp)
148            if ((!strcmp(ifp->if_name,"lo")) || (!strcmp(ifp->if_name,"tun")))
149                  error = (*ifp->if_output)(ifp, m, (struct sockaddr *)dst, rt);
150            else
151                  error = ENETUNREACH;
152          else {
153            if (acp->ac_flags & IS_AC_ACTIVE) {
154                  new_len = acp->ac_datagram.dt_len + iplen;
155                  if (new_len <= acp->ac_datagram.dt_max_buf) {
156                    dtb = malloc(sizeof(struct datagram_buffer), M_IFADDR, M_WAITOK);
157                    if (!dtb)
158                      error = ENOMEM;
159                    else {
160                      (dtb)->d_next = NULL;
161                      dtb->d_m = m;
162                      dtb->d_dst = *dst;
163                      dtb->d_rt = rt;
164                      dtb->d_len = iplen;
165                      if ((new_len > acp->ac_datagram.dt_max_len) ||
166                          (new_len < acp->ac_datagram.dt_min_len))
167                              calc_datagram_rate(acp);
169                      acp->ac_datagram.dt_len += dtb->d_len;
170                      dt_head = acp->ac_datagram.dt_queue[priority].dq_head;
171                      dt_tail = acp->ac_datagram.dt_queue[priority].dq_tail;
172                      if(!dt_head)
173                              acp->ac_datagram.dt_queue[priority].dq_head = dtb;
174                      else
175                              (dt_tail)->d_next = dtb;
176                      acp->ac_datagram.dt_queue[priority].dq_tail = dtb;
177                      sc_wakeup = (u_int32_t)acp;
178                      s = splimp();
179                      wakeup((caddr_t)&sc_wakeup);
180                      splx(s);
181                    }
182                  } else                  error = ENOBUFS;
184            } else              error = ENETUNREACH;
186          }
187          return(error);
188  }
```

cl_datagram()

Figura 6.7 - Enfileiramento de pacotes *best-effort*

140-141	Atribui a name o nome da interface, para procura da mesma na lista de interfaces do Controle de Tráfego.
142-151	Se não achou a interface, verifica se ela é *loopback* ou *tunnel*. Se for, envia diretamente, sem escalonamento. Caso contrário, retorna erro na rede.
152-153	Verifica se o Controle de Tráfego está ativo.
154-155	Calcula o novo tamanho do *buffer* e se for menor que o máximo permitido, inclui o pacote na fila.
156-159	Aloca espaço de memória para armazenar o *buffer*.
160-164	Atribui aos campos do *buffer* as informações sobre o pacote.
165-167	Verifica o tamanho do *buffer*, para solicitar o recalculo da reserva (Figura 6.8).
169	Recalcula o tamanho do *buffer*.
170-176	Coloca o *buffer* na fila do escalonador.
178	Atribui à variável usada para o *wakeup*, o endereço da interface onde foi colocado o novo pacote.
179-180	Altera a prioridade do processo para ativar o *wakeup* que irá avisar ao escalonador que há um novo pacote no *buffer*, repassando o endereço da interface junto.
182	Se o máximo do *buffer* atingiu seu limite, retorna erro de falta de *buffer* na interface.
183	Se no Controle de Tráfego, a interface constar como desativada, retorna erro na rede.
187	Retorna algum erro, se houver.

6.2.3 Reavaliação do tráfego *best-effort*

Sempre que o tamanho do *buffer* for maior que o *buffer* máximo para que o tráfego obtenha um atraso de até 1 (um) segundo e toda vez que um pacote é enviado pelo escalonador, é solicitada uma reavaliação da reserva para o tráfego *best-effort*. Esta reavaliação é executada pela função **calc_datagram_rate**() (Figura 6.8). A rotina irá verificar a disponibilidade de recursos, da mesma forma como é feito pelo Controle de Admissão (Figura 4.2.5). Se houver recursos disponíveis e se isso não for prejudicial às demais reservas, é alterada a reserva.

ac_isps.c

```
87    int calc_datagram_rate(acp)
88    struct AC_obj *acp;
89    {
90    u_int32_t new_rate, new_delay;
91    int32_t datagram_rate;
92    u_int32_t t_rate = acp->ac_tot_flow.res_rate;
93    int32_t   s_rate = 0;
94
95              if (acp->ac_datagram.dt_len > acp->ac_datagram.dt_max_len)
96                s_rate = acp->ac_datagram.dt_min_rate;
97              else
98                if (acp->ac_datagram.dt_len < acp->ac_datagram.dt_min_len)
99                        s_rate = -acp->ac_datagram.dt_min_rate;
100             datagram_rate = acp->ac_datagram.dt_rate + s_rate;
101             if (datagram_rate < acp->ac_datagram.dt_min_rate )
102               return(0);
103             new_rate = t_rate + s_rate;
104             if (new_rate > acp->ac_max_rate)
105               return(-1);
106             if (t_rate < ISPS_FLOW_MIN_RATE)
107               t_rate = ISPS_FLOW_MIN_RATE;
108             new_delay = (acp->ac_avg_delay * new_rate) / t_rate;
109             if (new_delay > acp->ac_tot_flow.res_delay)
110               return(-1);
111             acp->ac_datagram.dt_rate = datagram_rate;
112             acp->ac_tot_flow.res_rate = acp->ac_tot_flow.res_rate + s_rate;
113             acp->ac_loc_flow.res_rate = acp->ac_loc_flow.res_rate + s_rate;
114             acp->ac_datagram.dt_max_len = acp->ac_datagram.dt_rate / 8;
115             acp->ac_datagram.dt_min_len = 1;
116             acp->ac_datagram.dt_max_buf = (((u_int64_t)acp->ac_datagram.dt_rate / 8)
117                                             * ISPS_CLASS_MAX_DELAY) / 1000;
118             return(0);
119   }
```

calc_datagram_rate()

Figura 6.8 - Reavaliação da reserva para o tráfego *best-effort*

87-88 A função **calc_datagram_rate()** recebe como parâmetro o apontador para a estrutura de interface do Controle de Tráfego.

90-93 Para uso interno são definidas cinco variáveis inteiras de 32 bits.

95-96 Se o tamanho do *buffer* for maior que o máximo para o recalculo, atribui a s_rate o valor do degrau de reserva.

97-99 Se o tamanho do *buffer* for menor que o mínimo para redução da reserva, atribui a s_rate o valor negativo do degrau de reserva.

100 Calcula o novo valor da reserva.

101-102 Se o valor calculado for menor que o mínimo, retorna à função original.

103-105 Se o valor calculado for maior que a maior taxa permitida na rede, retorna um erro.

106-107 Se o valor total reservado na rede for menor que a menor reserva permitida na rede, atribui a menor reserva. Isto somente ocorrerá se houver algum dano nas estruturas de reserva.

108-110	Calcula o novo atraso, baseado nos cálculos de previsão de atraso (Seção 4.1). Se a previsão de atraso for maior que o menor atraso reservado na rede, rejeita a reserva.
111-113	Recalcula as taxas de reserva para datagrama, local e total.
114	Recalcula o tamanho máximo do *buffer* para recalculo.
115	Atribui 1 (um) ao tamanho mínimo do *buffer* para recalculo.
116	Recalcula o tamanho máximo do *buffer* para descarte.

6.3 Mecanismo de Transmissão dos Pacotes (*Dispatch*)

Após a colocação dos pacotes nas filas de escalonamento, deve haver um mecanismo que percorra estas filas, transmitindo os pacotes de acordo com a reserva solicitada. O mecanismo deve ser capaz de transmitir tanto os fluxos com reservas, quanto o tráfego *best-effort*.

O mecanismo aqui proposto visa uma implementação simplificada e consiste basicamente de múltiplas filas FIFO. As filas são verificadas de forma circular, tentando manter uma certa eqüidade na transmissão, de forma que cada fluxo seja independente dos demais. Além dos fluxos reservados, o algoritmo também suporta o tráfego *best-effort*, que é transmitido usando prioridades, com uma banda previamente reservada. A Figura 6.9 ilustra o funcionamento do modelo.

Para cada interface de saída, o Escalonador executa a varredura nas filas de reservas, verificando em uma fila de por vez. O pacote somente será transmitido se a atual taxa de transmissão do fluxo não tiver extrapolado o montante reservado. Isso impede que um fluxo tenha maior prioridade, prejudicando os demais.

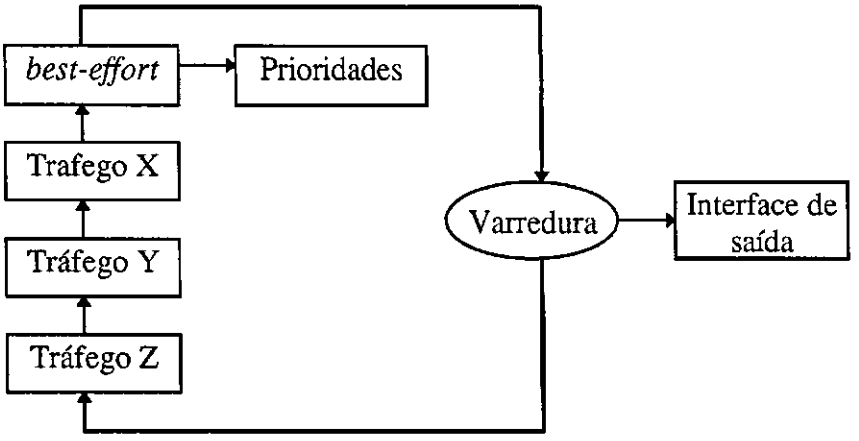
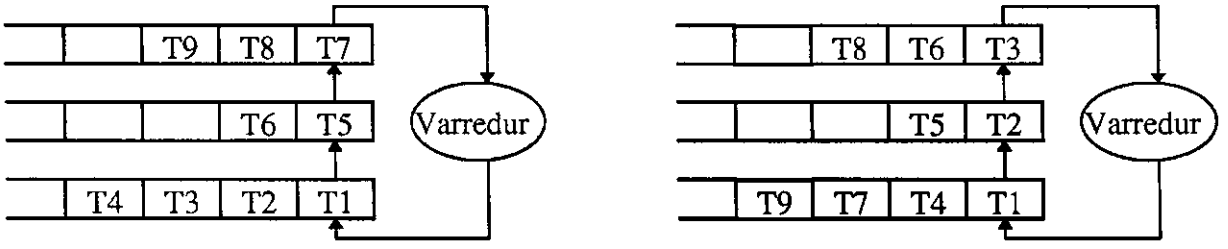


Figura 6.9 - Mecanismo de *Dispatch*

Ao invés de retirar de cada fila, esvaziando o *buffer* até atingir a taxa reservada, optou-se por transmitir um pacote de cada vez, em cada fila. O atraso médio durante a transmissão de todos os pacotes na fila, em ambos os casos é equivalente. Porém, o atraso máximo esperado de um pacote em cada fila é melhor no primeiro caso. A Figura 6.10 mostra os dois esquemas.



Esvaziamento do *buffer* Um pacote de cada vez

Figura 6.10 - Exemplo de filas de escalonamento

No exemplo, temos para o primeiro caso, um tempo total de 9 UT (Unidades de Tempo) para transmissão total dos pacotes nas filas. A primeira fila inicia sua transmissão em 1 UT, a segunda fila em 5 UT e a terceira fila em 7 UT. Neste caso, o menor tempo de transmissão do primeiro pacote na fila seria de 1 UT e o maior de 7 UT.

No segundo caso, a primeira fila inicia sua transmissão em 1 UT, a segunda fila em 2 UT e a terceira fila em 3 UT. Com isso, o menor tempo de transmissão do primeiro pacote na fila seria de 1 UT e o maior de 3 UT. Assim, temos no segundo caso, um tempo máximo de transmissão de cada pacote na fila de 3 UT, contra 7 UT do primeiro caso.

O mecanismo de cálculo das taxas de transmissão implementado utiliza valores reais para seus relógios e pode ser aprimorado por novas versões da implementação. Contudo, apesar da sua simplicidade, atende às necessidades (Capítulo 7). A implementação está aberta a novos mecanismos, que podem ser desenvolvidos em substituição à implementação atual.

6.3.1 Adaptação do Sistema Operacional

O mecanismo de *Dispatch* deve funcionar continuamente de forma independente dos demais processos. A adaptação do Sistema Operacional para comportar este mecanismo exige que se altere a função **main()** do sistema (Figura 6.11), onde são inicializados os processos. Lá será incluída a chamada do processo **sc_init()** que irá executar a transmissão dos pacotes.

init_main.c

```
363             if (fork(p, (void *) NULL, rval))
364                     panic("failed fork update daemon");
365             if (rval[1]) {
366                     p = curproc;
367                     updateproc = p;
368                     p->p_flag |= P_INMEM | P_SYSTEM;
369                     bcopy("sc_init", p->p_comm, sizeof("sc_init"));
370                     sc_init();
371                     /*NOTREACHED*/
372             }
```

main()

Figura 6.11 - Adaptação do *kernel* para o *Dispatch*

363-372 Cria um processo "filho" chamado sc_init, que executará a função **sc_init()**.

A seguir, será mostrado como é implementado o mecanismo de *Dispatch* e as rotinas que ele utiliza.

sc_init()

A rotina **sc_init()** é a responsável por verificar as taxas de transmissão de cada fila e transmitir os pacotes para a interface de rede, sempre que possível. O Sistema Operacional utiliza uma variável de tempo time para armazenar a hora do sistema. Esta variável é atualizada a cada 10 milissegundos. Este valor é alto para a transmissão de pacotes. Por isso, optou-se pelo cálculo da taxa de transmissão por degraus de 10 milissegundos. Ou seja, transmite-se

dados até atingir a taxa possível de ser transmitida neste tempo. Quando o número de *bytes* transmitidos chega a este limite, cessa a transmissão da fila, até que o relógio seja atualizado.

sc_isps.c

```
172   void sc_init()
173   {
174   register struct RES_table *rtp;
175   struct AC_obj *acp = AC_START;
176   struct timeval now;
177   u_int32_t delta;
178   u_int64_t prevision=0;
179   int sc_timo = 1;
180   int pkg_flag=0;
181   int s;
182   int error=0;
183
184       splnet();
185       while(1) {
186               if (!acp) {
187                   s = splimp();
188                   error = tsleep((caddr_t)&sc_wakeup,
189                                   PRIBIO|PCATCH, "sc_init", sc_timo);
190                   splx(s);
191                   if (!error)
192                           acp = (struct AC_obj *)sc_wakeup;
193                   else
194                           acp = AC_START;
195                   sc_wakeup = NULL;
196               }
197               pkg_flag = 0;
198               while (acp) {
199                   acp->ac_flags |= IS_SC_ACTIVE;
200                   acp->ac_flags &= ~IS_SC_PACKET;
201                   rtp = acp->ac_list_head;
202                   while (rtp) {
                                  Figura 6.13
230                   }
231                   error = sc_datagram(acp);
232                   if (error)
233                           break;
234                   if (acp->ac_flags & IS_SC_PACKET)
235                           pkg_flag++;
236                   acp = acp->ac_next_interface;
237                   if ((!acp) && pkg_flag) {
238                           acp = AC_START;
239                           pkg_flag = 0;
240                   }
241               }
242       }
243   }
```

sc_init()

Figura 6.12 - Implementação do *Dispatch*

174-182 A rotina **sc_init()** usa como variáveis: um apontador para a reserva (rtp); um apontador para a interface do Controle de Tráfego (acp); uma estrutura de tempo (now), uma variável inteira de 32 bits para cálculo de tempo (delta); uma variável inteira de 64 bits para cálculo de transmissão (prevision); e quatro

variáveis inteiras para interrupção de espera (sc_timo), controle de escalonamento (pkg_flag), prioridade de processo (s) e erro (error).

184 Muda a prioridade do processo para **splnet().**

185 Inicia o *loop* sem fim para verificação das interfaces.

186-189 Se nenhuma interface for especificada, altera a prioridade do processo para **splimp**(). Coloca o processo em suspensão até que um novo pacote seja escalonado ou se passe um ciclo do relógio. A variável sc_timo define um ciclo de relógio, que corresponde a 10 milissegundos no equipamento utilizado.

190 Retorna a prioridade original do processo.

191-194 Se não ocorrer nenhum erro na volta ao processo, é repassada a interface na qual o pacote foi colocado, que será utilizada na procura por pacotes.

197 É inicializada a variável pkg_flag. Ela é utilizada na verificação de pendências de pacotes para transmissão.

198 Executa a varredura das interfaces a procura de pacotes para transmissão.

199-200 Ativa os *flags* da interface, indicando que o escalonador está ativo e não há pacotes pendentes.

201-230 Executa o *Dispatch* para as reservas (Figura 6.13).

231-233 Executa o *Dispatch* para o tráfego *best-effort* (Figura 6.15).

234-235 Se houverem pacotes pendentes na interface, ativa a variável pkg_flags.

236 Aponta para a próxima interface.

237-240 Se for o final da lista de interfaces e a variável pkg_flags estiver ativa, indicando que há pacotes pendentes, chama a primeira interface novamente.

6.3.2 *Dispatch* das Reservas

O mecanismo de *Dispatch* das reservas percorrerá todas as reservas, procurando as filas que possuam pacotes pendentes e cuja taxa de transmissão seja menor do que a reservada. A Figura 6.13 ilustra o mecanismo.

sc_isps.c

```
201         rtp = acp->ac_list_head;
202         while (rtp) {
203                 if (rtp->res_queue.qu_head) {
204                   now = time;
205                   delta = (now.tv_sec - rtp->res_queue.qu_time.tv_sec)*1000000
206                           +(now.tv_usec - rtp->res_queue.qu_time.tv_usec)
207                           +ISPS_TIME_STEP;
208                   if (delta >= ISPS_SC_TIME) {
209                           delta = ISPS_TIME_STEP;
210                           prevision = (((u_int64_t)ISPS_SC_TIME *
211                                   (u_int64_t)rtp->res_flow.res_rate) / 8000000);
212                           if (acp->ac_datagram.dt_bytes > prevision)
213                             rtp->res_queue.qu_bytes -= prevision;
214                           else
215                             rtp->res_queue.qu_bytes = 0;
216                           rtp->res_queue.qu_time = time;
217                   }
218
219                   if (delta == 0)
220                           delta = ISPS_TIME_STEP;
221                   prevision = (((u_int64_t)delta *
222                                   (u_int64_t) rtp->res_flow.res_rate) / 8000000);
223                   if (prevision > rtp->res_queue.qu_bytes) {
224                           error = sc_dequeue(rtp, acp->ac_ifp);
225                           if (error)
226                                   break;
227                   }
228                 }
229                 rtp = rtp->res_next;
230         }
```

sc_init()

Figura 6.13 - *Dispatch* de pacotes com reserva

201-202 Percorre todas as reservas, a partir da primeira para verificação das filas.

203 Se houver algum pacote na fila, continua os testes.

204-207 Calcula o tempo decorrido desde a última inicialização da fila.

208-217 Se já for o momento de reinicializar a fila, verifica quantos *bytes* deveriam ter sido transmitidos e quantos foram. Se foram transmitidos mais *bytes* do que o permitido, coloca a diferença no número de *bytes* transmitidos, para o próximo ciclo de transmissão. Atribui a hora da inicialização à fila

219-220 Se a diferença de tempo desde a ultima inicialização for zero, atribui o valor do degrau de tempo à diferença. Isso se deve ao fato do degrau ser relativamente alto.

221-227 Calcula quantos *bytes* deveriam ter sido transmitidos desde a ultima inicialização, para estar de acordo com a taxa de transmissão reservada. Se o valor for maior do que o que realmente foi transmitido, solicita que o pacote seja retirado da fila. A retirada da fila é feita pela função **sc_dequeue()** (Figura 6.14).

229 Passa o processamento para a próxima reserva.

sc_dequeue()

A função **sc_dequeue()** envia o primeiro pacote na fila da reserva e atualiza os campos da fila. A Figura 6.14 mostra sua implementação.

sc_isps.c

```
143  int sc_dequeue(rtp, ifp)
144  struct RES_table *rtp;
145  register struct ifnet          *ifp;
146  {
147  struct sc_buf *buf;
148  register struct sockaddr *dst = &rtp->res_queue.qu_dst;
149  register struct rtentry        *rt = rtp->res_queue.qu_rt;
150  struct mbuf *m;
151  int error=0;
152
153          buf = rtp->res_queue.qu_head;
154          if (buf){
155                  m = buf->sc_m;
156                  error = (*ifp->if_output)(ifp, m, (struct sockaddr *)dst, rt);
157                  rtp->res_queue.qu_bytes += buf->sc_len;
158                  rtp->res_queue.qu_head = buf->sc_next;
159                  rtp->res_queue.qu_len -= buf->sc_len;
160                  free(buf, M_IFADDR);
161                  if (rtp->res_queue.qu_head)
162                          (rtp)->res_obj->ac_flags |= IS_SC_PACKET;
163          }
164          return(error);
165  }
```

sc_dequeue()

Figura 6.14 - Retirada do pacote com reserva da fila de escalonamento

143-145 A função **sc_dequeue()** recebe como parâmetros o apontador para a reserva (rtp) e o apontador para a interface de saída (ifp).

147-151 As variáveis internas são: um apontador para a estrutura de *buffer* da fila (buf); o endereço de destino armazenado na reserva (dst); a rota armazenada na reserva (rt); o apontador de pacote (m); e uma variável inteira para erros.

153-154 Atribui a buf o *buffer* enfileirado e se houver valor, continua o processamento.

155 Atribui a m o pacote contido na estrutura de *buffer*.

156 Envia o pacote para a interface de rede.

157-169 Atualiza os valores referentes ao número de *bytes* transmitidos, o tamanho da fila e aponta para o próximo pacote.

160 Libera o *buffer* transmitido da memória.

161-162 Se houver algum pacote na fila da reserva, ativa o *flag* que indica a pendência de transmissão na interface.

sc_datagram()

A função **sc_datagram()** (Figura 6.15) faz a busca de pacotes entre as filas do tráfego *best-effort,* a partir da maior prioridade, até a menor. Quando um pacote é encontrado e há banda para transmiti-lo, é solicitada a retirada da fila e o processamento retorna à função **sc_init()**.

sc_isps.c

```
104   int sc_datagram(acp)
105   struct AC_obj *acp;
106   {
107   u_int32_t delta;
108   u_int64_t prevision;
109   struct timeval now;
110   int priority;
111   int error = 0;
112
113           for(priority = 7; priority >= 0; priority--){
114             if (acp->ac_datagram.dt_queue[priority].dq_head) {
115                     now = time;
116                     delta = (now.tv_sec - acp->ac_datagram.dt_time.tv_sec)*1000000
117                         +(now.tv_usec - acp->ac_datagram.dt_time.tv_usec)
118                         +ISPS_TIME_STEP;
119                     if (delta > ISPS_SC_TIME) {
120                       acp->ac_datagram.dt_time = time;
121                       delta = ISPS_TIME_STEP;
122                       prevision = (((u_int64_t)ISPS_SC_TIME *
123                                       (u_int64_t)acp->ac_datagram.dt_rate) / 8000000);
124                       if (acp->ac_datagram.dt_bytes > prevision)
125                               acp->ac_datagram.dt_bytes -= prevision;
126                       else
127                               acp->ac_datagram.dt_bytes = 0;
128                     }
129                     prevision = (((u_int64_t)delta *
130                                   (u_int64_t)acp->ac_datagram.dt_rate) / 8000000);
131                     if (prevision > acp->ac_datagram.dt_bytes)
132                       error = sc_datagram_dequeue(acp, priority);
133                     break;
134             }
135           }
136           return(error);
137   }
```

sc_datagram()

Figura 6.15 - *Dispatch* dos pacotes *best-effort*

104-105 A função **sc_datagram()** recebe como parâmetro o apontador para a interface do Controle de Tráfego.

107-111 As variáveis internas utilizadas são: uma variável inteira de 32 bits para o cálculo de tempo (delta); uma variável inteira de 32 bits para cálculo da taxa de transmissão (prevision); uma estrutura te tempo (now); e dois inteiros para armazenar a prioridade da fila (priority) e o erro (error).

113 Percorre as listas de prioridades a partir da mais alta (7), até a mais baixa (0).

114 Se houver algum pacote na fila da prioridade, continua o processamento.

115-116 Calcula o tempo decorrido desde a última inicialização da fila.

117-128 Se já for o momento de reinicializar a fila, verifica quantos *bytes* deveriam ter sido transmitidos e quantos foram. Se foram transmitidos transmitidos mais *bytes* do que o permitido, coloca a diferença no número de *bytes* transmitidos para o próximo ciclo de transmissão. Atribui a hora da inicialização à fila.

129-132 Calcula quantos *bytes* deveriam ter sido transmitidos desde a ultima inicialização, para estar de acordo com a taxa de transmissão reservada. Se o valor for maior do que o que realmente foi transmitido, solicita que o pacote seja retirado da fila. A retirada da fila é feita pela função **sc_datagram_dequeue()** (Figura 6.16).

133 Interrompe a busca nas filas e retorna ao **sc_init()** (Figura 6.12).

sc_datagram_dequeue()

A função **sc_datagram_dequeue()** (Figura 6.16) retira da fila indicada pela prioridade o primeiro pacote, enviando-o à interface de saída.

sc_isps.c

```
68    int sc_datagram_dequeue(acp, priority)
69    struct AC_obj *acp;
70    int priority;
71    {
72    struct datagram_buffer *dtb;
73    register struct sockaddr *dst;
74    register struct rtentry          *rt;
75    register struct ifnet *ifp;
76    struct mbuf *m;
77    int prt;
78    int error = 0;
79
80              dtb = acp->ac_datagram.dt_queue[priority].dq_head;
81              ifp = acp->ac_ifp;
82              m = dtb->d_m;
83              dst = &dtb->d_dst;
84              rt = dtb->d_rt;
85              error = (*ifp->if_output)(ifp, m, (struct sockaddr *)dst, rt);
86              acp->ac_datagram.dt_bytes += dtb->d_len;
87              acp->ac_datagram.dt_len -= dtb->d_len;
88              acp->ac_datagram.dt_queue[priority].dq_head = dtb->d_next;
89              for(prt = 7; prt >= 0; prt--)
90                        if (acp->ac_datagram.dt_queue[priority].dq_head){
91                           acp->ac_flags |= IS_SC_PACKET;
92                           break;
93                        }
94              free(dtb, M_IFADDR);
95              if (acp->ac_datagram.dt_len < acp->ac_datagram.dt_min_len)
96                        calc_datagram_rate(acp);
97              return(error);
98    }
```

sc_datagram_dequeue()

Figura 6.16 - Retirada dos pacotes *best-effort* da fila de escalonamento

68-70 A função **sc_datagram_dequeue**() recebe como parâmetros o apontador para a interface do Controle de Tráfego (acp) e a prioridade da fila.

72-78 Como variáveis internas são usadas: uma variável de *buffer* do tráfego *best-effort* (dtb); o endereço de destino do pacote (dst); a rota do destino (rt); o apontador para a interface de saída (ifp); o apontador para o pacote (m); e duas variáveis inteiras para prioridade (prt) e para erro (error).

80 Indica em dtb o *buffer* a ser transmitido.

81-84 Atribui às variáveis locais a interface de saída, o destino, o pacote e a rota.

85 Envia o pacote para a interface de rede.

86-88 Atualiza os valores referentes ao número de *bytes* transmitidos, o tamanho da fila e aponta na reserva para o próximo *buffer*.

89-92 Pesquisa as filas de prioridades até achar algum pacote na fila. Se achar, ativa o *flag* que indica a pendência de transmissão na interface.

94 Libera o *buffer* transmitido da memória.

95-96 Se o tamanho do *buffer* do tráfego *best-effort* for menor do que o mínimo para reavaliação da reserva, solicita a reavaliação da reserva (Figura 6.8).

Capítulo 7
Caracterização

Após a implementação do mecanismo de Controle de Tráfego, foram realizados testes para verificar a eficácia do modelo proposto e suas deficiências. Alguns ajustes necessários e deficiências foram detectadas após uma análise dos resultados finais e estão citadas neste capítulo.

Para a avaliação do modelo foi montada uma pequena rede local Ethernet, com os recursos de hardware disponíveis (Seção 7.1). Para a execução dos testes utilizou-se um software de video-conferência e também uma aplicação que recebe e envia dados com taxa de transmissão constante e utiliza comunicação *multicast* (Seção 7.2). Os testes realizados estão citados na Seção 7.3. A análise dos resultados detectou algumas melhorias que podem ser introduzidas no modelo, bem como as limitações a serem impostas na utilização do mesmo. A Seção 7.4 mostra um resumo dos resultados dos testes.

7.1 Ambiente de Testes

A dificuldade para disponibilização de equipamentos para a realização dos testes dificultou uma avaliação mais extensa do mecanismo proposto. Por isso, apenas dois *hosts* foram utilizados na fase de testes. As máquinas foram conectadas a um *hub* 10baseT, através de suas portas Ethernet, utilizando cabos preparados de fábrica com conectores RJ45 (Figura 7.1).

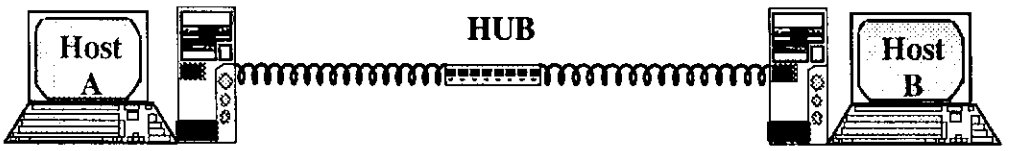


Figura 7.1 - Ambiente de Testes

Os *hosts* utilizados possuem as características a seguir .

***Host* A**

- Hardware

  Microcomputador 486-DX2/66 com processador Intel®, 8 *Mbytes* de memória RAM, um disco rígido de 270 *Mbytes* e outro de 540 *Mbytes* e placa de rede compatível com NE2000®.

- Software

  Sistema Operacional FreeBSD-2.1.0©, configurado para conexão TCP/IP e ambiente de janelas XFree86© versão 12 (padrão X11R6).

***Host* B**

- Hardware

  Microcomputador 486-DX4/100 com processador Intel®, 16 *Mbytes* de memória RAM, um disco rígido de 1 *Gbyte* e placa de rede compatível com NE2000® .

- Software

  Sistema Operacional FreeBSD-2.1.0©, configurado para conexão TCP/IP e ambiente de janelas XFree86© versão 12 (padrão X11R6).

7.2 Aplicações Usadas nos Testes

Para a realização dos testes foram utilizadas duas aplicações. Ambas podem utilizar comunicação *unicast* ou *multicast* e também estão preparadas para efetuar reserva de recursos.

A primeira aplicação é o software **NV**© (*Network Video Tool*) [Fre94] adaptado especialmente para efetuar reserva de recursos através do RSVP. A versão normal do software é distribuída pela Internet em ftp://ftp.parc.xerox.com/pub/net-reserach/nv-3.3beta. Já a versão adaptada vem incluída no pacote RSVP (Seção 3.6).

O **NV** é um software de transmissão de vídeo e pode ser executado em quase todas as plataformas UNIX existentes. A captura de vídeo pode ocorrer diretamente sobre a tela gráfica do computador (padrão X11) ou através de equipamento de captura de vídeo (câmera e placa de captura de vídeo). Existe no software a opção de selecionar durante sua execução a taxa de transmissão que está sendo gerada, até 1024 kbits/s. Porém, para a reserva de recursos, a taxa de reserva é fixada em 128 kbits/s, a taxa de *token bucket,* em 250 kbits/s e o atraso em 100 milissegundos. Como não foi possível a aquisição de equipamento de captura de vídeo, somente testes com a tela foram possíveis.

Como o **NV** não permite alteração na reserva de recursos e a taxa de transmissão é variável, foi necessário o desenvolvimento de uma aplicação que gerasse tráfego a taxas constantes e efetuasse reserva de recursos, contabilizando a transmissão e o atraso. Desenvolveu-se especialmente para estes testes, o **rs_aplic.**

O **rs_aplic** é um software não gráfico que permite ao usuário a simulação de uma transmissão de dados a taxas constantes, com ou sem reserva de recursos. Também pode ser selecionado o tempo de funcionamento, mudança automática de taxa de transmissão e reserva para a execução de testes consecutivos, até um limite determinado. A execução da aplicação segue um modelo cliente-servidor, onde uma máquina está transmitindo e a outra recebendo os dados. O tamanho de mensagem usado pela aplicação é de 1252 *bytes*. Este valor sobe para 1280 *bytes* após a inclusão do cabeçalho IP no *kernel*. O mecanismo de escalonamento utilizado para a geração de pacotes, a taxas constantes, é o mesmo implementado pelo mecanismo de Escalonamento de Pacotes do Controle de Tráfego (Capítulo 6).

O **rs_aplic** ativa um programa para contabilização do atraso através de mensagens ICMP (*Internet Control Message Protocol*) [Ste94]. Este programa é uma adaptação do programa **ping**© nativo do FreeBSD. Além do atraso experimentado em cada *probe*, é calculada a média e os atrasos mínimo e máximo.

7.3 Aplicação dos Teste

Na avaliação do mecanismo foram aplicados testes usando o **NV** e o **rs_aplic**. Para o primeiro foram efetuados apenas testes de funcionamento e variação da taxa de transmissão, com e sem reserva de recursos. Com o segundo foram feitos testes de variação das taxas de transmissão e reserva, com contabilização do atraso na rede.

7.3.1 Testes com o NV

Os testes com o **NV** foram bastante simples e consistiram na captura e transmissão da tela do computador. O software disponibiliza para quem está recebendo a imagem algumas informações de taxa de recepção e perda de pacotes. Os seguintes testes foram realizados:

1. transmissão sem Controle de Tráfego;
2. transmissão com Controle de Tráfego e reserva de recursos; e
3. transmissão com Controle de Tráfego e sem reserva de recursos.

Como a análise de desempenho da aplicação é principalmente visual e o armazenamento dos valores não é possível, não poderão ser apresentados os valores absolutos da análise.

No teste **1**, foi obtida uma taxa máxima de transmissão em torno de 450 kbits/s. Isso se deve ao pesado processamento necessário para a captura de tela.

No teste **2**, a taxa máxima obtida foi de 128 kbits/s. Como a reserva é constante a 128 kbits/s, a partir deste valor o Policiamento no Controle de Tráfego inicia o descarte de pacotes. Por isso, há uma deterioração considerável da qualidade da imagem, com a colocação de espaços vazios. Quanto maior a taxa de transmissão tentada, maior é a perda de qualidade e menor é a taxa recebida.

No teste **3**, a taxa máxima obtida girou em torno de 300 kbits/s. Os ajustes feitos pelo mecanismo de reavaliação do tráfego *best-effort* foram bastante rápidos, acompanhando as variações na taxa de transmissão imposta pelo NV. Porém, o atraso experimentado pelos pacotes foi bastante alto, mesmo utilizando prioridade máxima nos pacotes. Apesar disso, é possível a visualização e o entendimento das imagens pelo receptor, mesmo com atraso, principalmente quando a taxa de transmissão é alta. Ou seja, quanto maior a taxa de

transmissão da aplicação, melhor o ajuste do mecanismo de escalonamento do tráfego *best-effort* para transmitir os dados.

A melhor situação ocorre no segundo teste (com Controle de Tráfego e reserva de recursos), quando a taxa de transmissão da aplicação está ajustada à taxa reservada. A transmissão sem Controle de Tráfego é preferível para as transmissões sem reserva de recursos. Mas, isso não inviabiliza a transmissão com Controle de Tráfego e sem reserva de recursos. Em situações de limite para a carga da rede (levando-se em conta as reservas), o mecanismo de Controle de Tráfego deve possibilitar melhores resultados com relação a atraso e vazão, do que o uso da rede sem Controle de Tráfego.

7.3.2 Testes com o rs_aplic

Com o **rs_aplic** foram feitos os seguintes testes:

1. uma máquina transmitindo, sem escalonamento (Figura 7.2 e Figura 7.3);
2. duas máquinas transmitindo, sem escalonamento (Figura 7.4 e Figura 7.5);
3. uma máquina transmitindo, com escalonamento e reserva de recursos (Figura 7.6 e Figura 7.7);
4. duas máquinas transmitindo, com escalonamento e reserva de recursos (Figura 7.8 e Figura 7.9); e
5. uma máquina transmitindo, com escalonamento e sem reserva de recursos (Figura 7.10 e Figura 7.11).

Em todos os testes foi realizada a variação da carga até o máximo suportado. As variações foram feiras em degraus de 64 kbits/s até atingir 1024 kbits/s e a partir deste ponto, utilizou-se degraus de 128 kbits/s.

Para que o Controle de Tráfego pudesse comportar possíveis variações da taxa do fluxo por parte da aplicação foram utilizados valores de *token-bucket* (Seção 2.1.1) superiores às taxas reservadas. Até 1024 kbits/s foi usado o *token-bucket* com 64 kbits/s acima do reservado e a partir de 1024 kbits/s o valor do parâmetro foi 128 kbits/s acima do reservado. Com isso esperava-se evitar o descarte de pacotes.

Para cada taxa de transmissão solicitada, foi feita a transmissão dos dados durante 70 segundos e a coleta dos atrasos durante 60 segundos, com uma amostra por segundo,

totalizando 60 amostras. Devido a pequena amostra (quantidade de coletas de tempo), o coeficiente de variação do atraso e a covariância entre atraso e taxa de transmissão não puderam ser considerados. O pequeno tamanho da amostra decorre da necessidade de executar diversos testes em pouco tempo. Testes mais longos e com mais amostras podem ser realizados no futuro.

Nos testes onde as duas máquinas disputam o barramento, cada máquina executa seus próprios testes de atraso. Portanto, estes testes devem refletir valores diferentes para cada uma. Para ilustrar o atraso traçou-se gráficos com as seguintes variáveis:

- **Mínimo** - menor atraso experimentado por ambas as máquinas durante o teste na taxa especificada.
- **Máximo** - maior atraso experimentado por ambas as máquinas durante o teste na taxa especificada.
- **Média A** - atraso médio experimentado pela máquina A durante o teste na taxa especificada. A média é dada pela soma das amostras, dividida pelo número de amostras.
- **Média B** - atraso médio experimentado pela máquina B durante o teste na taxa especificada. A média é dada pela soma das amostras, dividida pelo número de amostras.

Teste 1

A Figura 7.2 mostra as taxas de transmissão alcançadas pela aplicação (na máquina B) ao tentar enviar o fluxo de dados, conforme o solicitado. Observa-se que a maior taxa alcançada está em torno de 7,1 Mbits/s.

No mesmo teste, verifica-se na Figura 7.3 o desempenho da rede. Enquanto a carga na rede era baixa, o atraso máximo experimentado permaneceu quase que constante. Após 2,4 Mbits/s o valor começou a sofrer variações aleatórias que, se intensificaram a partir de 4,8 Mbits/s. O atraso mínimo e o atraso médio seguiram o mesmo comportamento mas, não apresentaram grandes variações, ficando sempre abaixo dos 6 milissegundos.

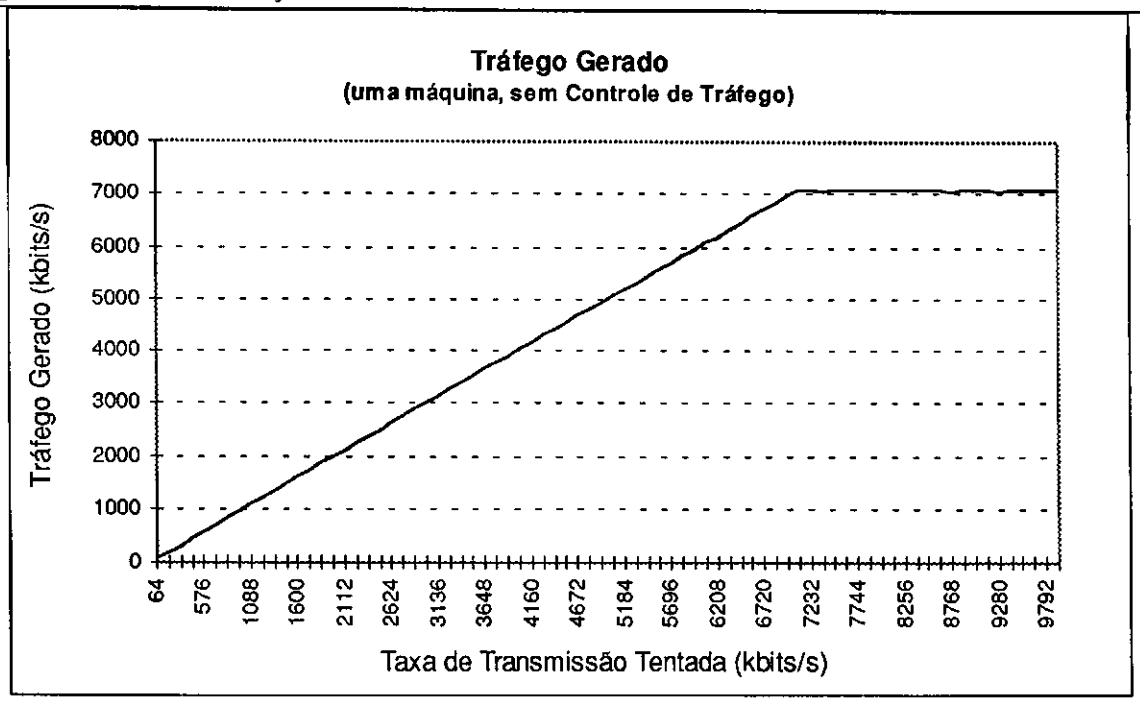


Figura 7.2 - Gráfico do tráfego de uma máquina, sem Controle de Tráfego

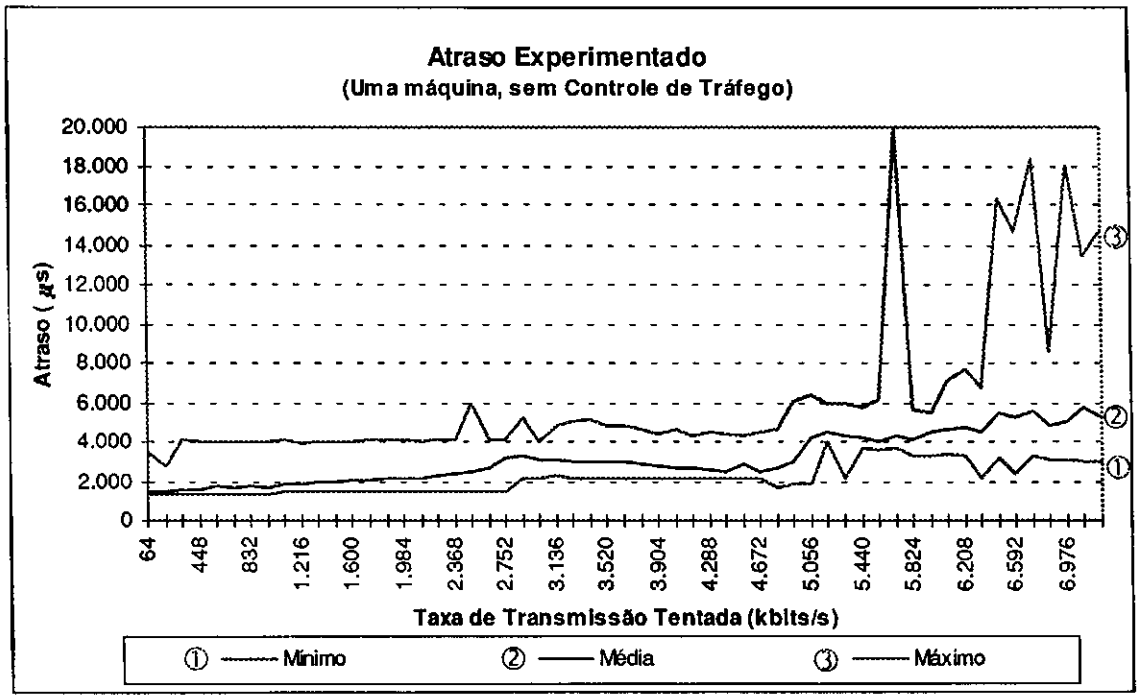


Figura 7.3 - Gráfico do atraso no tráfego de uma máquina, sem Controle de Tráfego

Teste 2

No segundo teste foram colocadas duas máquinas gerando o mesmo tráfego na rede e com isso, competindo pelo barramento. Na Figura 7.4 observa-se o desempenho de cada *host*

na competição pelo barramento. Até cerca de 6 Mbits/s (soma do tráfego gerado pelas duas máquinas) a taxa de transmissão das duas máquinas era equivalente. A partir deste ponto o *host* A começa a sofrer uma queda acentuada no desempenho. A taxa de transmissão total das duas máquinas chegou a 7,2 Mbits/s.

A diferença de desempenho entre as duas máquinas pode ser explicada pela capacidade de processamento que cada máquina possui. Como a máquina A é menos potente que a máquina B, se presume que o desempenho da aplicação seja menor quando a carga de processamento se eleva. Isso também afetaria o processo de transmissão do Sistema Operacional durante a disputa pelo meio após a ocorrência de colisões.

Ainda com as duas máquinas disputando o meio físico, sem escalonamento, observa-se o comportamento do atraso em ambas as máquinas. Na Figura 7.5 observa-se que as variações no atraso ocorrem bem mais cedo do que quando apenas uma máquina está transmitindo. O atraso máximo verificado reflete a aleatoriedade desta variável, devido à disputa pelo meio de transmissão. Os valores do atraso médio também sofrem discrepância a partir do ponto em que ocorre diferença na taxa de transmissão (Figura 7.4). Contudo, ele permanece baixo e sob controle.

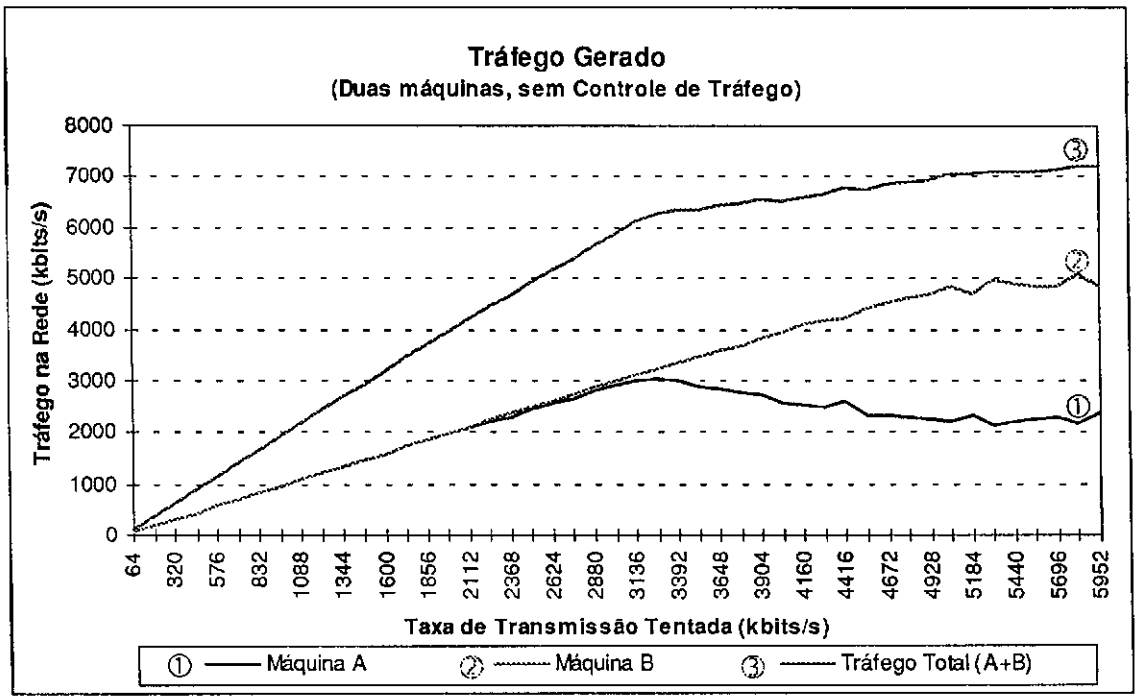


Figura 7.4 - Gráfico do tráfego de duas máquinas, sem Controle de Tráfego

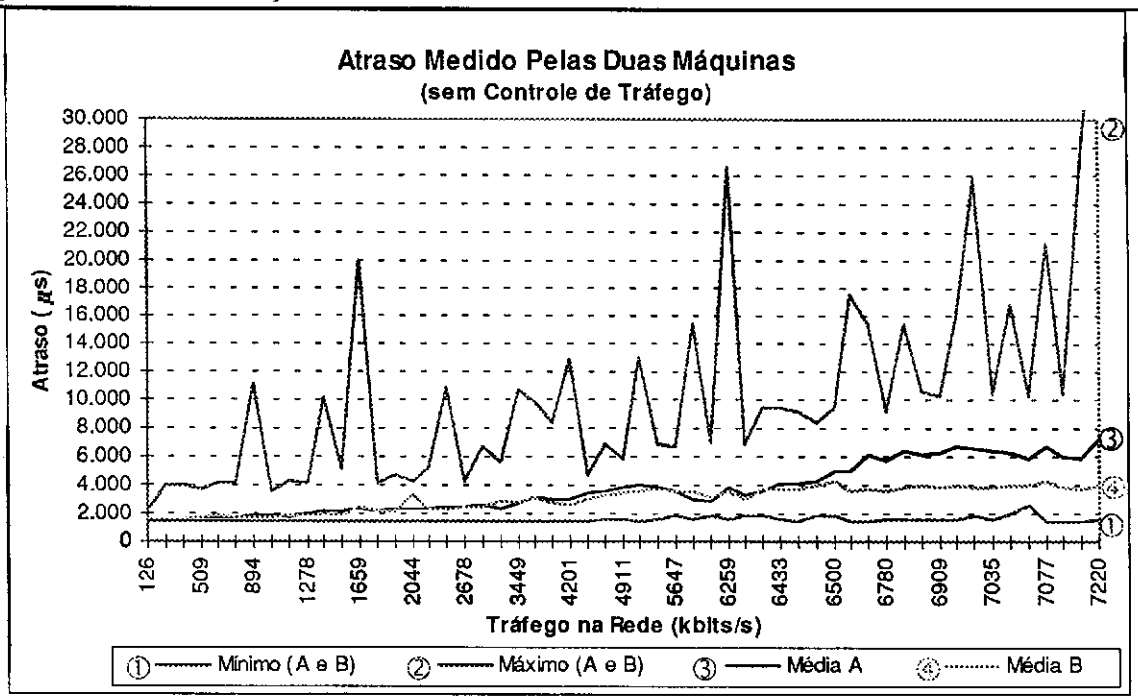


Figura 7.5 - Gráfico do atraso do tráfego de duas máquinas, sem Controle de Tráfego

Teste 3

Quando inserimos o mecanismo de Controle de Tráfego em uma máquina, a taxa máxima de transmissão é bastante reduzida, Com uma máquina transmitindo, esta taxa cai de 7,1 Mbits/s para 4,9 Mbits/s. Esta redução de cerca de 30% se deve, possivelmente, ao processamento extra para a localização das reservas e os pesados cálculos de taxa de transmissão e tempo no *Dispatch* dos pacotes. A utilização de máquinas com maior capacidade de processamento pode minimizar este problema.

Com relação ao atraso, observa-se na Figura 7.7 uma variação maior do atraso máximo experimentado. A variação ocorrida com o atraso máximo pode ser decorrente do mecanismo de escalonamento em degraus usado tanto pela aplicação, quanto pelo Controle de Tráfego. Um mecanismo com controle linear deve reduzir bastante esta variação.

Apesar disso, tanto o atraso mínimo, quanto o atraso médio obtiveram resultados mais uniformes e semelhantes aos que foram experimentados na máquina sem Controle de Tráfego, não passando também de 6 milissegundos. Este resultado pode ser considerado excelente, pois a inserção dos mecanismos de Classificação e Escalonamento no *kernel* poderia provocar um grande aumento no atraso mínimo e médio dos pacotes.

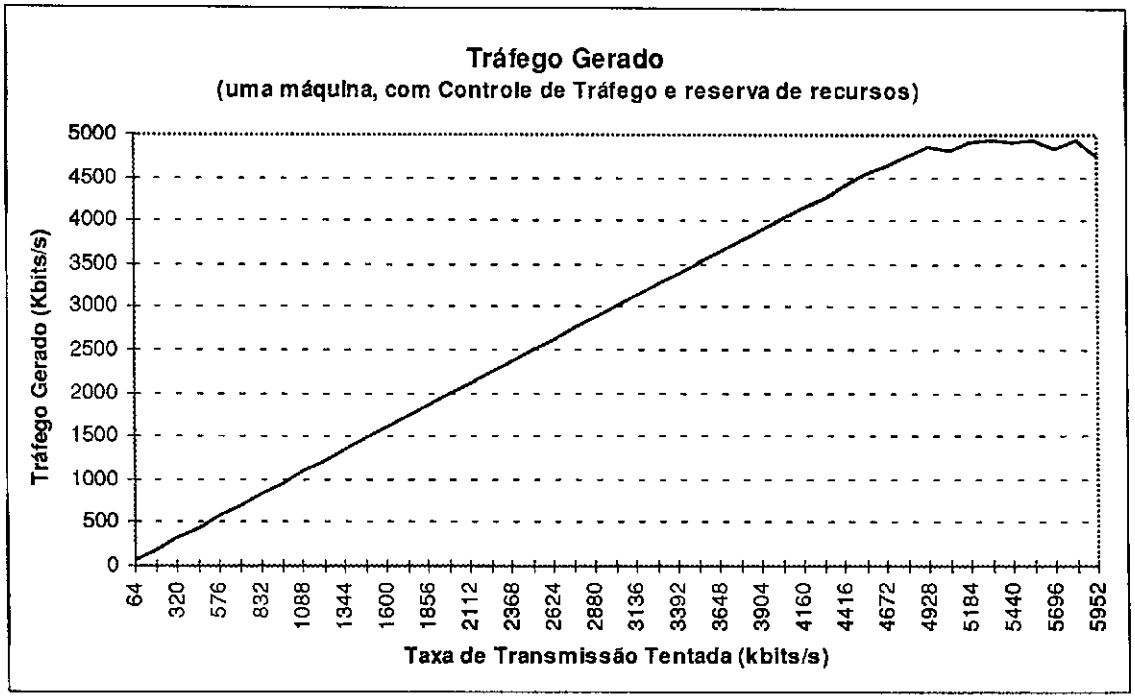


Figura 7.6 - Gráfico do tráfego de uma máquina, com Controle de Tráfego e reserva de recursos

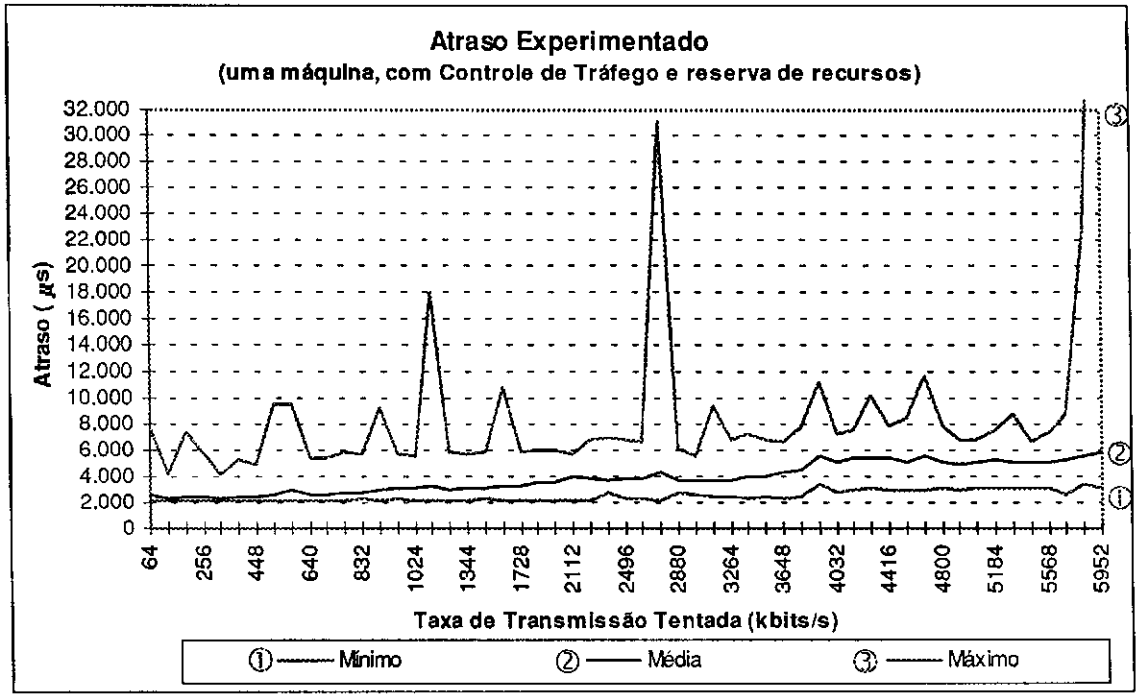


Figura 7.7 - Gráfico do atraso pelo tráfego gerado por uma máquina, com Controle de Tráfego e reserva de recursos

Teste 4

No teste com duas máquinas transmitindo dados com Controle de Tráfego e reserva de recursos somente foi possível a solicitação de reserva para as duas máquinas, quando solicitaram até 4,16 Mbits/s cada uma, como mostra a Figura 7.8. Após este valor o Controle de Admissão passou a recusar as reservas, pois o atraso esperado seria superior ao reservado. Porém, a vazão individual não atingiu a taxa reservada.

A taxa máxima transmitida na rede ficou em 6,3 Mbits/s, contra os 7,2 Mbits/s do teste sem escalonamento. A discrepância entre o valor tentado e o realmente transmitindo (tal qual ocorre no modelo sem escalonamento), surge a partir de 5 Mbits/s, contra os 6 Mbits/s anteriores. As justificativas para isso podem ser as mesmas da redução da taxa máxima transmitida por uma única máquina.

A vazão máxima obtida pela máquina B ficou em 4,1 Mbits/s e a da máquina A em 2,8 Mbits/s. Esses valores porém, não são atingidos ao mesmo tempo, havendo uma discrepância quando a taxa individual de transmissão atinge 2,6 Mbits/s. Com esse valor, pode-se estipular uma taxa máxima que pode ser reservada quando duas máquinas disputam o meio, para que não haja perda de performance na transmissão.

Comparando a vazão máxima obtida por uma única máquina com o valor controlável com duas máquinas, temos então um valor máximo de 5,2 Mbits/s na rede e 4,9 Mbits/s individual. Como a vazão individual de uma máquina depende também da sua capacidade de processamento, presume-se que a capacidade de processamento de aplicações que farão uso de reserva de recursos serão superiores às das máquinas utilizadas nos testes. Pode-se então, estender o limite para a rede, também para uma única máquina. Ou seja, o limite de reservas seria de 5,2 Mbits/s para as redes do tipo Ethernet, a 10 Mbits/s. Quando as solicitações de reserva ultrapassarem este limite, serão recusadas.

Como mostra a Figura 7.9, o atraso experimentado no teste com duas máquinas gerando tráfego, com escalonamento e reserva de recursos, também demonstrou uma grande variação no atraso máximo. Porém, o atrasos mínimo e o atraso médio tiveram um comportamento mais homogêneo, com diferenças pequenas entre as duas máquinas.

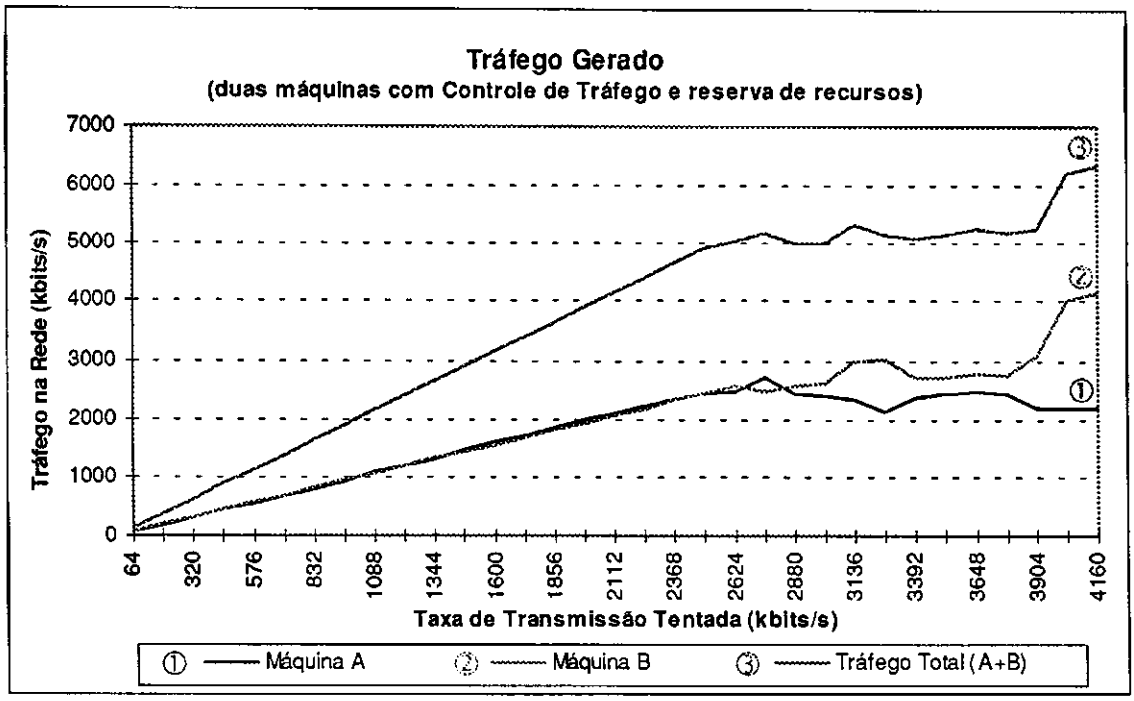


Figura 7.8 - Gráfico do tráfego de duas máquinas, com Controle de Tráfego e reserva de recursos

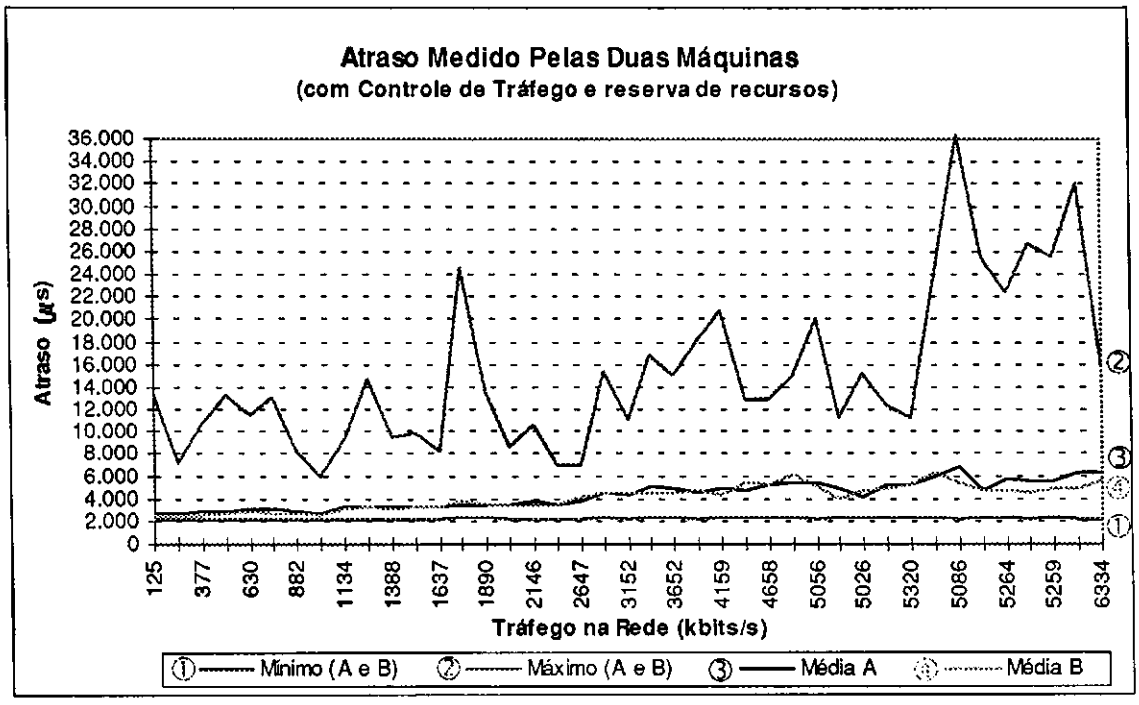


Figura 7.9 - Gráfico do atraso no tráfego de duas máquinas, com Controle de Tráfego e reserva de recursos

Teste 5

O último teste realizado foi com uma máquina transmitindo pacotes *best-effort*, com escalonamento (Figura 7.10). A taxa máxima obtida se estabilizou em torno de 4,3 Mbits/s.

O mecanismo de reavaliação das reservas teve o comportamento esperado. Os ajustes e a troca de mensagens ocorreram rapidamente, sem prejuízo do fluxo.

Até a realização dos testes a troca de mensagens pelo DCRP era feito com prioridade 7 (a mesma dos datagramas). Após os testes as mensagens passaram a ser transmitidas sem escalonamento. Espera-se com isso agilizar a transmissão das mensagens e medir o atraso na rede e não o do tráfego *best-effort*.

O atraso experimentado ilustrado na Figura 7.11 se manteve próximo ao limite de 1 segundo. Isto é por causa do degrau de reserva que permite um atraso máximo esperado de 1 segundo, antes de expandir a reserva (Seção 6.1). A grande variação do atraso mínimo e médio durante o teste se deve a um problema de sincronismo entre o módulo servidor e o cliente da aplicação, que utiliza mensagens UDP. A redução destes atrasos quando a taxa de transmissão se estabiliza também se deve a este probleme de sincronismo. Pois, quando os pacotes de sincronismos são perdidos, leva-se algum tempo para recincronizar a aplicação, permitindo o esvaziamento do buffer.

Conforme a necessidade, pode-se reduzir o limite para reavaliação da reserva do tráfego *best-effort* para, por exemplo, 100 ms. Com isso, o atraso máximo esperado para este tipo de tráfego ficaria de acordo com os limites para as atuais aplicações de tempo real.

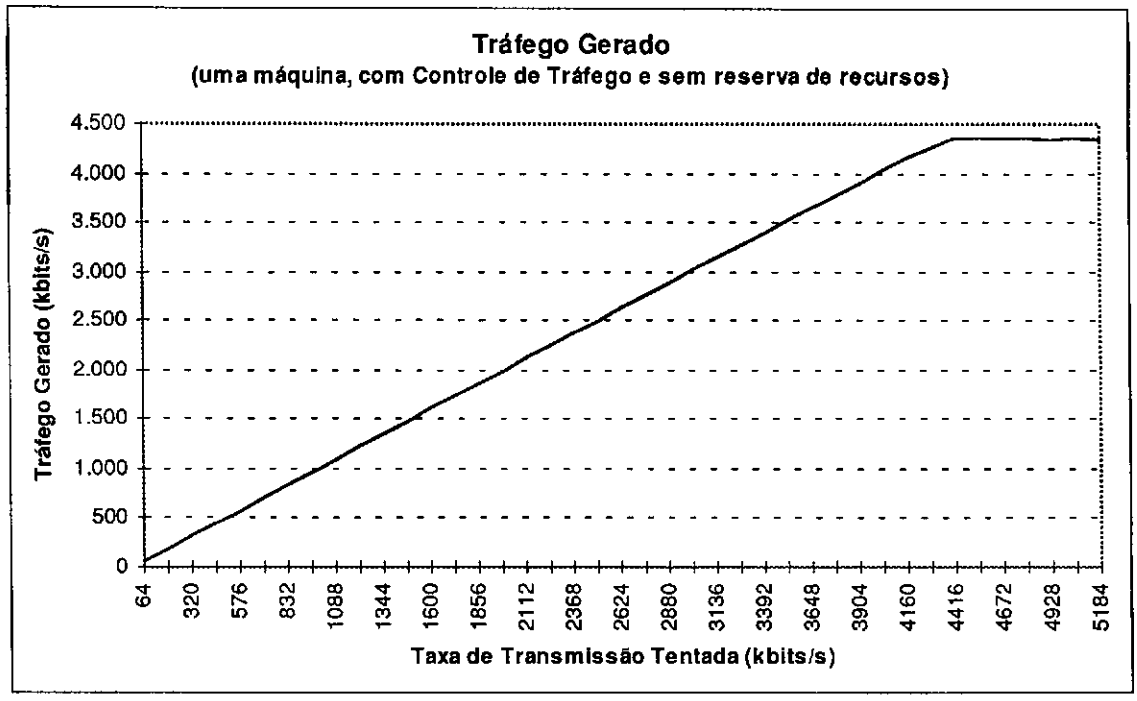


Figura 7.10 - Gráfico do tráfego de uma máquina, com Controle de Tráfego e sem reserva de recursos

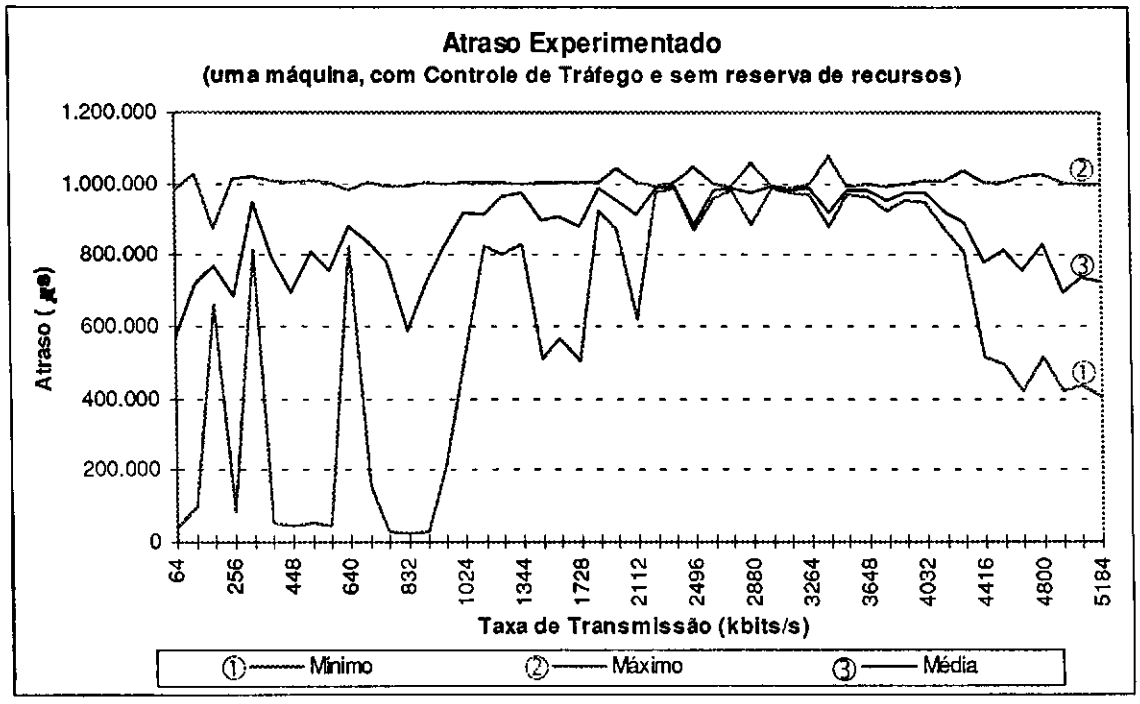


Figura 7.11 - Gráfico do atraso no tráfego de uma máquina com escalonamento e sem reserva de recursos

7.4 Resumo dos Resultados dos Testes

A seguir serão apresentados os resumos dos testes. Primeiro, os testes com a aplicação NV e em seguida os cinco testes realizados com a aplicação desenvolvida para os testes de tráfego.

NV

- Taxa máxima de transmissão sem Controle de Tráfego em torno de 450 kbits/s;
- Taxa máxima de transmissão com Controle de Tráfego e sem reserva em torno de 300 kbits/s, com rápida adaptação do Controle de Tráfego e qualidade melhor com taxas de transmissão altas;
- Taxa de reserva de recursos fixa;
- Perda de pacotes com transmissão acima do reservado;
- Qualidade de transmissão aceitável para vídeo com pouca variação de movimentos;

Rs_aplic - Teste 1

- Vazão máxima de 7,1 Mbits/s;
- Atraso máximo experimentado estável até 5,5 Mbits/s;
- Atraso médio experimentado abaixo de 6 ms e;
- Atraso mínimo experimentado abaixo de 4 ms.

Rs_aplic - Teste 2

- Vazão máxima de 7,2 Mbits/s na rede;
- Vazão individual controlada até 3 Mbits/s. A partir deste ponto, a curva de vazão de uma das máquinas declina;
- Atraso máximo experimentado bastante instável em ambas as máquinas;
- Atraso médio experimentado estável, com pequena discrepância entre as duas máquinas e;
- Atraso mínimo semelhante ao do teste 1.

Rs_aplic - Teste 3

- Vazão máxima de 4,9 Mbits/s na rede;
- Atraso máximo experimentado maior e mais instável devido ao mecanismo de escalonamento utilizado tanto pela aplicação quanto pelo *Dispatch*;
- Valores baixos para o atraso médio e atraso mínimo, ficando nos mesmos patamares do teste 1 e;
- Valores mais homogêneos para o atraso médio e mínimo, demonstrando uma variação menor nestes tipos de atraso.

Rs_aplic - Teste 4

- Vazão máxima na rede de 6,3 Mbits/s;
- Vazão individual controlada até 2,6 Mbits/s. A partir deste ponto, a curva de vazão de uma das máquinas declina;
- Pode-se estabelecer um limite máximo de reserva na rede (por todas as máquinas), em 5,2 Mbits/s para redes do tipo Ethernet a 10 Mbits/s;
- Valores baixos para o atraso médio e atraso mínimo, ficando nos mesmos patamares do teste 2 e;

- Valores mais homogêneos para o atraso médio e mínimo, demonstrando uma variação menor nestes tipos de atraso.

Rs_aplic - Teste 5

- Vazão máxima na rede de 4,3 kbits/s;
- Grande variação no atraso devido ao mecanismo de reavaliação do tráfego *best-effort*;
- Atraso máximo em torno de 1 segundo;
- Pode-se reduzir o degrau de reavaliação do tráfego *best-effort* para se obter atrasos menores e mais condizentes com aplicações de tempo real;

Capítulo 8
Conclusão

Ao longo deste documento foram apresentados alguns conceitos, que provavelmente deverão ser comuns a quem trabalha ou irá trabalhar com o desenvolvimento de tecnologias de rede e até mesmo a usuários das novas aplicações de tempo-real que estão surgindo. Entre estes conceitos, estão a Integração de Serviços (Capítulo 2), reserva de recursos (Capítulo 3) e mecanismos de Controle de Tráfego (Capítulos 4, 5 e 6).

Para experimentar o impacto da inserção destes conceitos em um ambiente tipicamente de comutação de pacotes (como é o ambiente oferecido atualmente pelo IP), foram propostos e implementados mecanismos que adaptem o atual modelo IP. Nestas propostas foram utilizadas as sugestões definidas pelos grupos de trabalho que estudam a Integração de Serviços (Int-Serv), o RSVP e a aplicação disso nas tecnologias de redes (ISSLL).

Como suporte de rede, foram escolhidas as redes compartilhadas (no caso, a Ethernet). Como estes tipos de redes não possuem mecanismos de controle de atraso ou banda, foi necessário desenvolver um mecanismo de gerência de recursos de rede que executasse este controle.

O mecanismo proposto para a execução do gerenciamento de recursos na rede é o DCRP (*Distributed Control Reservation Protocol*). Este mecanismo consiste de um protocolo de controle distribuído, que mantém um controle quantitativo dos recursos de rede utilizados

por todas as máquinas conectadas à rede. Os recursos gerenciados são a banda utilizada por cada uma destas máquinas e o atraso decorrente do tráfego gerado na rede.

As informações controladas por este protocolo são repassadas ao mecanismo de Controle de Tráfego. O mecanismo de Controle de Admissão irá utilizar estas informações para decidir pela aceitação ou não de novas solicitações de recursos de rede.

As solicitações de recursos (ou serviços) podem ser feitas diretamente por uma aplicação ou protocolo, utilizando a interface com o Controle de Tráfego, dentro do modelo de Integração de Serviços, ou através do RSVP.

O Controle de Tráfego implementa dois tipos de serviço:

1. um serviço simples de reserva de banda com um atraso médio controlado e;
2. um serviço de transmissão do tráfego *best-effort* por demanda.

O serviço de reserva oferecido pode ser "mapeado" para outros serviços especificados para Integração de Serviços, conforme as regras definidas em [BS95]. Por enquanto, usaram-se os serviços implementados na interface RSVP/Controle de Tráfego. Estes serviços são o Serviço Garantido, Serviço Predito e de Datagramas. Como o serviço oferecido não dá as mesmas garantias definidas nos serviços "mapeados", deve-se considerar este mapeamento como "não confiável" [BS95].

As máquinas também devem ser capazes de oferecer um serviço de transmissão do tráfego *best-effort*. Como a simples transmissão deste tráfego na rede irá degradar seu funcionamento, optou-se por uma reserva prévia de recursos em cada máquina, para sua transmissão.

Através de um mecanismo que chamamos de "*buffer* elástico", é feita uma reserva mínima de banda para o tráfego *best-effort*. Quando tráfego aumenta e o *buffer* fica cheio, é solicitada a ampliação desta reserva. Se novamente o *buffer* vier a encher, uma nova ampliação é solicitada. Isto pode ocorrer enquanto houver recursos disponíveis. Se não houver mais recursos disponíveis e o *buffer* se encher, os novos pacotes passam a ser descartados. Quando o tráfego diminui e o *buffer* se esvazia, são feitas reduções consecutivas da reserva até chegar à reserva inicial

A identificação dos pacotes para que possam receber o tratamento adequado no Controle de Tráfego, é feita pelo mecanismo de Classificação de Pacotes. O mecanismo proposto implementa dois tipos de classificação:

1. classificação de pacotes com reserva através de filtros e;

2. classificação do tráfego *best-effort*.

Nos pacotes que possuem reserva, é verificado o filtro do pacote, "abrindo" o pacote para verificar o endereço de destino, o endereço de origem, a porta (UDP/TCP) de destino, a porta (UDP/TCP) de origem e o protocolo que gerou o pacote. Uma vez identificada a reserva, o pacote é enviado para o escalonamento. Se não for identificada a reserva, o pacote é tratado como *best-effort*.

Para o tráfego *best-effort* não há padronização. Por isso, é proposto um esquema com 8 (oito) níveis de prioridades (7 para maior prioridade e 0 para menor prioridade) . Os pacotes podem ser identificados de quatro formas. A ordem em que é feita a identificação é a seguinte:

1. através de reserva de recursos;
2. campo PRECEDENCE do cabeçalho IP;
3. campo TOS do cabeçalho IP ou;
4. reconhecimento da aplicação através das portas TCP/UDP.

A primeira é a reserva de recursos, onde, ao invés de definir banda ou atraso, se escolhe um nível de prioridade para o fluxo de dados. Neste caso, o pacote é identificado através de filtro, da mesma forma que ocorre com os demais fluxos de dados com reserva de recursos.

Se não há reserva de recursos para o pacote, é verificado o campo PRECEDENCE do cabeçalho IP. Este campo possui três bits, podendo comportar a definição de prioridade do pacote. Seu uso atualmente é raro e não padronizado.

Se não houver informações no campo PRECEDENCE, é verificado o campo TOS. Este campo possui uma padronização de uso e é utilizado por diversas aplicações, principalmente para a priorização dos pacotes de controle (ICMP) e na escolha de rotas. No Capítulo 5 foi mostrado como pode ser feito um mapeamento entre as prioridades padrão para este campo e o mecanismo de Classificação de Pacotes.

Se não houver nenhum valor no campo TOS, pode-se tentar identificar a aplicação que gerou o pacote através das portas TCP/UDP bem conhecidas, como o NFS, Telnet, FTP, Fax, etc. Para isso, verifica-se a identificação das portas de origem e depois de destino do pacote, comparando com uma tabela de prioridades. Se não for possível identificar o pacote, ele é enviado com a menor prioridade possível (0).

Os serviços definidos pelo Int-Serv frizam que não deve haver fragmentação dos pacotes IP, para que se possa usar Qualidade de Serviço. Porém, as atuais estruturas de redes são bastante heterogêneas, o que implica na passagem dos pacotes por diversos tipos de enlaces, onde um deles pode ter uma MTU menor do que os demais, ocorrendo fragmentação. Por isso, é proposto um mecanismo de identificação de fragmentos para os fluxos de dados com reserva de recursos.

Quando chega o primeiro pacote fragmentado, sua identificação (identificação de fragmentos IP) é armazenada em uma tabela, junto com os descritores de reserva. Assim, quando chegam outros fragmentos do mesmo pacote, eles poderão ser imediatamente identificados e enviados para o escalonamento, sem a necessidade de qualquer mecanismo de recomposição de pacotes.

O mecanismo de identificação de fragmentos foi inicialmente definido apenas para os pacotes com reserva de recursos. Mas, pode ser estendido para os demais pacotes, na identificação de prioridades.

Uma vez identificados os pacotes, eles são enviados para o mecanismo de Escalonamento de Pacotes, que irá colocá-los em uma fila de escalonamento. Cada reserva de recursos e prioridade tem sua própria fila de escalonamento. Isto possibilita a separação do tráfego, para que cada um receba a Qualidade de Serviço solicitada.

O mecanismo de transmissão (*Dispatch*) do Escalonador de Pacotes consiste de um processo sem fim que varre constantemente as filas de escalonamento a procura de pacotes para transmissão. Quando é localizado um pacote na fila e o tráfego de saída desta fila não extrapolou a taxa de transmissão reservada, o pacote é imediatamente enviado para a interface de saída.

O mecanismo de *dispatch* envia apenas um pacote de cada vez em cada uma das filas, ao invés de esvaziar primeiro uma fila para então ir para a fila seguinte. Isto possibilita um atraso mínimo de pacotes menor do que na segunda opção, conforme foi demonstrado no Capítulo 6.

As filas de prioridades são contabilizadas pelo *Dispatch* como se fossem uma única fila. Quando é encontrado um pacote para transmissão e não foi extrapolada a taxa de transmissão reservada para o tráfego *best-effort*, o pacote encontrado na fila de maior prioridade é enviado à interface de rede. Após o pacote ser enviado, o *Dispatch* aponta para a fila com reserva seguinte e não para a próxima fila de prioridade.

Durante a aplicação dos testes, os mecanismos implementados apresentaram resultados satisfatórios. O atraso médio obtido foi sempre inferior ao atraso esperado pelo Controle de Admissão, ficando entre 2 e 6 milissegundos. O atraso mínimo nos testes ficou abaixo de 3 milissegundos e o atraso máximo não chegou a 40 milissegundos.

A variação de atraso ocorrida durante os testes é de se esperar em uma rede onde a ocorrência de colisões é comum. Como o mecanismo não se propõe a fazer controle de variação de atraso (*jitter*) e a maioria dos serviços definidos pelo Int-Serv também não exigem este controle, este resultado não chega a ser preocupante. É claro que, para alguns destes serviços, espera-se a menor variação de atraso possível.

Resultados melhores podem vir a ser obtidos se aplicarmos os mecanismos propostos em redes derivadas da Ethernet com maior vazão (mas, ainda com colisões), ou nas redes locais onde não há colisões, como as redes Token-Ring e FDDI [Tan94]. Nas redes em anel é possível obter uma estimativa de tempo de transmissão e, com isso, reduzir as variações no atraso.

Há porém uma dificuldade de especificação de uma adaptação do DCRP para o funcionamento em redes com *switches*. A simples aplicação da proposta neste tipo de rede reduziria bastante o atraso e sua variação, mas produziria uma subalocação de recursos. Porém, há soluções para o problema, com a adaptação do protocolo distribuído para executar questionamento de recursos em quem vai receber o fluxo de dados no momento da solicitação de reserva. Isto também exigiria alterações no mecanismo de Controle de Admissão. Tais adaptações podem ser feitas em trabalhos futuros.

O mecanismo de Classificação de Pacotes funcionou como deveria. O uso de mecanismo de identificação de fragmentos sem precisar reconstruir o pacote também funcionou a contento. Sua ampliação para os pacotes sem reserva é recomendada. Assim, ficaria mais fácil a identificação dos pacotes, sem a perda da prioridade exigida pela aplicação. A possibilidade de fragmentação de pacotes, ao longo do caminho dos dados, reduz a necessidade de controle sobre a MTU mínima ao longo do caminho por parte da aplicação ou protocolo de controle. Isso também abre a possibilidade de uso das atuais aplicações que não fazem este tipo de controle. Porém, a fragmentação não deve ser tratada como regra. Sempre que possível, deve-se eviá-la e, com isso, reduzir o processamento necessário na transmissão/retransmissão dos pacotes.

O uso do campo PRECEDENCE do cabeçalho IP parece ser uma boa opção para a identificação de prioridades para as aplicações. A atribuição de valores para este campo é

bastante simples e fácil de implementar. Contudo, deve haver uma padronização de uso para se evitar, por exemplo, que uma aplicação pouco sensível ao atraso venha solicitar prioridade máxima de transmissão. A experimentação de vários mecanismos de atribuição de prioridades, como os propostos, permite que se possa decidir sobre a melhor escolha.

O mecanismo de Escalonamento de pacotes também apresentou o resultado esperado. O problema de variação de atraso na transmissão de pacotes, provocada por este mecanismo, se deve principalmente ao degrau de tempo utilizado pelo *kernel*. O problema seria resolvido com degraus de tempo menores ou o uso de tempo real, implicando em maior processamento, menor vazão e maior atraso. Espera-se que com o aumento da capacidade de processamento das máquinas, isso não venha a ser fator limitante.

Pode-se identificar então os seguintes pontos positivos no trabalho:

- ☺ gerenciamento dos recursos de rede;
- ☺ mecanismo de controle descentralizado;
- ☺ uso de *soft-state* pelo DCRP;
- ☺ atraso médio controlado;
- ☺ atraso mínimo pequeno;
- ☺ atraso máximo abaixo das necessidades das aplicações atuais;
- ☺ mapeamento dos serviços existentes para o serviço oferecido;
- ☺ mecanismo de identificação de fragmentos;
- ☺ múltiplos mecanismos de identificação para o tráfego *best-effort*;
- ☺ separação das filas de escalonamento;
- ☺ funcionamento com o RSVP;
- ☺ uso imediato para testes de aplicações e reserva de recursos e;
- ☺ possibilidade de uso sobre outras tecnologias de rede.

Como pontos a serem ajustados temos o seguinte:

- ☹ variação de atraso alta;
- ☹ comunicação assíncrona entre o DCRP e o *kernel* e;
- ☹ DCRP não faz tratamento de superalocação de recursos por colisões de reservas.

A implementação do mecanismo de controle de tráfego implicou na inclusão de aproximadamente 1600 linhas de código no *kernel* do Sistema Operacional, sem com isso aumentar o atraso dos pacotes.

Para trabalhos futuros, pensou-se nas seguintes opções:

- realização de testes com amostras mais significativas e comparação com outras implementações (quando houver);
- adaptação do DCRP e do Controle de Admissão para funcionar em redes Ethernet com *switch*;
- possibilidade de configuração pelo administrador do sistema;
- criação de uma interface própria entre o RSVP e o Controle de Tráfego.
- aplicação e testes dos mecanismos em outras tecnologias de rede;
- preparação de artigos sobre o assunto;
- preparação de um documento para ser enviado ao grupo de trabalho ISSLL com a especificação do DCRP e;
- preparação da implementação para distribuição pela Internet, para testes e críticas.

Além dos mecanismos propostos, procurou-se com este documento oferecer uma grande variedade de referências bibliográficas sobre o assunto. As referências são bastante atuais e a maioria delas pode ser localizada na Internet, por quem se interessar pelo assunto.

Glossário

ACD	*Admission Control Daemon*
API	*Application Program Interface*
ATM	*Asynchronous Transfer Mode*
B-ISDN	*Broadband Integrated Service Digital Network*
BSD	*Berkeley Software Distribution*
CSZ	Mecanismo de Controle de Tráfego proposto por Clark, Shenker e Zhang [CSZ92]
DCRP	*Distributed Control Reservation Protocol*
DestAddress	Endereço de destino em uma sessão
DSBM	*Designated SBM*
DstPort	Porta de destino em uma sessão
DVMRP	*Distance Vector Multicast Routing Protocol*
FD	*File Descriptor*
FDDI	*Fiber Distributed Data Interface*
FF	*Fixed-Filter Style*
FIFO	*First In First Out*
filterspec	Especificação de Filtro de Pacotes
flowspec	Especificação de Fluxo de Dados
FTP	*File Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IEEE	*Intitute of Eletrical and Eletronics Engineers*
IETF	*Internet Engineering Task Force*
IGMP	*Internet Group Management Protocol*
Int-Serv	*Integrated Services Work Group*
IOCTL	Função de controle de interrupções para o *kernel* do UNIX
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
ISDN	*Integrated Service Digital Network*

ISI	*International Science Institute*
ISPN	*Integrated Services Packet Network*
ISSLL	*Integrated Services over Specific Link Layers*
jitter	Variação de Atraso
MBONE	*Multicast Backbone*
MOSPF	*Multicast Open Shortest Path First*
MTU	*Maximum Transmission Unit*
NFS	*Network File System*
NRR	*Network Resource Reservation Program*
NV	*Network Video Tool*
OPWA	*One Pass With Advertising*
OSPF	*Open Shortest Path First*
PIM	*Protocol Independent Multicasting*
PPP	*Point-to-Point Protocol*
QoS	*Quality of Service*
RAPI	*Resource ReServation Protocol Application Program Interface*
RDP	*Real Time Datagram Protocol*
RSpec	Especificação de Requisição de Serviço
RSVP	*Resource ReServation Protocol*
RTP	*Real-time Transport Protocol*
SBM	*Subnet Bandwidth Manager*
SE	*Shared Explicit Style*
SLIP	*Serial Line Internet Protocol*
ST	*STream Protocol*
ST-II	*Experimental Internet STream Protocol - Version 2*
TCP	*Transport Control Protocol*
TOS	Type of Service
TSpec	Especificação de Tráfego
TTL	*Time-to-Live*
UDP	*User Datagram Protocol*
UNIX	Sistema Operacional
UT	Unidades de Tempo
VIC	*Video Conferencing tool*

WF	*Wildcard-Filter Style*
WFQ	*Weighted version of the Fair Queuing algorithm*

Referências Bibliográficas

[Alb96] A. Albanese. Comunicação Pessoal, aa@ICSI.Berkeley.EDU, Abril 1996.

[Alm92] P. Almquist *'Type of Service in the Internet Protocol Suite'* Internet Network Working Group RFC 1349, July 1992.

[Atk95] R. Atkinson *'Security Architecture for the Internet Protocol'* Internet Network Working Group RFC 1825, August 1995.

[BCS94] R. Braden, D. Clark & S. Shenker *'Integrated Services in the Internet Architecture: an Overview'* Internet Network Working Group RFC 1633, July 1994

[Bel94] S. Bellovin *'Security Concerns for IPng'* Internet Network Working Group RFC 1675, August 1994.

[BG96] M. Borden & M. Garret *'Interoperation of Controlled-Load and Guaranteed-Service with ATM'* IETF Draft draft-borden-intserv-atm-mapping-00.txt, June 1996.

[BH96] R. Braden & D. Hoffman *'RSVP Application Programming Interface (RAPI) for SunOS/BSD: V4.0'* IETF Draft draft-ietf-rsvp-bsdapi.ps, May 1996.

[BS95] L. Breslau & S. Shenker *'A proposal for accommodating Heterogeneity'* IETF draft-ietf-intserv-hetero-01.txt, Xerox, November 1995.

[BZ93] R. Braudes & S. Zabele *'Requirements for Multicast Protocols'* Internet Network Working Group RFC 1458, may 1993.

[BZ96] R. Braden & L. Zhang *'Resource Reservation Setup Protocol (rsvp) Charter'* URL = http://www.ietf.org/html.charters/rsvp-charter.html, August 1996.

[BZBHJ96] R. Braden, L. Zhang, S. Berson, S. Herzog & S. Jamin *'Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification'* IETF Draft draft-irtf-rsvp-spec-13.ps, August 1996.

[CD92] S. Casner & S. Deering, "First IETF Internet Audiocast" *ACM SIGCOMM Computer Communications Review*, San Diego California, July 1992, pp. 92-97.

[Com91] D. Commer ***'Internetworking With TCP/IP Volume I'***: Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[CSZ92] D. Clark, S. Shenker & L. Zhang "Supporting Real-Time Applications in a

Integrated Service Packet Network: Architeture and Mechanism" *Proceedings of SIGCOMM'92*, Baltimore, MD, August 1992.

[DB95] L. Delgrossi, L. Berger (eds.) *'Experimental Internet Stream Protocol, Version 2 (ST-II) Protocol Specification - Version ST2+'* Internet Network Working Group RFC 1819, ST2 Working Group, August 1995.

[DC85] S. Deering & D. Cheriton *'A Multicast Extension to the Internet Protocol'* Internet Network Working Group RFC 966, December 1985.

[Dee89] S. Deering *'Host Extensions for IP Multicasting'*: Internet Network Working Group RFC 1112, August 1989.

[DEFJLW95] S. Deering, D. Estrin, D. Farrinaci, V. Jacobson, C. Liu & L. Wei *'Protocol Independent Multicasting (PIM): Protocol Specification'* IETF Network Working Group draft-ietf-idmr-pim-spec-01.txt, January, 1995

[DH95] S. Deering & R. Hinden *'Internet Protocol, Version 6 (IPv6) Specification'* Internet Network Working Group RFC 1883, Xerox PARC, Ipsilon Networks, December 1995.

[Dua96] O. Duarte *'Qualidade de Serviços e Protocolo de Alta Velocidade - minicurso'* 14º Simpósio Brasileiro de Redes de Computadores, Fortaleza, Maio 1996.

[FGD96] F. Baker, R. Guerin & D. Kandlur *'Specification of Committed Rate Quality of Service'* IETF Integrated Service WG draft-ietf-intserv-commit-rate-svc-00.txt, June 1996.

[For79] J. Forgie *'ST - A Proposal Internet Stream Protocol'* Internet Experimental Notes IEN-119, September 1979.

[Fre94] R. Frederick, *'Experiences With Real-Time Software Video Compression'* XEROX PARC, July 1994.

[Fre96] FreeBSD 'FreeBSD Home Page' URL = http://www.freebsd.org, August 1996.

[GHMN94] A. Gupta, W. Howe, M. Moran & Q. Nguyen *'Scalable Resource Reservation for Multi-Party Real-Time Communication'* Tenet Group/University of California at Berkeley & International Computer Science Institute TR-94-050, October 1994.

[HC94] E. Huizer & D. Crocker *'IETF Working Group Guidelines and Procedures'*: Internet Network Working Group RFC 1603, March 1994.

[Her96a] S. Herzog *'Accounting and Access Control Policies for Resource Reservation Protocols'* IETF Draft draft-ietf-rsvp-policy-arch-00.ps, June 1996.

[Her96b] S. Herzog *'Policy Enforcement for Resource Reservation Protocols'* IETF Draft draft-ietf-policy-lpm-00.ps, June 1996.

[Her96c] S. Herzog *'RSVP Extensions for Policy Control'* IETF Draft draft-ietc-policy-ext-00.ps, June 1996.

[IBM94] IBM. *'IBM Network Resource Reservation Program'* IBM Anoucement 294-551, September 1994.

[ISI96] International Science Institute, RSVP Implementation, URL= ftp://ftp.isi.edu/rsvp/release/ rel4.0a7.tar.Z, August 1996.

[Kim96] P. Kim *'Link Level Resource Management Protocol (LLRMP) - Protocol Specification - Version 0'* IETF Draft draft-kim-llrmp-00.ps, June 1996.

[Kra96] J. Krawczyk *'Providing Integrated Services in the Presence of Layer-2 Frame Switching Devices'* Internet Draft draft-krawczyk-intserv-12-switch-00.txt, February 1996.

[Kum95] V. Kumar ***'MBone: Interactive Multimedia On The Internet'***: Macmillan Publishing, Simon & Schuster, November 1995.

[Kum96] V. Kumar "The MBone Information WEB" URL = http://www.best.com/ ~prince/techinfo/mbone.html, June 1996.

[Man94] P. Manzoni *'Introducing Reseouces Management in IP-Based Nodes'* TR-94-040, International Computer Science Institute, October 1994.

[MESZ94] D. Mitzel, D. Estrin, S. Shenker & L. Zhang. "An Architectural Comparation of ST-II and RSVP" *Proceedings of Infocomm*, 1994.

[Mon94] J. Monteiro ***'Rede Digital de Serviços Integrados de Faica Larga (RDSI-FL)'***: IX Escola de Computação / UFPE-DI, Recife, 1994.

[Moy94a] J. Moy *'Multicast Extensions to OSPF'* Internet Network Working Group RFC 1584, March 1994.

[Moy94b] J. Moy *'OSPF Version 2'* Internet Network Working Group RFC 1583, March 1994.

[Par92a] A. Parekh *'A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks'* Techinical Report LIDS-TR-2089, Laboratory for Information and Decision Systems, Massachusets Institute of Technology, 1992.

[Par92b] C. Partridge *'A Proposed Flow Specification'* Internet Network Working Group RFC 1363, September 1992.

[Par95] C. Partridge *'Using the Flow Label Field in IPv6'* Internet Network Working Group RFC 1819, June 1995.

[Pos81] J. Postel *'Internet Protocol'* Internet Network Working Group RFC 791, September 1981.

[PP88] W. Prue & J. Postel *'A Queuing Algorithm to Provide Type-of-Service for IP Links'* Internet Network Working Group RFC 1046, February 1988.

[Rom88] J. Romkey *'A Nonstandard for Transmission of IP Datagrams Over Serial Lines -SLIP'* Internet Network Working Group RFC 1055, June 1988.

[SB95] S. Shanker & L. Breslau "Two Issues in Reservation Establishment" *Proc. ACM SIGCOMM'95*, Cambridge, MA, August 1995.

[SBB96] M. Stanton, L. Barra & C. Bastos *'Integração de Serviços na Internet - minicurso'* 14º Simpósio Brasileiro de Redes de Computadores, Fortaleza, Maio 1996.

[SCFJ96] H. Schlzrinne, S. Casner, R. Frederick & V. Jacobson *'RTP: A Transport Protocol for Real-Time Applications'* Audio-Video Transport Work Group RFC1889, January 1996.

[She95] S. Shenker *'Specification of general Characterization Parameters'*: IETF Integrated Service WG draft-ietf-intserv-charac-01.txt, November 1995

[Sim93] W. Simpson *'The Point-to-Point Protocol (PPP)'* Internet Network Working Group RFC 1548, December 1993.

[SPDB95] S. Shenker, C. Partridge, B. Davie & L. *Breslau 'Specification of Predictive Quality os Service'* IETF Integrated Service WG draft-ietf-intserv-predictive-svc-01.txt, June 1995.

[SPG96] S. Shenker, C. Partridge & R Guerin *'Specification of Guaranteed Quality of Service'* IETF Integrated Service WG draft-ietf-intserv-guaranteed-svc-06.txt, August 1996.

[SPW95] S. Shenker, C. Partridge & J. Wroclawski *'Specification of Controled Delay Quality os Service'* IETF Integrated Service WG draft-ietf-intserv-control-del-svc-02.txt, November 1995.

[Ste94] W. Stevens ***'TCP/IP Illustraded, Volume 1 - The Protocols'***: Addisson-Wesley, Reading, Massachusetts, 1994.

[Sun90] Sun Microsystems, ***'SunOS Reference Manual Vol II'***: Sun Microsystems, march 1990.

[SW95] S. Shenker & J. *Wroclawski 'Network Element Service Specification Template'* IETF Integrated Service WG draft-ietf-intserv-template-02.txt, November 1995.

[Tan94] A. Tanenbaum ***'Redes de Computadores - Segunda Edição Americana'***: Editora Campus Ltda, Rio de Janeiro, 1994.

[Tob94] F. Tobagi. "High Speed Networks and Multimedia Communications: Applications, Infraestructure and Protocols" *ITS - International Telecommunications Symposium*, Rio de Janeiro, 1994.

[Wak94] I. Wakeman *'The UCL/LBL/Sun Stream Implementation of CBQ'* IETF RSVP meeting, Toronto, Canada, July 1994.

[WHD94] L. Wolf, R. Herrtwich & L. *Delgrossi 'Filtering Multimedia Data in Reservation-Based Internetworks'* Techinical Report 43.9408, IBM European Networking Center, Heidelberg, Germany, August 1994.

[WPD88] . D. Waitzman, C. Partridge & S. Deering *'Distance Vector Multicast Routing Protocol'*: : Internet Network Working Group RFC 1075, November 1988.

[Wro95] J. Wroclawski *'Standard Data Encoding for Integrated Services Objects'* IETF Integrated Service WG draft-ietf-intserv-data-encoding-01.txt, November 1995.

[Wro96] J. Wroclawski *'Specification of the Controlled-Load Network Element Service'* IETF Integrated Service WG draft-ietf-intserv-crtl-load-svc-02.txt, June 1996.

[WS95] G. Wright & W. Stevens ***'TCP/IP Illustraded, Volume 2 - The Implementation'***: Addisson-Wesley, Reading, Massachusetts, 1995.

[YPH96] R. Yavatkar, E. Patki & D. Hoffman *'SBM (Subnet Bandwidth Manager): A Proposal for Admission Control over Ethernet)'* IETF Draft draft-yavatkar-sbm-ethernet-00.txt, June 1996.

[ZDESZ93] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala. RSVP: 'A New Resuouce ReSerVation Protocol - Novel Design features lead to an Internet protocol that is flexible and scalable' *IEEE Network*, September 1993.