Alocação de Recursos em Redes Programáveis

Antônio Pires de Castro Júnior

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Alocação de Recursos em Redes Programáveis

Antônio Pires de Castro Júnior[1]

Novembro de 2001

Banca Examinadora:

- Nelson Luis Saldanha da Fonseca (Orientador)
- Jorge Moreira de Souza
  R&D Lucent Technologies
- Ricardo de Oliveira Anido
  Instituto de Computação, UNICAMP
- Edmundo R. M. Madeira (Suplente)
  Instituto de Computação, UNICAMP

[1]Projeto financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Alocação de Recursos em Redes Programáveis

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Antônio Pires de Castro Júnior e aprovada pela Banca Examinadora.

Campinas, 09 de Novembro de 2001.

Nelson Luis Saldanha da Fonseca
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 09 de novembro de 2001, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Jorge Moreira de Souza
R & D - Lucent

Prof. Dr. Ricardo de Oliveira Anido
UNICAMP

Prof. Dr. Nelson Luís Saldanha da Fonseca
IC - UNICAMP

As almas grandes têm esse privilégio; suas ações, que nos outros inspiram a admiração, se aniquilam em face dessa nobreza inata do coração superior, para o qual tudo é natural e possível. — Jose de Alencar em "O Guarani"

Em memória de meu avô *Segismundo Ramos Santeiro...*

Lista de Tabelas

Lista de Figuras

Acrônimos

ATM. *Asynchronous Transfer Mode*

DARPA. *Defense Advanced Research Projects Agency*

DiffServ. *Differentiated Services*

DNS. *Domain Name Server*

HTTP. *HiperText Transport Protocol*

IntServ. *Integrated Services*

IP. *Internet Protocol*

ISP. *Internet Service Provider*

MCF. *MultiCommodity Flow*

OPENSIG. *Open Signalling*

PNNI. *Private Network-Network Interface*

QoS. *Quality of Service*

RI. Roteadores Inteligentes

SLA. *Service Level Agreement*

SMTP. *Simple Mail Transfer Protocol*

TCP. *Transmission Control Protocol*

VN. *Virtual Networks*

VPN. *Virtual Private Networks*

VSP. *Virtual Service Provider*

Capítulo 1

Introdução

Dentre as áreas de pesquisa em redes sob intensa investigação podem-se citar: Redes Programáveis e Redes Virtuais Privadas. Redes programáveis permitem que os recursos da rede sejam acessados e manipulados de acordo com as necessidades dos usuários. Para introduzir programabilidade, é necessária a separação do hardware de comunicação do software de controle, que, atualmente, estão verticalmente integrados.

Dada a separação entre o hardware e o software, é possível disponibilizar interfaces abertas de programação, permitindo, assim, acelerar a virtualização da infra-estrutura da rede, a rápida criação e desenvolvimento de novos serviços de redes, a criação de ambientes para o particionamento de recursos e a coexistência de distintas arquiteturas de redes.

Para tornar o conceito de redes programáveis uma realidade, estão surgindo técnicas que permitem criar, controlar e gerenciar redes virtuais. Uma destas técnicas é conhecida como *switchlet*, cuja capacidade é dividir os recursos de um comutador físico em vários comutadores lógicos, possibilitando o gerenciamento dinâmico dos recursos de maneira flexível e diferenciada. As *switchlets* são usadas para criar redes virtuais sob-demanda.

As redes virtuais privadas (VPNs) [RSW99, Isa00] são entidades de serviços adaptadas para determinado tipo de tráfego e/ou determinado grupo de usuários. VPNs são utilizadas para diminuir o custo e a complexidade de prover uma rede física dedicada ao tráfego de serviços distintos. Assim sendo, existe a necessidade de alocar recursos de uma infra-estrutura física para as VPNs a fim de que se possa garantir requisitos de qualidade de serviço (QoS).

Alocação de recursos da rede para as várias VPNs não é simples, pois, dependendo de como os recursos são alocados, pode ocorrer uma degradação na utilização da rede ou desperdício de recursos.

É notória, contudo, a necessidade de os *VPN Service Providers* (VSP) adaptarem dinamicamente suas redes aos requisitos dos usuários, no sentido de automatizar e parametrizar o processo de criação, desenvolvimento e gerenciamento de distintas arquiteturas

de redes.

A presente dissertação investiga o problema de alocação dinâmica de recursos na criação de VPNs, e introduz métodos para que os VSPs maximizem a utilização dos recursos de rede, bem como o ganho neste processo. São propostos modelos baseados em *multicommodity flow* para solucionar o problema de alocação de recursos em redes baseadas em *switchlets*. Será mostrado que estes métodos são factíveis de serem implementados em tempo real.

O texto dessa dissertação está organizado da seguinte forma: o capítulo 2 apresenta uma revisão de literatura em redes programáveis; o capítulo 3 descreve, resumidamente, alguns problemas em Otimização de Fluxos em Rede; o capítulo 4 expõe o conceito de qualidade de serviço e de engenharia de tráfego em redes de dados; no capítulo 5, estuda-se a alocação de recursos para VPNs em redes baseadas em *switchlets* com abordagem usando busca em profundidade e Dijkstra para achar K caminhos mínimos; no capítulo 6, estuda-se a alocação de recursos para VPNs com múltiplas classes de serviços em redes baseadas em *switchlets* e, finalmente, no capítulo 7 as conclusões são derivadas.

Capítulo 2

Redes Programáveis

Este capítulo faz uma revisão de literatura em redes programáveis. São apresentadas as principais linhas de pesquisa para o desenvolvimento deste trabalho.

2.1 Introdução

A habilidade para rapidamente criar, desenvolver e gerenciar novos serviços em resposta à demanda dos usuários é fator chave que direciona a comunidade de pesquisa em redes programáveis. O impacto do resultado neste campo de pesquisa irá influenciar, de maneira considerável, usuários, provedores de serviços e os fabricantes de equipamentos no setor de telecomunicações[Laz97]. A competição entre os provedores de serviços será vencida por quem responder, rapidamente, à nova demanda de serviços do mercado.

O objetivo das redes programáveis é permitir o acesso e manipulação dos recursos físicos das redes mediante uma interface aberta de programação. É preciso, portanto, entender melhor as limitações das redes existentes e os fundamentos para torná-las programáveis.

O paradigma de redes programáveis está criando importantes inovações. Estas inovações incluem a separação entre o hardware de transmissão e o software de controle, e a conseqüente disponibilização de interfaces abertas de redes programáveis. Estas interfaces permitem: acelerar a virtualização da infra-estrutura da rede; a rápida criação e desenvolvimento de novos serviços de redes; a criação de ambientes para o particionamento de recursos e a coexistência de múltiplas arquiteturas de redes distintas.

A separação do hardware de comunicação (*switching fabrics*, *routing engines*) do software de controle é fundamental para tornar a rede programável. Esta separação é difícil de ser realizada, em razão de serem os comutadores verticalmente integrados. Os provedores de serviço, tipicamente, não têm acesso ao ambiente de controle de um comutador. Faz-se, então, necessário definir interfaces padrões para o acesso à rede física.

A introdução de novos serviços nas redes existentes é usualmente manual, consumindo tempo e custo no processo. O objetivo das redes programáveis é simplificar o desenvolvimento de novos serviços de redes de maneira que estes serviços sejam introduzidos de maneira automatizada em resposta à demanda dos usuários.

Duas escolas de pensamento diferem em como tornar as redes programáveis. A primeira escola é a *Open Signalling*[CKMV99], que foi estabelecida por meio de uma série de *workshops* internacionais. A outra escola, estabelecida pela *Defense Advanced Research Projects Agency* (DARPA)[TSS+97, Pro96], é constituída de um grande número de projetos em redes ativas.

Apartir de 1995, iniciou-se uma série de estudos, por meio de *workshops*, intitulados *Open Signalling* (OPENSIG), com o objetivo de tornar a ATM, Internet e redes móveis abertas, extensíveis e programáveis. Almejou-se introduzir a tecnologia de redes abertas para tornar as redes tão programáveis quanto um PC normal. A comunidade OPENSIG[Gro00] argumenta em favor de se modelar o hardware de comunicação usando o conjunto de interfaces abertas de redes programáveis, possibilitando, assim, um acesso aberto para os comutadores e roteadores. Afirma-se que novas e distintas arquiteturas e serviços podem ser realizados tornando os comutadores abertos a programação, possibilitando estabelecer uma clara distinção entre o transporte, controle e gerenciamento e a criação de serviços[CKMV99, CMK+99b].

O conceito de redes ativas surgiu por meio da comunidade de pesquisa em redes da DARPA. Em redes ativas, os comutadores da rede conseguem personalizar os fluxos de mensagens que fluem por intermédio deles. Estas redes são ativas no sentido de que os nós podem realizar computações, ou seja, realizar determinadas ações com o conteúdo dos pacotes. Em contraste, as regras de computação internas aos pacotes das arquiteturas de rede tradicionais, como a Internet, são extremamente limitadas. As redes ativas, então, permitem que os comutadores modifiquem o cabeçalho dos pacotes que trafegam por eles, tomando certas decisões de acordo com a programabilidade estabelecida nestes comutadores[TSS+97, CMK+99b].

2.2 Modelo de Redes Programáveis

Em [CMK+99b] é apresentado um modelo genérico para redes programáveis em três dimensões, como ilustrado pela Figura 2.1. Esta figura mostra o modelo de referência TCP/IP (Camadas: física; rede; transporte e aplicação) com planos de transporte, controle e gerenciamento. A divisão entre transporte, controle e gerenciamento permite ao modelo ser aplicado tanto para redes de telecomunicações quanto para a Internet.

A programabilidade dos serviços da rede é obtida pela introdução da computação interna às redes, mediante a extensão da computação realizada nos comutadores existentes.

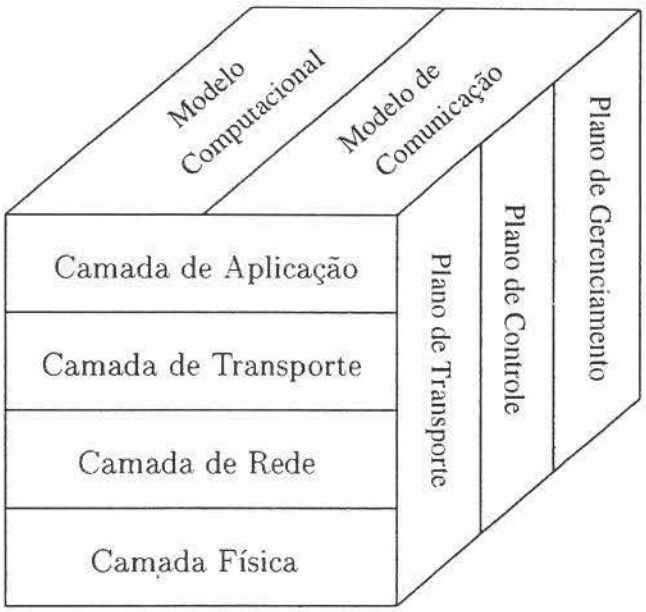

Figura 2.1: Modelo de Redes Programáveis em três dimensões.

Para distinguir a noção da arquitetura de redes programáveis da arquitetura de redes, o modelo de comunicação foi interpretado como modelo de computação. Coletivamente, os modelos de comunicação e computação tornam a rede programável.

Um modelo alternativo de redes programáveis é mostrado pela Figura 2.2 [CMK+99b], no qual pode-se ver, de maneira clara, os modelos de comunicação e computação. Os componentes chaves do modelo de computação são o ambiente de programação da rede e o *node kernel*. Os *node kernels* são sistemas operacionais que realizam gerenciamento dos recursos. Estes têm somente um significado local, ou seja, eles simplesmente gerenciam os recursos do nó, potencialmente compartilhados por múltiplas arquiteturas de redes programáveis. O Ambiente de redes programáveis é um *middleware* que provê suporte para serviços de redes programáveis distribuídas. Nesta Figura 2.2, pode-se ver, também, a separação do hardware (comutador) do software (programação e comunicação). Dois conjuntos de interfaces são ilustrados. O primeiro conjunto representa interfaces de programação da rede entre o ambiente de programação da rede e a arquitetura de redes programáveis. O segundo conjunto representa as interfaces dos nós entre os *node kernels* e o ambiente de redes programáveis. Com estas interfaces, fica evidente a necessidade de alguma padronização para permitir que as redes programáveis sejam independentes das plataformas. Esta padronização está em fase de estudo, por meio de inúmeros forums de pesquisa, como a interface de programação para redes IEEE[Bis98, P1500], Redes Ativas DARPA[Pro96], *Multiservice Switching Forum*[For00] e OPENSIG[CKMV99].

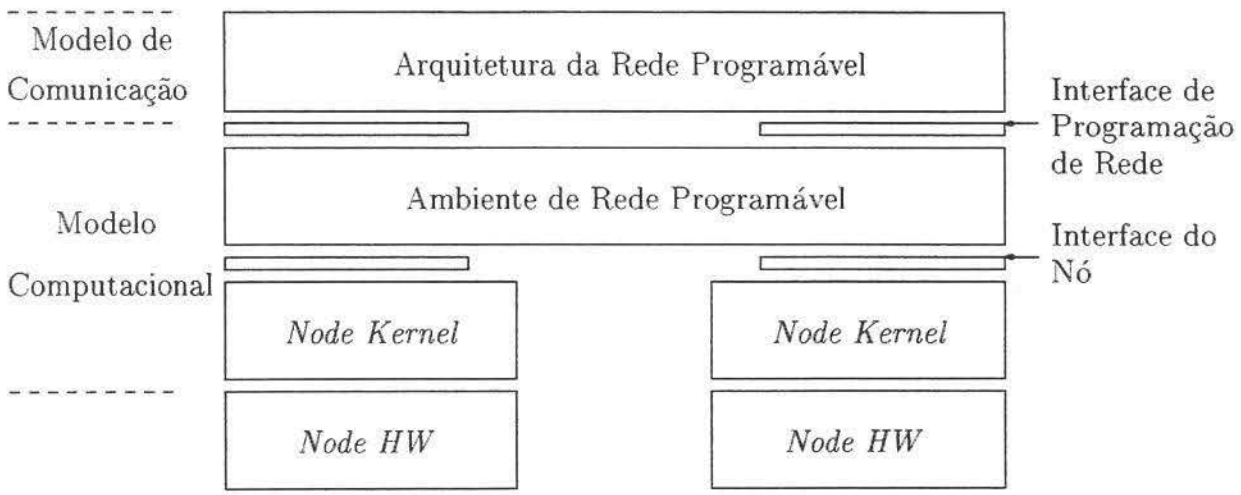

Figura 2.2: Modelo geral para Redes Programáveis.

Vários provedores de equipamentos incorporam um sistema operacional em seus comutadores e/ou roteadores para descrever funções de comunicação nos nós da rede (exemplo, os roteadores da CISCO usam o ambiente IOS e os comutadores ATM da ATML usam o micro-kernel ATMOS). Os sistemas operacionais, tipicamente, suportam uma variedade de primitivas de comunicação, as quais são proprietárias e dificultam o envolvimento de ambientes de programação de redes. O modelo computacional permite a programabilidade do modelo de comunicação, particionamento dinâmico dos recursos e uma melhor segurança dos equipamentos.

O *node kernel* representa o mais baixo nível do ambiente de programação, provendo um pequeno conjunto de interfaces nos nós. As interfaces permitem a manipulação do estado do nó (recursos do nó) e invocações do serviço de comunicação. O *node kernel* é responsável por compartilhar a computação do nó, os recursos de comunicação e, também, a segurança do nó.

O ambiente de redes programáveis suporta a construção de redes, habilitando o desenvolvimento dinâmico de serviços e de protocolos. Este ambiente permite diferentes níveis de programabilidade, metodologias de programação, tecnologias de redes e aplicações de domínio. O ambiente de redes programáveis opera sobre um conjunto de interfaces bem definidas do *node kernel*, oferecendo ferramentas distribuídas para a criação de arquiteturas de redes programáveis através do desenvolvimento de componentes de serviços distribuídos.

Neste sentido, este ambiente pode ser visto como um *middleware* entre a execução de arquiteturas de redes e o *node kernel*, como ilustrado na Figura 2.2, provê os projetistas/arquitetos de redes um ambiente e ferramentas necessárias para construção de distintas arquiteturas de redes. Assim, o ambiente de redes programáveis suporta a cons-

trução de arquiteturas de redes por meio da composição de serviços, controle de serviços, recursos e gerenciamento de estados.

O objetivo do ambiente de redes programáveis é prover o suporte necessário para dinamicamente programar novas arquiteturas de redes. Este ambiente não oferece algoritmos de redes (roteamento, *signalling*) que definem ou diferenciem a arquitetura da mesma maneira que o sistema operacional, não embutindo aplicações específicas no *kernel*. De maneira mais clara, o ambiente de redes programáveis oferece um conjunto de interfaces de programação de redes para a construção de arquiteturas de redes, o que assemelha a construção de novas aplicações usando as ferramentas para o desenvolvimento de software. Entretando, neste caso, a aplicação é a arquitetura da rede.

Em [CMK+99b], arquitetura de rede é definida como portadora dos seguintes atributos:

- *Serviços de redes*, os quais a arquitetura da rede executa como um conjunto de algoritmos de rede distribuído, que oferecem para o usuário final;
- *Algoritmos de redes*, os quais incluem transporte, *signalling/control* e mecanismos de gerenciamento;
- *Múltiplas escalas de tempo*, as quais impactam e influenciam o projeto dos algoritmos da rede;
- *Estado de gerenciamento da rede*, os quais incluem os estados que os algoritmos da rede operam (*switching*, roteamento, estado do QoS) para suportar serviços consistentes.

A programação das arquiteturas da rede são realizadas segundo o desenvolvimento de um conjunto de algoritmos. Estes algoritmos são tão diversos quanto a aplicação base que existem nos sistemas finais hoje em dia. Para suportar uma grande variedade de arquiteturas de redes programáveis é necessário que o ambiente de redes programáveis e o *node kernel* se tornem extensíveis e programáveis.

2.3 Redes Virtuais

Redes Virtuais (VN) são entidades de serviço personalizadas para determinado tipo de tráfego e/ou determinados grupos de usuários (VPN)[RSW99]. As redes virtuais são úteis para reduzir o custo e diminuir a complexidade de prover uma rede física dedicada a tráfegos distintos.

As VNs não controlam somente seu roteamento ou mecanismos de endereçamento, mas, também, qualquer aspecto básico de recursos da rede que pode ser programado.

As redes virtuais podem prover diferentes tipos de tráfego em uma mesma rede física, com controle e gerenciamento[ML97] próprio. Abaixo encontram-se alguns exemplos de tráfego, que são candidatos à configuração dinâmica dos recursos da rede:

- Tráfego destinado a um computador em particular ou a uma entidade administrativa (um *Internet Bank Web Site*);
- Tráfego associado a um grupo de usuários (departamento de recursos humanos de uma empresa);
- Tráfego (IP sobre ATM) associado a um servidor Internet, *Internet Service Provider* (ISP) (SMTP, HTTP, DNS, telnet,...) específico;
- Tráfego associado a uma aplicação específica (telefone pela Internet, vídeo, áudio, transações com cartão de crédito);
- Tráfego de diferentes qualidades e prioridades, como serviços diferenciados pela Internet.

Para configurar o tráfego, na criação ou modificação das VPNs, é necessário que os usuários especifiquem suas requisições e que o VSP concorde com elas. Assim, para criar ou modificar uma VPN tem de haver um contrato de serviço. Este contrato de serviço é conhecido como *Service Level Agreement* (SLA).

São algumas características para a criação ou modificação das VPNs requeridas pelos usuários:

- Topologia;
- Características de performance (*bandwidth*, *delay*, perdas, tolerância, tamanho do espaço de rótulo, etc);
- Atributos temporais (tempo de início e duração);
- Custo;
- Construções extras (redundância);
- Procedimentos contratuais, para aplicar penalidades.

O objetivo do projeto de VPNs é criar redes virtuais em conformidade com os requisitos dos usuários, maximizando os subsequentes usos dos recursos disponíveis de um *VPN Service Provider* (VSP).

2.4 *Switchlets*

Uma *switchlet* é um subconjunto dos recursos de sinalização e de *buffers* de um comutador (*switch*)[ML97]. *Switchlets* de um mesmo comutador podem ser usadas independentemente, e suas funções de sinalização podem diferir sobremaneira, a fim de permitir a programação da rede de comunicação. Apesar de *switchlets* terem sido implementadas única e exclusivamente com tecnologia ATM, o conceito é extensível a outras tecnologias.

Dessa forma, é possível que os recursos de um comutador físico sejam gerenciados e divididos em vários comutadores lógicos. Cada comutador lógico é uma *switchlet*. Esta técnica possibilita o uso destes recursos de maneira flexível, diferenciada e controlada dinamicamente.

A técnica de *switchlet* apresenta-se como uma nova alternativa para controlar e gerenciar redes ATM, dado que permite que diferentes arquiteturas de controle estejam operacionais simultaneamente, internas em uma mesma rede e no mesmo comutador. Arquitetura de controle é definida em [Roo98] como um conjunto de protocolos, políticas e algoritmos para controlar a rede.

O uso dos recursos pode ser feito de diferentes formas, tendo em vista:

- controlar e gerenciar redes ATM;
- utilizar diferentes serviços em um mesmo comutador ATM;
- introduzir novas arquiteturas de controle;
- permitir a construção dinâmica de redes virtuais de arbitrária funcionalidade.

O conjunto de *switchlets*, em diferentes comutadores ATM, podem ser combinadas para formar uma rede virtual ATM. Um meio físico pode ter várias redes virtuais. Cada rede virtual criada desta maneira pode, potencialmente, usar diferentes mecanismos de controle e gerenciamento, os quais são coletivamente chamados de arquitetura de controle[ML97, Isa00]. Um exemplo é mostrado na Figura 2.3, que retrata uma rede com cinco comutadores, e com três redes virtuais de diferentes tipos.

Switchlets permitem que novas arquiteturas de controle sejam introduzidas em rede de comunicação sem impactar negativamente as aplicações e serviços existentes, facilitando a mudança de gerenciamento e controle de maneira elegante. Essas arquiteturas de controle podem, ainda, ser diferentes instâncias da mesma arquitetura de controle, ou podem ser completamente diferentes umas das outras[ML98]. Por exemplo: uma rede virtual pode executar *ATM Forum signalling*[CG99, For96]; outra pode implementar a arquitetura de controle *IP switching*[ML98], enquanto outra pode, ainda, ser reservada para outro tipo de serviço.

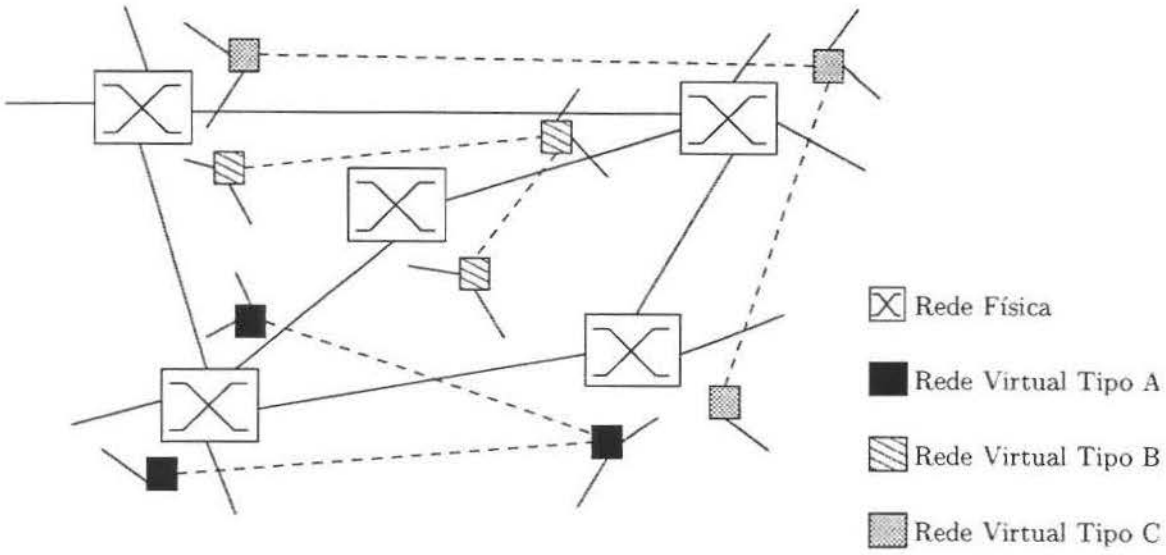

Figura 2.3: Rede Virtual ATM com diferentes arquiteturas de controle.

A ação de criar *switchlets* e combiná-las em redes virtuais é, atualmente, realizada por meio de operação humana. O processo pode, entretanto, ser automatizado, ou seja, essas redes virtuais podem ser dinamicamente criadas. A presente dissertação explora essa possibilidade, isto é, usa essa técnica para permitir a construção dinâmica de redes virtuais privadas.

O particionamento de um comutador em *switchlets* requer a especificação das *switchlets* como o subconjunto de recursos físicos disponíveis em um comutador. Na presente dissertação de mestrado, o único recurso do comutador em questão é a banda passante.

2.4.1 Criando *Switchlets*

Abaixo encontra-se uma descrição da criação de uma rede virtual ATM utilizando *switchlets*, como descrito em [ML97, Isa00].

O software de controle, por meio do *Ariel* (interface aberta de controle do comutador), controla o comutador ATM. O software de controle do comutador executando em uma estação de trabalho de propósito geral, invoca operações na interface *Ariel* em ordem para controlar e gerenciar o comutador.

O objetivo da interface *Ariel* é prover uma interface aberta e genérica de controle do comutador, com um conjunto de funções programáveis. No caso ideal, a interface *Ariel* deve prover o controle do comutador de um tipo definido. A interface de controle *Ariel* consiste nas seguintes interfaces:

- *Configuração* - a interface de configuração é primeiramente usada para obter informações da configuração do comutador para então configurá-lo;

- *Portas* - a interface da porta é provida para cada porta do comutador e negociada como uma entidade completa;
- *Conexões* - a interface de conexões é responsável por, basicamente, mapear o VPI/VCI, e negociar os procedimentos de QoS pelo índice de contexto obtido por meio da interface de contexto;
- *Contexto* - a interface de contexto empacota abstrações de QoS em simples interfaces;
- *Estatísticas* - a interface de estatísticas permite o controlador obter estatísticas do comutador e informações para avaliação;
- *Alarme* - a interface de alarme permite que o controlador seja informado quando certos eventos acontecerem no comutador.

A separação das interfaces de conexão e de contexto deve-se ao fato da interface de contexto prover as maneiras para alocar os recursos do comutador, e a interface de conexão provê as maneiras para usar estes recursos. Assim, as conexões com QoS são iniciadas por meio da interface *Ariel*, por primeiro criar o contexto, e então associar este contexto ao mapeamento do VPI/VCI.

A Figura 2.4 mostra como a interface *Ariel* (interface de controle aberta), no comutador físico, pode ser usada pelo *Switch Divider Controller* (prospero) para criar várias *switchlets*. O Prospero aloca subconjuntos de recursos físicos do comutador (*switchlets*), e os torna disponíveis para o software de controle da *switch* através da interface *Ariel*. O software de controle do comutador, de um tipo particular, irá controlar a *switchlet* através da sua interface *Ariel*, da mesma maneira como seria no comutador físico. Como exemplo, a Figura 2.4 mostra três possíveis arquiteturas de controle conhecidas como *Hollowman*[Roo97], *IP Switching*[Net96] e *ATM Forum's PNNI*[For96], em que cada uma utiliza uma das três redes virtuais mostradas na Figura 2.3.

A interface *Ariel* no comutador físico pode ser usada pelo *Switch Divider Controller* para criar várias *switchlets*. O *Switch Divider Controller* aloca um subconjunto de recursos físicos do comutador e os torna disponíveis para o software de controle por intermédio da interface *Ariel*. O software de controle de um tipo particular irá controlar a *switchlet* por invocar a interface *Ariel* da *switchlet*. As *switchlets* podem ser combinadas em redes virtuais de certos tipos ou arquiteturas de controle.

O *Switch Divider Controller* tem que conhecer a capacidade física do comutador e somente permitir que novas *switchlets* sejam criadas quando os recursos deste comutador não tenham sido esgotados. A alocação de recursos no comutador para *switchlets* não envolve nenhuma invocação (ou alocação) física[ML97]. O *Switch Divider Controller*, entretanto, aloca em sua representação interna a capacidade física do comutador e usa isso para

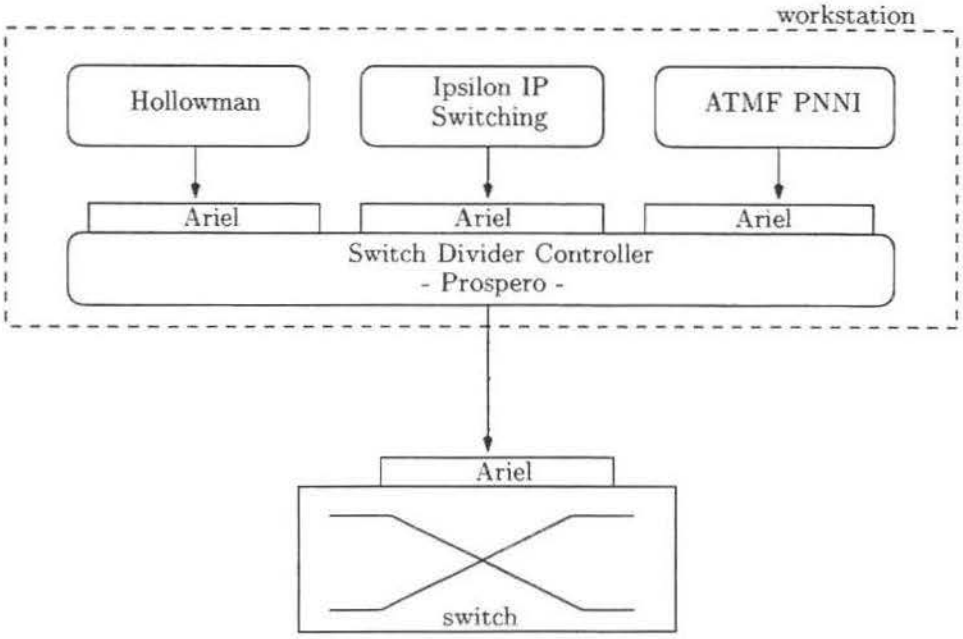

Figura 2.4: Criando *Switchlets*.

policiar novas invocações na interface *Ariel* da *switchlet*. Uma vez que a conexão tenha sido estabelecida na *switchlet*, o *Switch Divider Controller* tem de confiar nos mecanismos de policiamento do comutador físico, para assegurar que conexões de uma *switchlet* não interfiram em outras *switchlets*.

Em primeira instância, o *Switch Divider Controller* provê uma interface de configuração, que pode ser usada por operadores humanos para criar *switchlets* e redes virtuais. É claro que o interesse maior é a criação dinâmica de redes virtuais sob-demanda por sistemas de software. Este processo, porém, não é simples.

Um dos fatores que dificultam, atualmente, a criação dinâmica de redes virtuais é a inexistência de metodologias que permitam otimizar os recursos alocados nas redes. Neste sentido, o presente trabalho, explora este assunto, definindo uma metodologia para criação dinâmica de redes virtuais sob-demanda, visando a maximização do uso dos recursos da rede e a minimização do custo neste processo de alocação.

Recursos utilizados por uma *switchlet*

Como já mencionado anteriormente, o subconjunto de recursos de um comutador é conhecido como *switchlets*. O particionamento do comutador em *switchlets* requer a especificação da *switchlet* em termo do subconjunto dos recursos físicos disponíveis neste

comutador. Os recursos do comutador que precisam de especificação são:

- Portas;
- Espaço de VPI/VCI;
- Banda Passante;
- Espaço de Buffer;

Portas e espaços de VPI/VCI constituem recursos de conexão do comutador e podem ser particionados em vários níveis de granularidade. Banda Passante e espaço de buffer são combinados para representar a capacidade.

Como exemplo, a especificação da *switchlet* pode consistir de um número de portas requeridas, e para cada porta a seguinte informação:

- Quantidade de VPIs requeridas;
- Quantidade de VCIs por VPI requerida;
- Categoria de serviço requerida;
- Capacidade por categoria de serviço requerida.

2.4.2 Criando uma rede virtual usando *switchlets*

A Figura 2.5 ilustra os passos envolvidos na aquisição de uma rede virtual por um sistema de controle.

O *Divider Controller* informa ao *Network Builder* sobre sua existência, assim como sua capacidade de comutação[ML97]. O *Divider Controller* também exporta uma interface (*Ariel*), a qual permite criar e destruir as *switchlets*.

Os passos envolvidos para a criação da rede virtual são: (1) o sistema de controle localiza o *Network Builder* usando o *Trader*; (2) o sistema de controle fornece as especificações desejadas para criar a rede virtual para o serviço *Network Builder*. As especificações da rede podem ser fornecidas pela saída de outro serviço, ou ser providas por um usuário sob-demanda.

O *Network Builder* localiza o *Divider Controller* em cada nó da rede (3) e requer uma nova *switchlet* em cada um. Após criar a *switchlet*, o *Divider* retorna uma nova interface *Ariel* da *switchlet* criada para o *Network Builder*. As interfaces das *switchlets* são, então, passadas de volta para o sistema de controle, o qual pode fazer as invocações nas *switchlets* diretamente (4), permitindo controlar diretamente os recursos físicos dos comutadores.

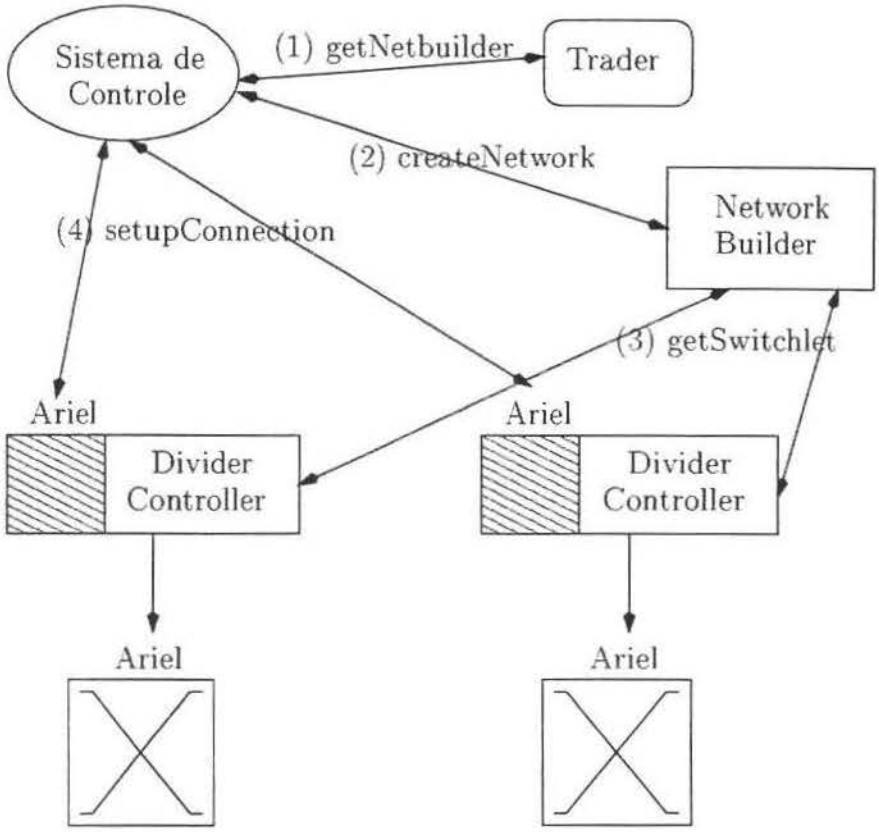

Figura 2.5: Aquisição de uma rede virtual sob-demanda.

2.5 Redes *Spawning*

Em razão do fato de o desenvolvimento de arquiteturas de redes ser muitas vezes manual, consumindo, assim, tempo e custo no processo, técnicas são criadas para fazer com que as redes sejam capazes de proliferar, criar (*spawning*) e gerenciar arquiteturas de redes virtuais *on-the-fly*.

Surge, então, uma técnica de programação de redes conhecida como *Spawning*. Segundo o dicionário de inglês contemporâneo *Longman*[Ltd95], *spawning* significa criação. Na comunidade de redes de computadores *spawning* é uma técnica de programação de redes. Esta técnica descreve o processo de criar, desenvolver e gerenciar automaticamente novas arquiteturas de redes em resposta à demanda dos usuários[LC97].

Spawning introduz um novo paradigma para automatizar o processo de ciclo de vida na criação, desenvolvimento e gerenciamento de arquiteturas de redes, utilizando redes programáveis capazes de criar (*spawning*) redes virtuais "filhas"com seus próprios controles e sistemas de gerenciamento. A rede filha opera em um subconjunto dos recursos da

sua rede "pai" e de maneira isolada de outras redes virtuais. Redes filhas suportam o acesso controlado para comunidades de usuários com determinada conectividade, proteção e QoS[CMK+99a].

Pode-se fazer um paralelo entre o conceito de *spawning* e o conceito de processos pais e filhos em um sistema operacional. Um processo A pode criar vários outros novos processos, os quais são filhos de A, ao passo que o processo A é conhecido como processo pai. Os processos filhos herdam do processo pai seus atributos, tipicamente executados no mesmo hardware[LC97].

Neste sentido, as redes *spawning* são vistas, no conceito de redes programáveis, como tendo a capacidade de criar não apenas processos, mas complexas arquiteturas de redes [LC97].

2.5.1 *Genesis Kernel*

Abaixo é descrito brevemente o projeto de *spawning*, denominado *Genesis*[CMK+99a].

O *Genesis virtual network kernel*[CMK+99a] representa a próxima geração na tentativa de desenvolver uma construção de redes programáveis nas pesquisas em programação aberta em banda larga e rede móveis. *Genesis Kernel* é um sistema operacional de rede virtual capaz de proliferar, criar (*spawning*) e gerenciar arquiteturas de redes virtuais *on-the-fly*. O *Genesis Kernel* tem a capacidade de criar (*spawn*) arquiteturas de redes filhas que podem suportar pacotes de sinalização, serviços de comunicação, controle da QoS e gerenciamento da rede em comparação com a arquitetura de rede do seu pai. O *kernel* da rede virtual pai tem a capacidade de criar "redes filhas". A rede filha opera de maneira isolada no subconjunto do conjunto básico de recursos e da topologia da "rede pai", suportando o acesso controlado a um conjunto de usuários com conectividade específica, proteção e QoS.

O *Genesis Kernel* é governado por um conjunto de princípios de projeto:

- Separação

  Redes virtuais filhas operam isoladamente, com seu tráfego sendo transportado de maneira segura e independente das outras redes. Quando a rede filha é criada, ela tem completa liberdade para gerenciar e controlar seus recursos de maneira autônoma baseada na sua arquitetura instanciada. Este é o princípio da separação.

- Aninhamento(*Nesting*)

  A rede filha herda a capacidade para criar outras redes virtuais, criando a noção de ninhos (*nested*) de redes virtuais. Isto é, toda rede virtual filha pode ser pai da sua própria rede filha.

- Herança

  A rede filha pode herdar componentes da arquitetura da rede pai. A rede virtual filha pode herdar qualquer aspecto da arquitetura do seu pai, representada pelo conjunto de objetos de transporte, controle e gerenciamento da rede. Em contraste, a composição da rede filha pode ser completamente distinta do seu pai, não usando a herança.

O *Genesis Kernel* age como um alocador de recursos, decidindo entre requisições de conflito feitas pela criação de redes virtuais.

A capacidade chave do *Genesis* está na habilidade de suportar a criação e o desenvolvimento dinâmico de arquiteturas de redes virtuais por meio do processo de:

- *profiling* - o qual captura o perfil de projeto da arquitetura da rede virtual, mantido em um *profiling script*;
- *spawning* - o qual executa o *profiling script* para criar a topologia da rede e endereçar o espaço e a ligação do controle e gerenciamento do transporte de objetos à infraestrutura da rede física;
- *management* - o qual gerencia a arquitetura da rede virtual e os seus recursos.

Quando a rede virtual é criada, o núcleo separado da rede virtual é criado pela rede pai no interesse do filho. O ambiente de transporte do núcleo da rede virtual filha é dinamicamente criado pela visualização do ambiente de transporte do pai. O *profiling* e o *spawning* da rede filha é controlado pela sua rede virtual pai. Em contraste, o núcleo da rede virtual filha é responsável por gerenciar sua própria rede.

2.6 Roteadores Inteligentes na Internet

Esta linha de pesquisa em redes programáveis emprega a programabilidade nos roteadores da rede seguindo a escola de pensamento de redes ativas[TSS+97].

Um dos conceitos básicos de projetos da Internet é fornecer toda a inteligência nos pontos finais de comunicação, como os sistemas operacionais e as aplicações. O único propósito das redes (camadas física e enlace de dados do modelo ISO/OSI[Tan97]) é servir, primeiramente, como um mecanismo rápido para troca de pacotes IP entre aplicações. Este fato tem vantagens sobre sistemas onde a maior parte da inteligência é provida pelas redes, e é muito mais fácil prover novas aplicações do que prover novos mecanismos internos às redes. O mecanismo do nível de aplicação pode ser implementado espontaneamente; ao contrário, mudanças em mecanismos internos às redes têm de ser planejadas,

modeladas e finalmente implementadas por provedores de equipamentos de redes. Assim, com a maioria da inteligência nos roteadores (camada de redes do modelo ISO/OSI), novos serviços com novas características podem ser introduzidas facilmente sem nenhuma necessidade para modificar a rede básica.

Surge, então, outro problema: as aplicações, em um mesmo elemento de rede, não se conhecem, ou não sabem quais serviços são mais cruciais para os usuários finais. Além disso, nas aplicações se desconhece a importância do usuário interno à comunidade e/ou corporação. Por exemplo: como os empregados da empresa que "surfam na Web" vão saber que devem reduzir a utilização dos recursos da rede consumidos por eles, porque o departamento de *marketing* tem uma sessão de telefones IP com grande utilização e considerável importância para a empresa?

Para superar estes problemas (difícil mudança nos mecanismos de rede e a falta de conhecimento das aplicações), foi proposta em [RSW99] uma arquitetura que desacopla aplicações de mecanismos de alocação de recursos do núcleo da rede. Nesta arquitetura, a função de alocação de recursos é provida pelos roteadores que conectam as LANs com a Internet. Estes roteadores (Roteadores Inteligentes-RI) provêem funções programáveis em adição ao seu roteamento e às suas funções de tráfego.

Podem-se, rapidamente, programar novos critérios para classificação de tráfego, e inicializar tratamentos para a associação das redes virtuais com estas classes de tráfego. Além disso, foi proposto uma interface de programação aberta para inicializar, controlar e gerenciar redes virtuais de possíveis localizações remotas.

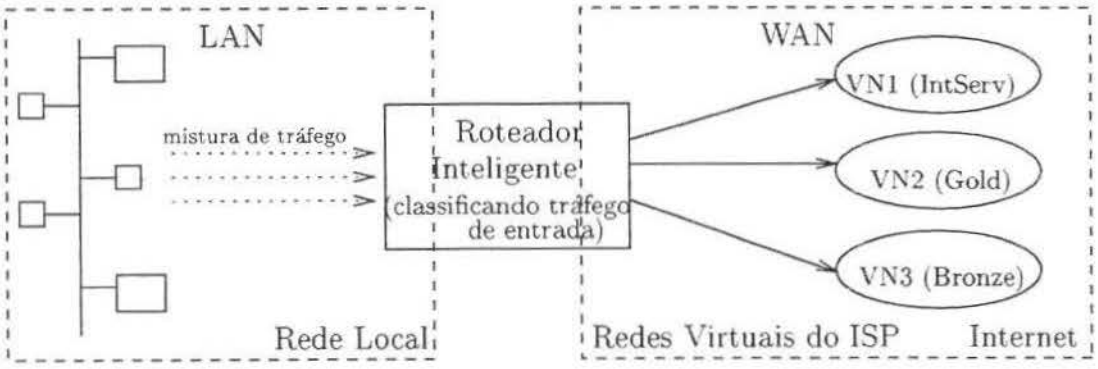

Figura 2.6: A configuração do Roteador Inteligente.

A Figura 2.6 mostra esta configuração. O RI é conectado ao ISP, o qual pode oferecer várias redes virtuais aos seus usuários. Cada uma destas redes virtuais pode ter uma característica muito específica de QoS. Estas redes virtuais podem compartilhar o mesmo *hardware* físico ou podem, atualmente, usar as composições de acesso físico, talvez per-

tencentes a diferentes ISPs. Em adição às redes virtuais que são providas pelo ISP, o RI pode implementar seu próprio conjunto de redes virtuais, cada um com um único valor adicional de funções e interfaces.

O RI é responsável por atribuir os recursos providos pelo ISP para os vários fluxos de pacotes IP que são emitidos das aplicações processadas na LAN. Este processo de alocação de recursos é governado por uma política, que é usualmente definida pelo administrador da LAN.

Em ordem, para fazer as decisões, o RI precisa analisar o tráfego que recebe. O endereço IP de origem pode ser usado para determinar o usuário que está associado a este tráfego.

A customização de tráfego na Internet pode ser classificada em duas categorias: aumento de serviços baseado no IPv4 corrente da Internet e uso dos serviços existentes para construir customizados *frameworks* para as redes baseadas, também, no IPv4.

A primeira categoria inclui a arquitetura de serviços integrados (IntServ) e arquitetura de serviços diferenciados (DiffServ). A arquitetura de serviços diferenciados é preferida por escalar melhor.

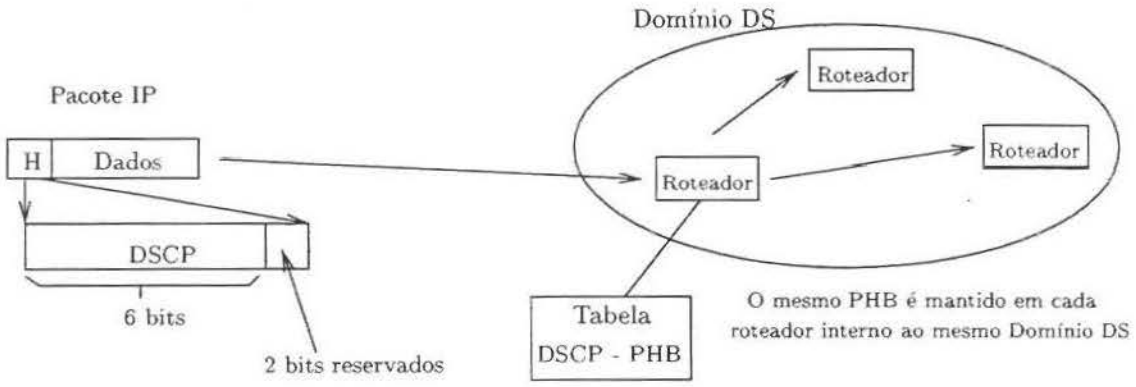

Figura 2.7: Arquitetura de Serviço Diferenciada.

A arquitetura de serviço diferenciado classifica o tráfego na rede. Todo roteador que faz parte do mesmo domínio DS (*Differentiated Services*) tem que implementar o mesmo *Per Hop Behavior* (PHB), correspondente a um específico *Differentiated Services Code Point* (DSCP) embutido no cabeçalho dos pacotes de entrada, Figura 2.7. Seis bits são atribuidos para o DSCP, os quais são usados para especificar o tipo de serviço[Nic98]. Um inteiro de 32-bits sem sinal é mantido nos roteadores, os quais são reservados para definir o PHB. O mapeamento entre o DSCP e o PHB é definido e mantido em cada roteador. Quando um roteador recebe um pacote IP, ele extrai o DSCP do cabeçalho do pacote,

olha para a tabela de mapeamento entre o DSCP e o PHB, determina qual PHB deve ser usado, e processa o pacote de acordo com a política definida no PHB.

Serviços Diferenciados podem contribuir para a implementação de redes virtuais. O valor do DSCP em cada pacote IP atribuído para a rede virtual pode definir o PHB que é parte de um tratamento especial para as redes virtuais.

Capítulo 3

Fluxos em Rede

Neste capítulo, apresentam-se alguns conceitos fundamentais em Fluxos de Rede[AMO93, CLR90].

3.1 *Node Splitting*

Em estruturas de dados, as redes são representadas por grafos. Frequentemente, é necessário fazer transformações em grafos, tendo em vista simplificar as redes, para mostrar equivalências entre diferentes problemas de redes, ou modelar estes problemas nas formas requeridas por códigos computacionais. Esta seção descreve a transformação de rede conhecida como *node splitting*. Esta transformação foi utilizada para modelar as *switchlets*.

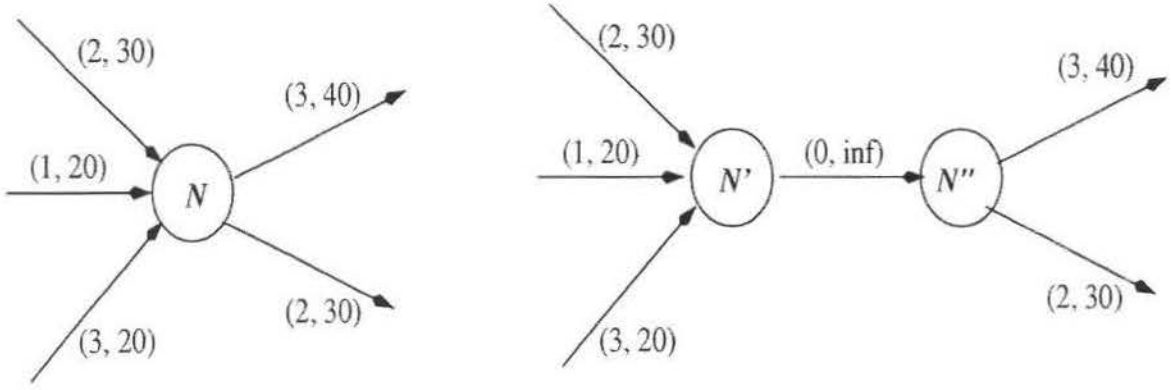

Figura 3.1: Exemplo de *node splitting*. Em cada arco é associado um par ordenado representando, respectivamente, seu custo e sua capacidade.

A técnica *node splitting* consiste em dividir cada nó N do grafo em dois nós N' e N''. O nó N' contém os mesmos arcos de entrada do nó N e apenas um arco de saída com

custo zero e capacidade infinita ligando N' a N''. O nó N'' contém apenas o arco de entrada (N', N'') e os mesmo arcos de saída do nó N (Figura 3.1).

Assim, esta transformação troca cada arco original (N, K) pelo arco (N'', K') de mesmo custo e mesma capacidade. É possível mostrar a correspondência de um fluxo na rede original com um fluxo na rede transformada. Note que os fluxos em ambas as redes possuem o mesmo custo.

A transformação *node splitting* permite modelar várias aplicações em problemas práticos. Dessa forma, o presente trabalho utiliza-se desta transformação para modelar o particionamento de um comutador em diversas *switchlets* sujeitas ao compartilhamento da mesma banda passante.

3.2 Problemas clássicos de fluxos em rede

Existem três problemas clássicos de fluxos em rede, a saber: achar os caminhos de custo mínimo (*shortest paths*); achar os caminhos de fluxo máximo (*maximum flow paths*) e achar os caminhos de fluxo de custo mínimo (*minimum cost flow paths*).

O problema de caminhos de custo mínimo e caminhos de fluxo máximo são complementares, e são subproblemas do problema de caminhos de fluxo de custo mínimo.

Estes dois problemas diferem no sentido de capturarem diferentes aspectos do problema de fluxo de custo mínimo:

- Caminhos mínimos modelam custos nos arcos e não a capacidade;
- Fluxo máximo modelam a capacidade nos arcos e não o custo.

Em caminhos mínimos, o objetivo é escolher o caminho cujo custo seja mínimo. Em caminhos de fluxo máximo o objetivo é enviar a maior quantidade de fluxo por um caminho, sem exceder a capacidade de nenhum arco. Já em caminhos de fluxo de custo mínimo o objetivo é escolher caminhos analisando duas métricas, custo e fluxo, ou seja, escolher caminhos onde o custo seja mínimo, mas a capacidade seja máxima para enviar a maior quantidade de fluxo.

3.3 Problemas de *Multicommodity Flow*

Esta seção introduz o problema de *multicommodity flow*. O objetivo deste problema é atribuir caminhos na rede para várias *commodities* de forma a minimizar o custo ou o retardo. Segundo o dicionário de inglês contemporâneo *Longman*[Ltd95], *commodity* significa: mercadoria; produto; utensílio e serviços. No contexto do presente trabalho *commodity* representa VPNs.

Considere uma rede direcionada G com N nós e A arcos. Para cada arco é definida uma capacidade e um custo não negativo. São fornecidos, nesta rede, K pares de nós, cada um com um fluxo. Estes pares de nós definem a origem e o destino destes fluxos, os quais são chamados de *commodities* ou VPNs. Deseja-se minimizar o custo de conduzir estas *commodities* através da rede G.

O problema de *multicommodity flow* consiste em escolher caminhos para cada fluxo, de um nó origem para um nó destino, sem exceder as restrições de capacidade dos arcos. Define-se custo de um fluxo como sendo a soma dos custos dos arcos do caminho multiplicado pela quantidade de fluxo sobre esses arcos.

Se caso existir somente uma *commodity*, o problema reduz-se a um problema de encontrar o caminho mais curto (*shortest path algorithm*). Se for permitido que o fluxo se divida em diferentes caminhos, o problema pode ser resolvido encontrando os caminhos mínimos usando o valor do custo nos arcos. Se este caminho não tiver capacidade suficiente para satisfazer toda a demanda de fluxo, um segundo caminho mínimo precisa ser encontrado para satisfazer a demanda de fluxo restante. Se ainda houver demanda de fluxo, um terceiro caminho mínimo deve ser encontrado e assim por diante, até que todo o fluxo seja transportado do nó de origem para o nó de destino. Por outro lado, se não for permitido que o fluxo seja dividido por diferentes caminhos, o problema pode ser resolvido usando, também, o algoritmo de caminho mínimo, em que somente arcos com suficiente capacidade são utilizados. Neste caso, também, os caminhos mínimos são calculados usando o valor do custo dos arcos.

Quando K *commodities* são fornecidas, o problema não pode ser resolvido simplesmente usando o algoritmo de caminhos mínimos para cada *commodity*. Isto porque as *commodities* compartilham os recursos. Agora, neste problema, a atribuição de um caminho para uma *commodity* influencia na atribuição de caminhos para as demais *commodities*, o que torna o *multicommodity flow* um problema NP-difícil.

O problema de *multicommodity flow* pode, também, ser formulado usando para cada *commodity* uma capacidade e um custo no arco, reduzindo o problema para K únicas *commodities*, ou seja, as *commodity* não influenciam umas as outras. Este problema pode ser resolvido usando K aplicações de um algoritmo para encontrar o caminho de fluxo de custo mínimo.

Desta forma, o problema de *multicommodity flow*, utilizado no presente trabalho, possui as seguintes características:

- Todas as *commodities* compartilham a capacidade dos arcos, ou seja, todos os arcos têm somente uma capacidade;
- Todas as *commodities* compartilham o custo dos arcos, ou seja, todos os arcos têm somente um custo;

- O custo dos arcos são constantes. Isto significa que mesmo que sobre somente uma unidade de capacidade nos arcos, o seu custo não será alterado. Para outras aplicações a última parte da capacidade terá um custo maior se for usada (por exemplo, em problemas de enfileiramento).

O modelo geral do problema de *multicommodity flow* permite dividir o fluxo de uma *commodity* em mais de um caminho, mas existem algumas aplicações que não permitem que este fluxo seja dividido, como, por exemplo, nas ligações telefônicas. Nestas aplicações somente um caminho é atribuído para cada *commodity*, ou seja, o fluxo desta *commodity* só pode passar por um determinado caminho, tendo, assim, um problema inteiro.

O exemplo abaixo mostra que a solução para um mesmo problema é diferente, se for usado o modelo de programação linear, ou se for usado o modelo de programação inteira.

Exemplo: A Figura 3.2 mostra uma rede direcionada. A capacidade e os custos são fornecidas como pares (custo, capacidade) para cada arco. Serão conduzidas duas *commodities* nessa rede:

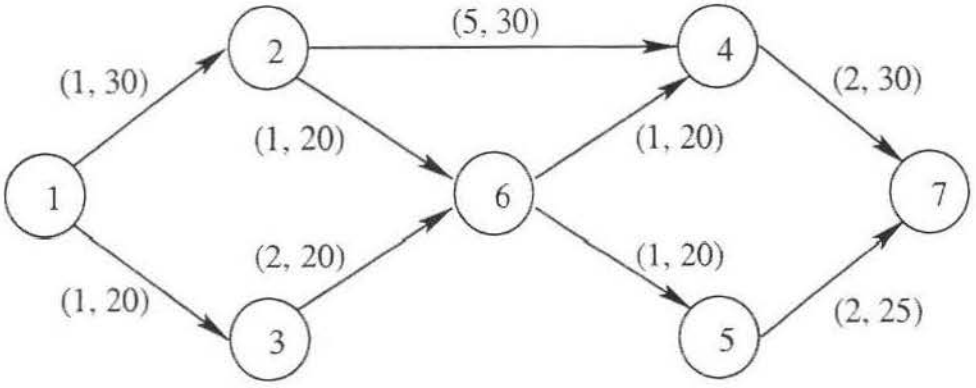

Figura 3.2: A figura mostra uma rede direcionada com custos e capacidades nos arcos.

- *Commodity 1:* Nó de origem: 1; nó de destino: 7 e fluxo de 30;
- *Commodity 2:* Nó de origem: 1; nó de destino: 7 e fluxo de 15;

Resolvendo o problema linear, que permite a divisão do fluxo, obtem-se a seguinte função ótima:

- *Commodity 1:*
  - enviar um fluxo de 20 no caminho: $[1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7]$.
  - enviar um fluxo de 10 no caminho: $[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7]$.

- *Commodity 2:*
  - enviar um fluxo de 10 no caminho: $[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7]$.
  - enviar um fluxo de 5 no caminho: $[1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 7]$.

Dessa forma, o custo total de conduzir as duas *commodities* através da rede é 260.

Resolvendo o problema inteiro, não permitindo a divisão do fluxo, obtem-se a seguinte função ótima:

- *Commodity 1:*
  - enviar um fluxo de 30 no caminho: $[1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 7]$.
- *Commodity 2:*
  - enviar um fluxo de 15 no caminho: $[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7]$.

Neste caso, o custo total de conduzir as *commodities* é de 330.

As soluções para o problema inteiro são também soluções factíveis para o problema linear. O espaço de soluções do problema inteiro é um subconjunto do espaço de soluções do problema linear. Note que se o problema for inteiro não implica que os fluxos e/ou as capacidades sejam inteiras, mas implica que a atribuição do caminho seja inteira.

De forma geral, o objetivo é minimizar o custo total para trafegar os fluxos (*commodities*) na rede. Este tipo de problema em fluxos de rede tem várias aplicações em diferentes campos, como: problemas em transporte; problemas em telecomunicações e problemas em produção. O presente trabalho utiliza uma solução baseada em *multicommodity flow* para explorar o problema de alocação de recursos (banda passante) em redes para diferentes VPNs, ou seja, objetiva-se determinar os melhores caminhos para acomodar todas as VPNs visando a otimização dos recursos da rede.

3.4 *Branch-and-bound*

O algoritmo *branch-and-bound* é um método eficaz para resolver problemas de otimização NP-difícil. Para resolver um problema, o algoritmo *branch-and-bound* analisa o espaço de soluções, que é um produto de todos os caminhos para cada par de nós origem e destino, como uma estrutura de árvore, na qual cada nó corresponde a uma variável de decisão do problema, e cada nó folha desta árvore é uma solução factível do problema. Cada arco dessa árvore representa a atribuição de um caminho para um determinado serviço k. O nível k da árvore possui todos os nós correspondentes ao serviço k, assim sendo, esta árvore tem altura igual ao número total de serviços. Os nós correspondentes ao serviço

k são ordenados do caminho de menor custo ao de maior custo, pois intuitivamente o caminho de menor custo para um serviço k é o mais provável de pertencer à solução ótima.

O método *branch-and-bound* tenta reduzir o número de nós podando essa árvore de soluções. Isso é feito por meio do cálculo de um limite inferior para cada nó da árvore e comparando este valor com a melhor solução corrente. Se o limite inferior ultrapassar o valor da melhor solução, então esse nó e todos os seus descendentes são podados, economizando tempo. O limite inferior é calculado pelo método de geração de colunas, que consiste em gerar colunas no problema linear somente quando for necessário. Esse procedimento faz com que o problema linear seja muito menor e mais fácil de resolver. Resumindo, o método *branch-and-bound* encontra a solução ótima de um problema complexo de programação linear resolvendo diversos problemas menores.

A fim de ilustrar o funcionamento deste algoritmo trabalha-se a seguir um problema cujo objetivo é minimizá-lo. Considerando duas *commodities* onde a primeira possui cincos caminhos possíveis e a segunda possui três caminhos possíveis, o espaço de soluções para este problema é apresentado na Figura 3.3. Assim, fica claro que, mesmo para um pequeno número de *commodities*, o espaço de soluções cresce exponencialmente.

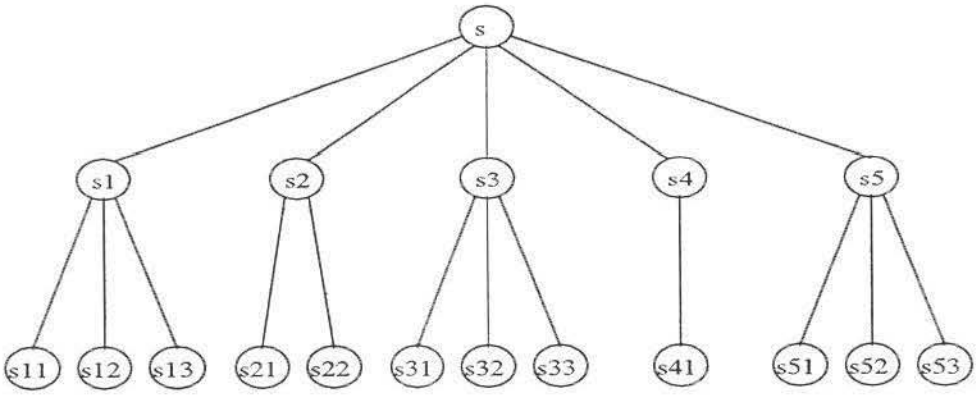

Figura 3.3: A figura mostra o espaço de soluções.

Resolvendo um problema utilizando *branch-and-bound* em uma primeira etapa é encontrada uma solução com valor da função objetiva de x no nó folha $s11$, na Figura 3.3. Dando continuidade à análise do resto do espaço de soluções, com o intuito de encontrar uma melhor solução, tem-se que, por exemplo, ao examinar o nó $s3$ encontra-se um limite inferior para esta parte da árvore de soluções. Se este limite inferior encontrado for maior ou igual à melhor solução encontrada x, o nó $s3$ e todos os seus descendentes são podados. Isso significa que não será preciso examinar os nós $s31$, $s32$, $s33$, pois não será encontrada uma solução melhor nestes nós.

Este algoritmo é esboçado pelo pseudocódigo da Figura 3.4.

```
função BranchAndBound(c)
para k^th caminhos mínimos k = 1, 2, ... faça
    atribuir k^th caminho mínimo para a commodity c
    calcular o limite inferior da corrente atribuição
    se o limite inferior < melhor solução encontrada
      se c = número de commodities
        nova melhor solução encontrada
      senão
        BranchAndBound(c + 1)
    liberar o k^th caminho mínimo da commodity c
```

Figura 3.4: A Figura mostra o pseudocódigo do algoritmo de *branch-and-bound*.

Assim, é importante que este algoritmo seja capaz de podar a árvore de busca consideravelmente; caso contrário será necessário examinar todos os espaços de soluções na árvore, o que é, muitas vezes, exponencial.

3.5 Algorítmo de Dijkstra para achar os K melhores caminhos

Para obter a solução de vários problemas em fluxos de rede é aplicado o algoritmo clássico de Dijkstra com o objetivo de achar os caminhos mínimos, de acordo com uma determinada métrica, de um nó de origem para todos os outros nós da rede. O problema de encontrar os K caminhos mínimos de um nó de origem para um nó de destino é uma generalização do problema clássico de Dijkstra[SS99, MPS98]. Esta generalização é mostrada pelo pseudocódigo da Figura 3.5, explicado abaixo.

São descritas, primeiramente, as variáveis usadas e, em seguida, é explicado como o algoritmo funciona. Quando o algoritmo começa, os seguintes parâmetros são passados:

- Um grafo $G = (N, A)$ onde N é o conjunto de nós e A é o conjunto de arcos;
- Um inteiro K, que representa a quantidade de caminhos para encontrar;
- Um nó de origem s e um nó de destino t;

Parâmetros usados no algoritmo:

- $count_i$ é o número de vezes que o nó i foi visitado, ou seja, o número de caminhos que usam este nó;

- elm é um índice que corresponde ao nó visitado;
- h é um *array* que amarra o elemento elm ao correspondente nó visitado;
- π_{elm} é a distância do nó origem para o nó h (elm);
- X é o *heap* contendo os elementos. A distância do nó de origem para o nó do elemento é a chave do *heap*;
- P^j é o j^{th} caminho mínimo;
- ξ_j é o elemento anterior do nó j do caminho;
- c_{ij} é o custo do arco (i, j).

No algoritmo clássico de Dijkstra, para encontrar os caminhos mínimos, a distância dos nós inseridos no *heap* para o nó de origem é usada como chave. Em cada passo, o nó de distância mínima é extraído do *heap* e as distâncias dos seus vizinhos são atualizadas. Cada nó é, então, visitado somente uma vez. Quando o *heap* fica vazio, as distâncias do nó de origem para todos os outros nós da rede são determinados.

No algoritmo generalizado de Dijkstra, os nós podem ser visitados mais de uma vez. Isto acontece quando o nó é usado em mais de um caminho, significando que não pode ser definida uma distância para cada nó, assim são introduzidos elementos. Um elemento é criado cada vez que o nó é visitado. Pelo elemento é possível achar o nó através do *array* h. Desta maneira, cada elemento tem uma distância do nó de origem e são inseridos no *heap* com suas distâncias como chaves e em cada passo o elemento de distância mínima é extraído.

Tal como o algoritmo clássico de Dijkstra, o algoritmo generalizado somente trabalha em redes com custos não-negativos nos arcos. Isto porque, quando ambos os algoritmos visitam um nó, o caminho mínimo para aquele nó terá sido encontrado. Se forem permitidos arcos com custos negativos, esta afirmação não vai ser sempre verdadeira.

Cada vez que o elemento elm é encontrado, o contador $count_i$ do correspondente nó i é incrementado, e o ponteiro ξ_{elm} é atribuído ao elemento anterior do caminho. Quando o nó de destino t é visitado, os ponteiros ξ podem ser usados para traçar o caminho do nó de origem para o nó de destino. Quando o nó de destino é visitado K^{th} vezes (quando $count_t$ alcança K), K caminhos mínimos tem sido encontrados e o algoritmo termina. Se o *heap* ficar vazio durante os passos, significa que não existe os K caminhos mínimos. Neste caso, o algoritmo termina com somente $count_t$ caminhos mínimos.

Abaixo é descrita a complexidade de tempo e a complexidade de memória usada no algoritmo generalizado de Dijkstra. Assumiu-se que todas as atribuições e cálculos são executados em um tempo constante. Foi implementado um *heap* binário que realiza

```
count_i ⟵ 0, ∀i ∈ N
elm ⟵ 1
h(elm) ⟵ s
π_elm ⟵ 0
X ⟵ {elm}
P^j = 0, ∀j = 1, ..., K
enquanto (count_t < K) e (X ≠ 0) faça
    e ⟵ ExtractMin(X)
    i ⟵ h(e)
    count_i ⟵ count_i + 1
    se (i = t) então
      j ⟵ i
      enquanto (j ≠ s) faça
        P^count_i ⟵ P^count_i ∪ {h(j)}
        j ⟵ ξ_j
      P^count_i ⟵ P^count_i ∪ {s}
    se (count_i ≤ K) então
      para todos os arcos (i, j) ∈ A que não geram ciclo faça
        elm ⟵ elm + 1
        π_elm ⟵ π_e + c_ij
        ξ_elm ⟵ e
        h(elm) ⟵ j
        Insert(X, elm)
```

Figura 3.5: A Figura mostra o pseudocódigo do algoritmo generalizado de Dijkstra.

inserções e extrações em tempo logaritmo. No pior caso, o comando de repetição `while` é executado $K \times m$ vezes, onde K é o número de caminhos mínimos para serem encontrados e m é o número de arcos no grafo.

Uma comparação do novo nó com todos os outros nós é feita para checar se o ciclo é gerado, a complexidade desta checagem é de $O(n)$. Sendo que a checagem por ciclos é executada no máximo $K \times m$ vezes, tem-se uma complexidade de $O(Kmn)$. A operação `ExtractMin` e `Insert` no *heap* são executadas também no máximo $K \times m$ vezes desde que tenham uma complexidade de $O(logKm)$, esta não altera a complexidade no pior caso. A estrutura de repetição interna `while` é executada K vezes, fornecendo, assim, uma complexidade de $O(Kn)$. Esta, também, não influencia na complexidade deste algoritmo. Assim, a complexidade do algoritmo generalizado de Dijkstra é de $O(Kmn)$.

O *heap* contém no máximo $K \times m$ elementos em qualquer tempo durante a execução. Os elementos são todos de tamanho constante, assim a complexidade de memória usada no *heap* é $O(Km)$. Os *arrays* h e π contêm $K \times m$ ponteiros, os quais fornecem uma complexidade de memória utilizada de $O(Km)$ para estes *arrays*. São armazenados,

também, os K caminhos mínimos encontrados pelo algoritmo, desde que cada caminho contenha no máximo n nós a complexidade do total de memória utilizada é de $O(kn)$. Assim a complexidade total de memória utilizada pelo algoritmo generalizado de Dijkstra é de $O(km + kn)$.

Capítulo 4

Qualidade de Serviço e Engenharia de Tráfego

Este capítulo expõe o conceito de qualidade de serviço (QoS) e o conceito de engenharia de tráfego em redes de dados.

4.1 Qualidade de Serviço

O tráfego nas redes tem aumentado exponencialmente em função do número de usuários e do número de aplicações. Em adição, a capacidade dos comutadores e dos enlaces tem aumentado de maneira significativa. O aumento na capacidade das redes, entretanto, não é suficiente para acomodar a presente demanda por tráfego.

O problema é que o tráfego não tem, simplesmente, aumentado seu volume, mas tem mudado sua característica, sua natureza. Diferentes aplicações de redes estão exigindo diferentes requerimentos operacionais que demandam diferentes serviços de redes.[SQ99]

A variedade de novas aplicações, principalmente as que lidam com fluxos de mídia contínuas (como aplicações multimídia que normalmente envolvem áudio e vídeo) estão requerendo das redes certas garantias. Essas garantias referem-se à alocação de largura de banda, sensibilidade no *delay*, controle da latência e variação do retardo (*jitter*) para determinada aplicação. Para essas garantias, é empregado o conceito conhecido como Qualidade de Serviço (QoS).

Qualidade de Serviço é a habilidade de um elemento de rede (aplicação, servidor ou comutador) em possuir algum nível de garantia de que seus requerimentos de tráfego e serviço sejam satisfeitos.

QoS aplica-se a diferentes tecnologias. O ATM, por exemplo, é uma tecnologia que, por definição, é orientada a QoS. O IP é uma tecnologia que, atualmente, está sendo redefinido para dotar sua arquitetura ao suporte a QoS (Serviços Integrados e Serviços

Diferenciados).

Desta forma, as redes atualmente devem ser capazes de assegurar QoS. A Tabela 4.1[MS99, CL95] mostra como diferentes aspectos de serviços precisam ser ajustados para cada tipo de tráfego.

Demanda de Tráfego na rede	Tráfego de Voz	Transferência de Arquivo	Vídeo Conferência	*Broadcast* de Vídeo
Banda Passante Média	Muito Baixa	Alta	Baixa	Muito Alta
Banda Passante Máxima	Baixa	Alta	Alta	Muito Alta
Delay	Muito Baixo	Alto	Baixo	Muito Alto
Variação no *Delay*	Muito Baixo	Alto	Muito Baixo	Baixo

Tabela 4.1: Ajuste de vários aspectos de serviço com cada tipo de tráfego.

4.2 Engenharia de Tráfego

A engenharia de tráfego preocupa-se com a otimização do desempenho das redes operacionais. Seu principal objetivo é fazer com que as redes sejam eficientes e confiáveis, enquanto, simultaneamente, minimizam o custo e maximizam os recursos ao transportar os dados[MS99].

Para facilitar o entendimento desse assunto, são expostos os parâmetros básicos de tráfego:

- *Carga ou Intensidade de Tráfego (A)*: diz respeito à utilização do sistema pelos usuários, isto é, o número de vezes e durante quanto tempo estes clientes utilizam o sistema em relação a um período de observação;
- *Grau de Serviço (B)*: refere-se à eficiência ou ao comportamento do sistema, estando intimamente ligado à satisfação do usuário e à economicidade do sistema, sendo basicamente o grau de aproveitamento que a administração do servidor deseja oferecer aos seus usuários;
- *Número de recursos (N)*: consiste do número de recursos (característica dos *links*, circuitos, etc) necessários ao sistema para poder atender às solicitações de serviço efetuadas pelo usuário (carga) de acordo com a disponibilidade preestabelecida (grau de serviço).

A engenharia de tráfego relaciona os três parâmetros acima entre si por meio da criação de modelos matemáticos. De uma maneira geral, tem-se:

$$\boxed{A = f(B, N) \iff N = f(A, B) \iff B = f(A, N)}$$

A função $N = f(A, B)$ é a mais importante para a engenharia de tráfego, uma vez que através desta relação é que se obtém o número de recursos (N) necessários para se atender uma carga de tráfego (A) de acordo com padrões de eficiência pré-estabelecidos (B).

4.2.1 Engenharia de Tráfego para o Projeto de Rede de Dados

O conceito de Engenharia de Tráfego é aplicável em uma rede de dados, no sentido de otimizar os recursos para a quantidade de serviços que irão fluir nessa rede.

Situações comuns são aquelas em que a rede tem todos os seus *links* em situação de sobrecarga, ou quando a carga não está bem distribuída na rede, apresentando alguns *links* mais congestionados que outros. Essa situação pode ser especulada num cenário no qual o caminho mais curto entre dois pontos na rede passa por um *link* congestionado, implicando em falha no atendimento a certas requisições dos serviços para esse fim. Dessa forma, a Engenharia de Tráfego contribui para a racionalização na distribuição de carga na rede, ajustando a carga nos diversos *links* de um domínio, por exemplo, de forma a retirar carga dos *links* mais congestionados e distribuí-la aos menos congestionados.

Além disso, as restrições de Qualidade de Serviço (QoS) impõem uma tarefa desafiante para os provedores de serviços de rede, que querem tanto maximizar a utilização dos recursos da rede quanto multiplexar o tráfego de comunicações de maneira eficiente.

4.2.2 Engenharia de Tráfego para o Projeto de VPNs

As VPNs permitem absorver estruturas já existentes de redes sem necessariamente criar novas redes físicas. Os provedores de serviços de redes vendem os recursos das suas estruturas físicas para comportar as diferentes requisições, ou seja, as VPNs dos clientes. Essas requisições respeitam certos padrões de eficiência previamente estabelecidos entre o cliente e o VSP.

Estes padrões de eficiência são comumente tratados como contrato de serviços, ou *Service Level Agreement* (SLA). SLAs são, tradicionalmente, contratos entre os clientes e os provedores de serviços de rede em que detalham o nível de concordância dos serviços em termos dos parâmetros mensurados. O SLA, geralmente, engloba o tipo de serviço, taxa de dados e a QoS.

O objetivo do projeto das VPNs é criar redes virtuais que respeitem as especificações requeridas pelos clientes, enquanto maximizam os recursos disponíveis da rede pertencente ao VSP. Para o VSP ser capaz de aumentar seu lucro ou ganho, deve esforçar-se para otimizar a utilização dos seus recursos (exemplo, banda passante dos *links* e dos multiplexadores).

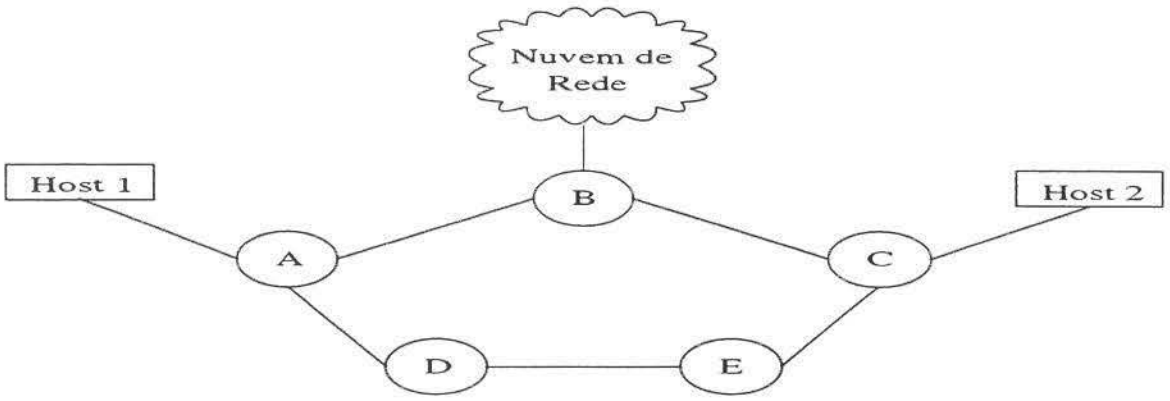

Figura 4.1: Exemplo da rede onde há diferença na escolha do caminho para a topologia da VPN do Host 1 para o Host 2, entre o cliente e o VSP.

Como exemplo, a Figura 4.1 mostra uma topologia de rede na qual, apesar de o cliente preferir que sua VPN passe do Host 1 para o Host 2 por meio de um único nó (B), o VSP irá passar a VPN por meio dos nós D e E para maximizar os recursos locais disponíveis, pois os recursos do nó B já estão sendo alocados para as VPNs originadas internamente a núvem de rede.

Assim sendo, é relevante fazer os seguintes questionamentos:

- Os requisitos dos clientes estão sendo satisfeitos ?
- Os recursos da rede são suficientes para criar a VPN ?
- A alocação dos recursos na rede do VSP está otimizada ?

Dessa maneira, não é uma tarefa simples para os provedores de serviços de rede atender eficientemente os requerimentos de diferentes classes de VPNs, maximizando os recursos das redes.

Capítulo 5

Alocação de Recursos para Redes Virtuais Privadas em Redes Baseadas em *Switchlets*

Redes programáveis oferecem perspectiva favorável para a implementação rápida de novos serviços de telecomunicação. *Switchlets* é uma técnica de redes programáveis na qual se pode alocar um subconjunto dos recursos de um comutador.

Este capítulo introduz um método para alocação de recursos em redes virtuais privadas (VPN) baseadas em *switchlets*. O método é baseado em *multicommodity flow* e sua implementação é factível em tempo real[JRF01].

Atualmente, a alocação de recursos para diferentes tipos de serviços é manual (exemplo, criação de PVCs), consumindo tempo e custo no processo. Há, também, a possibilidade de alocar esses recursos dinamicamente (exemplo, criação de SVCs), mas as alocações ainda demonstram ser um processo de dimensionamento custoso e que desperdiçam uma quantidade considerável de recursos.

Dessa forma, os *VPN Service Providers* (VSP) sofrem com a dificuldade em adaptar, dinamicamente, suas redes aos requisitos dos usuários, no sentido de automatizar e parametrizar o processo de criação, desenvolvimento e gerenciamento de distintas arquiteturas de redes.

Assim sendo, o presente trabalho preocupa-se com este problema e introduz um método, baseado em *multicommodity flow*, para que os VSPs possam alocar recursos dinamicamente, na criação de VPNs, em sua infra-estrutura física, visando maximizar o uso eficiente da banda bassante em suas redes.

Foi construído um esquema de programas para comprovar e validar o algoritmo desenvolvido. Com os resultados obtidos pelos exemplos numéricos gerados é possível evidenciar que o método proposto é factível de ser implementado em tempo real.

5.1 O Problema de Alocação de Recursos

Para estabelecer uma rede virtual, os recursos precisam ser alocados ao longo da rede. Este trabalho resolve a seguinte questão: *Como alocar recursos dinamicamente para VPNs de maneira eficiente e em tempo real?*

Em redes onde os elementos de comutação são blocos monolíticos de sinalização, a banda passante é o maior recurso a ser alocado. Entretanto, em redes que empregam a técnica de *switchlets*, os recursos de um comutador podem ser decompostos em vários comutadores lógicos. Além disso, a banda passante de entrada e saída desse comutador é compartilhada entre todas as *switchlets* de um comutador.

Desse modo, para representar a alocação de um conjunto de *switchlets* em um comutador de acordo com a sua banda passante em problemas de fluxos de rede, foi utilizada uma técnica de transformação de rede conhecida como *node splitting*, introduzida em 3.1. Usando esta transformação, foi possível modelar as *switchlets* para dividir os recursos de um comutador para criação dinâmica de VPNs.

O problema de alocação de recursos para VPNs em redes baseadas em *switchlets* consiste em alocar canais e *switchlets*. Deseja-se alocar recursos para VPNs de forma a minimizar o custo total do projeto. Utiliza-se, portanto, uma solução baseada em *multicommodity flow*, introduzida em 3.3.

A solução deste problema de *multicommodity flow* distribui o tráfego na rede de forma a minimizar o custo. O problema de *multicommodity flow* consiste em escolher caminhos para cada fluxo sem exceder as restrições de capacidade dos arcos. Define-se custo de um fluxo como sendo a soma dos custos dos arcos do caminho multiplicado pela quantidade de fluxo sobre esses arcos. O objetivo é minimizar o custo total dos fluxos.

Dessa forma, para resolver o problema de dimensionamento da rede, considera-se que não é permitido a divisão do fluxo por caminhos distintos. Apenas caminhos com capacidade suficiente para atender à demanda do fluxo devem ser considerados.

Assim, seja uma rede direcionada G, na qual o conjunto de nós é denotado por N, o conjunto de arcos por A e o conjunto de serviços por K. O conjunto $P(k)$ contém todos os caminhos do nó de origem até o nó de destino em G para o serviço $k \in K$. A unidade de custo atribuída a um caminho $p \in P(k)$ é c_p e a quantidade de fluxo requerida para o serviço k é q^k. Assim, tem-se que o total de custo atribuído ao serviço k para o caminho $p \in P(k)$ é $q^k c_p$.

Exemplo: Usando o mesmo exemplo do capítulo 3, seção 3.3, têm-se a rede da Figura 3.2 e as *commodities* 1 e 2. Assim, têm-se os conjuntos $P(k)$, $k = 1, 2$:

$$P(1) = \{p_{11}, p_{12}, p_{13}, p_{14}, p_{15}\} = \{[1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 7], \\ [1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7],$$

$$[1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7],$$
$$[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7],$$
$$[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7]\}.$$

$$P(2) = \{p_{21}, p_{22}, p_{23}, p_{24}, p_{25}\} = \{[1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 7],$$
$$[1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7],$$
$$[1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7],$$
$$[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7],$$
$$[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7]\}.$$

As unidades de custo correspondestes são: $c_{p_{11}} = 8$, $c_{p_{12}} = 5$, $c_{p_{13}} = 5$, $c_{p_{14}} = 6$, $c_{p_{15}} = 6$, $c_{p_{21}} = 8$, $c_{p_{22}} = 5$, $c_{p_{23}} = 5$, $c_{p_{24}} = 6$, $c_{p_{25}} = 6$.

A variável de decisão do problema y_p^k possui o valor 1, se o serviço k for atribuído ao caminho p; caso contrário y_p^k possui valor zero. As variáveis d_a, $a \in A$, são as capacidades dos arcos, e as variáveis ∂_a^p, $a \in A$ e $p \in P(k)$, $k \in K$, possuem valor 1, se o arco a estiver contido no caminho p. Por outro lado, ∂_a^p possuem valor zero.

Assim sendo, o problema de alocação de recursos pode ser formulado da seguinte forma:

$$Minimize \sum_{k \in K} \sum_{p \in P(k)} c_p q^k y_p^k \tag{5.1}$$

$$subject\ to \sum_{k \in K} \sum_{p \in P(k)} q^k y_p^k \partial_a^p \leq d_a, \forall a \in A, \tag{5.2}$$

$$\sum_{p \in P(k)} y_p^k = 1, \forall k \in K, \tag{5.3}$$

$$y_p^k \in \{0,1\}, \forall p \in P(k), \forall k \in K. \tag{5.4}$$

A função objetiva (5.1) é um somatório da unidade de custo c_p do caminho $p \in P(k)$ multiplicado pelo fluxo q^k requerido pelo serviço k. A variável de decisão y_p^k é igual a 1, se, e somente se, o serviço k for designado para o caminho p. O objetivo é minimizar esta soma com as seguintes restrições:

- A restrição (5.2) limita a soma de todos os fluxos que utilizam o arco a à capacidade d_a;
- A restrição (5.3) limita a designação do serviço k a somente um dos caminhos de $P(k)$;

- A restrição (5.4) garante que nenhum fluxo é divido entre dois ou mais caminhos ou enviado parcialmente por um caminho. Esta restrição juntamente com a restrição (5.3) garantem que um fluxo é enviado integralmente por um único caminho.

Esse modelo é uma formulação por caminho do problema MCF inteiro. As variáveis de decisão y_p^k atribuem caminhos para os serviços e possuem valores binários (0 ou 1). Esse problema é mais complexo que o problema de MCF linear, no qual as variáveis de decisão podem ser valores reais. Para resolver o problema MCF inteiro de maneira mais eficiente, pode-se utilizar o algoritmo de *branch-and-bound.*

Exemplo: A formulação do problema de *multicommodity flow* inteiro para as duas *commodities* da rede fica:

$$\begin{aligned} \textit{Minimize} \quad & c_{p_{11}}q^1y^1_{p_{11}} + c_{p_{12}}q^1y^1_{p_{12}} + c_{p_{13}}q^1y^1_{p_{13}} + c_{p_{14}}q^1y^1_{p_{14}} + c_{p_{15}}q^1y^1_{p_{15}} + c_{p_{21}}q^2y^2_{p_{21}} + \\ & c_{p_{22}}q^2y^2_{p_{22}} + c_{p_{23}}q^2y^2_{p_{23}} + c_{p_{24}}q^2y^2_{p_{24}} + c_{p_{25}}q^2y^2_{p_{25}} \\ & = 240y^1_{p_{11}} + 150y^1_{p_{12}} + 150y^1_{p_{13}} + 180y^1_{p_{14}} + 180y^1_{p_{15}} + 120y^2_{p_{21}} + 75y^2_{p_{22}} + \\ & 75y^2_{p_{23}} + 90y^2_{p_{24}} + 90y^2_{p_{25}} \end{aligned}$$

Subject to

$$\begin{aligned}
30y^1_{p_{11}} + 30y^1_{p_{12}} + 30y^1_{p_{13}} \qquad\qquad & +15y^2_{p_{21}} + 15y^2_{p_{22}} + 15y^2_{p_{23}} \qquad\qquad && \leq 30, \\
+30y^1_{p_{14}} + 30y^1_{p_{15}} \qquad\qquad & +15y^2_{p_{24}} + 15y^2_{p_{25}} && \leq 20, \\
30y^1_{p_{11}} \qquad\qquad & +15y^2_{p_{21}} && \leq 30, \\
+30y^1_{p_{12}} + 30y^1_{p_{13}} \qquad\qquad & +15y^2_{p_{22}} + 15y^2_{p_{23}} && \leq 20, \\
+30y^1_{p_{14}} + 30y^1_{p_{15}} \qquad\qquad & +15y^2_{p_{24}} + 15y^2_{p_{25}} && \leq 20, \\
30y^1_{p_{11}} + 30y^1_{p_{12}} \qquad +30y^1_{p_{14}} \qquad & +15y^2_{p_{21}} + 15y^2_{p_{22}} \qquad +15y^2_{p_{24}} \qquad && \leq 30, \\
+30y^1_{p_{13}} \qquad +30y^1_{p_{15}} \qquad & +15y^2_{p_{23}} \qquad +15y^2_{p_{25}} && \leq 25, \\
+30y^1_{p_{12}} \qquad +30y^1_{p_{14}} \qquad & +15y^2_{p_{22}} \qquad +15y^2_{p_{24}} \qquad && \leq 20, \\
+30y^1_{p_{13}} \qquad +30y^1_{p_{15}} \qquad & +15y^2_{p_{23}} \qquad +15y^2_{p_{25}} && \leq 20, \\
y^1_{p_{11}} + y^1_{p_{12}} + y^1_{p_{13}} + y^1_{p_{14}} + y^1_{p_{15}} \qquad\qquad & && = 1, \\
y^2_{p_{21}} + y^2_{p_{22}} + y^2_{p_{23}} + y^2_{p_{24}} + y^2_{p_{25}} & && = 1,
\end{aligned}$$

$$y_p^k \in \{0, 1\}\,,\ \forall p \in P(k),\ \forall k \in K.$$

Nas restrições do exemplo acima, as primeiras nove linhas correspondem aos arcos da rede. O lado direito das inequações corresponde às capacidades dos arcos. As colunas do lado esquerdo representam os possíveis caminhos na rede. A instância p_{11} usa os arcos (1,2), (2,4) e (4,7), assim, a variável de decisão $y^1_{p_{11}}$ aparece nas restrições das linhas 1, 3 e 6. Os coeficientes das variáveis y_p^k's são a demanda de fluxo da *commodity* k. As linhas 10 e 11 correspondem às duas *commodities*, ou seja, na linha 10 aparecem as variáveis

que satisfazem a primeira *commodity* e na linha 11 as variáveis que satisfazem a segunda *commodity*. Estas duas linhas determinam que os fluxos irão trafegar por somente um caminho, isso porque na última linha foi especificado que as variáveis de decisão assumem valores inteiros ou binários ($y_p^k \in \{0, 1\}$), tornando o problema inteiro.

Desta forma, a única diferença deste problema inteiro para um problema linear é a última restrição. Se as variáveis de decisão assumirem valores reais ($y_p^k \in [0, 1]$), o problema passa a ser linear.

De maneira prática, para formular o problema de *multicommodity flow* inteiro é necessário encontrar os caminhos possíveis na rede para cada *commodity* ou VPN. Primeiramente, foi utilizada uma modificação do algoritmo de busca em profundidade. Esse algoritmo mostrou-se incapaz de tratar problemas maiores, demonstrando, de forma geral, haver um problema de escalabilidade na formulação desta metodologia.

Partindo deste princípio foi aplicada uma generalização do algoritmo de Dijkstra, introduzido em 3.5. Com este algoritmo, o problema de escalabilidade na formulação desta metodologia foi melhorado, ou seja, foi possível tratar problemas maiores, os quais, usando a modificação de busca em profundidade não foi possível resolver. Assim, a metodologia desenvolvida demonstrou ser viável na resolução do problema de alocar recursos de maneira eficiente e em tempo real.

5.2 Descrição do Ambiente de Software

Foi construído um esquema de programas para comprovar e validar a eficiência do modelo proposto. Essa seção apresenta a metodologia utilizada para resolver o problema de *multicommodity flow* inteiro usando *switchlets* introduzido nas seções anteriores.

Os programas foram desenvolvidos utilizando o C ANSI como linguagem de programação e foram executados em uma máquina Pentium III, 450 MHz, 512 KB *Cache*, memória primária de 384 MB, memória para *swap* de 72 MB. Foi utilizado o sistema operacional Linux, kernel 2.2.12-20.

Assim, foram implementados quatro módulos de programas:

Módulo gera rede e VPNs(*commodities*)

Este módulo é constituído de dois programas: um para gerar topologias de rede e outro para gerar as VPNs. Tanto as topologias, quanto as VPNs são obtidas usando um gerador de números aleatórios.

Assim, são gerados dois arquivos: um para a rede e outro para as VPNs/*commodities*. No arquivo da rede, cada linha contém: nó de origem; nó de destino; custo e fluxo (banda passante do *link*). Os custos são gerados em função da banda passante no *link*,

ou seja, quanto maior o fluxo, maior o custo. No arquivo das VPNs cada linha contém: nó de origem; nó de destino e fluxo (banda passante exigida pela VPN). Detalhes de implementação podem ser obtidos no apêndice A.

Módulo *Node Splitting*

Este programa recebe o arquivo com a rede original e faz a transformação (*node splitting*), gerando um arquivo com a rede transformada. Essa rede é maior que a rede original, dado o aumento do número de *links*, seguindo a fórmula abaixo:

$$\boxed{\textit{Links} \text{ rede transformada} = (\textit{Links} \text{ rede original} * 2 + \text{Nós rede original})}$$

Tanto o módulo "gera rede e VPNs", quanto o módulo "*node splitting*" criam o problema a ser resolvido. Em uma rede operacional, os dados de saída desses dois módulos correspondem à topologia e aos recursos da rede, os quais deseja-se maximizar a utilização. Em outras palavras, esses módulos não fazem parte de um agente responsável pelo gerenciamento da rede.

Módulo Achar Caminhos

Este módulo recebe um arquivo com a rede transformada e um arquivo com as *commodities*, e gera um arquivo com a formulação do problema (função objetiva, matriz e limites). A função desse módulo é achar os caminhos possíveis para cada *commodity* na rede. Esses caminhos serão as variáveis da função objetiva.

Para achar os caminhos possíveis para cada *commodity*, foram utilizadas duas modificações de algoritmos clássicos encontrados na literatura:

- Busca em Profundidade: uma modificação do algoritmo de busca em profundidade foi utilizada para encontrar os caminhos possíveis, aleatoriamente, para cada VPN (*commodity*). Esta modificação é apresentada de maneira detalhada pelo pseudocódigo no apêndice A;

- Dijkstra para K caminhos: uma modificação do algoritmo de Dijkstra foi utilizada. Com este algoritmo foi possível obter K caminhos mínimos para cada *commodity*. Foi utilizada a estrutura de dados *heap* para implementar este algoritmo. Os detalhes de implementação deste algoritmo são abordados na seção 3.5, incluindo-se na abordagem seu pseudocódigo.

Módulo *Multicommodity Flow*

Esse módulo resolve o problema de *multicommodity flow* inteiro. Ele recebe um arquivo com a formulação do problema de *multicommodity flow* inteiro e chama as funções do módulo MIP do CPLEX para resolver o problema e fornecer os resultados.

O CPLEX é um software da ILOG utilizado para resolver problemas de otimização. Foi utilizado o CPLEX *Mixed Integer optimizer*, versão 6.5, para resolver problemas de otimização inteira [ILO99]. No presente caso, todas as variáveis foram setadas como binárias. O CPLEX *Mixed Integer optimizer*, ou o otimizador MIP, explora o algoritmo de *branch-and-bound*.

5.3 Exemplo Numérico

Para ilustrar a solução do problema de alocação de recursos para VPNs em redes baseadas em *switchlets*, utilizou-se a rede da Figura 5.1[MR99]. Os rótulos nos arcos são representados por um par ordenado: custo e banda passante.

Dessa forma, intenciona-se alocar recursos na rede (Figura 5.1) para as VPNs cujos fluxos estão especificados na matriz de tráfego da tabela 5.1. A solução pode ser vista na Tabela 5.2.

Cada caminho gerado corresponde a uma variável na função objetiva. A complexidade do algoritmo aumenta, particularmente, em função do número de *links*.

Nós	1	2	3	4	5	6	7	8
1	-	6	8	3	4	3	5	8
2	8	-	7	8	5	9	6	4
3	1	7	-	1	8	7	8	7
4	3	8	8	-	11	5	3	3
5	9	8	1	3	-	1	4	9
6	3	6	2	5	7	-	10	16
7	5	13	16	3	21	10	-	5
8	8	21	26	3	31	16	5	-

Tabela 5.1: Matriz de Tráfego das VPNs. Os valores internos a matriz representam a banda passante, em Mbps, requerida para cada VPN.

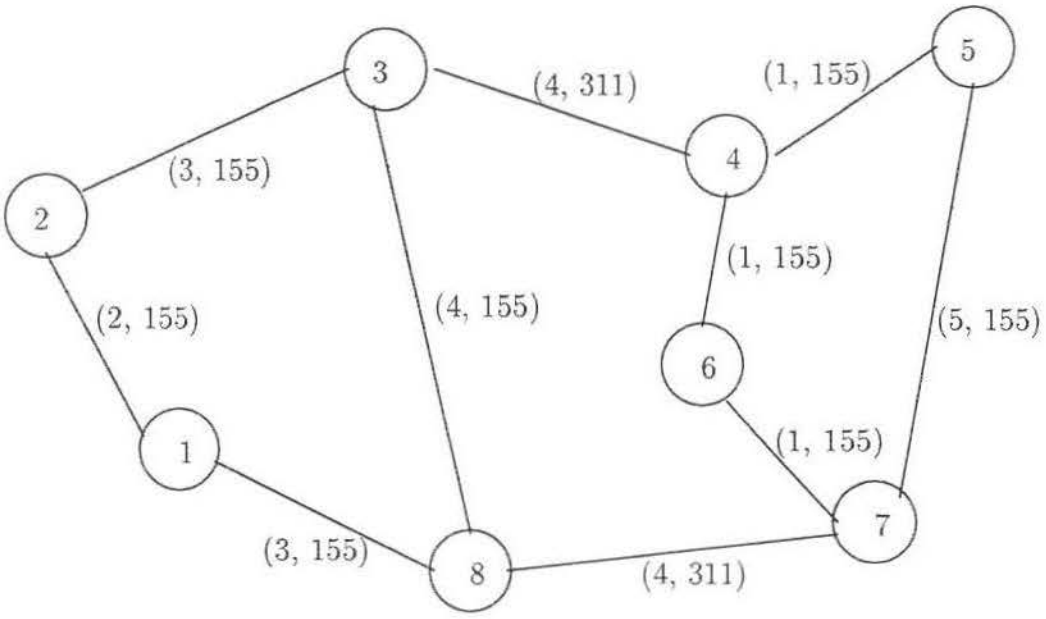

Figura 5.1: Topologia da Rede.

Variáveis Analisadas	Valores Obtidos
Custo Total	25866
Tempo de Execução	00:00:00.09 *
Número de Switches	8
Número de *Links*	28
Número de *Commodities*	56
Caminhos Gerados	162

*hora:minuto:segundo

Tabela 5.2: Resultado da Alocação das VPNs na rede da Figura 5.1. Valores gerados utilizando *switchlets* (*node splitting*).

5.4 Eficiência do método proposto

Para avaliar a viabilidade da adoção do método proposto no presente trabalho em uma rede operacional, foram gerados 18 redes com diferentes números de nós e *links*, variando, assim, o grau de conectividade da rede.(Tabela 5.3)

Problema	Nós	Arcos	*Commodities*
1	8	25	21
2	8	25	33
3	10	15	28
4	10	15	41
5	10	45	28
6	10	45	41
7	12	50	65
8	20	30	42
9	20	30	90
10	30	40	43
11	30	40	97
12	30	43	43
13	30	43	97
14	30	45	97
15	30	50	57
16	30	100	43
17	30	300	43
18	30	300	97

Tabela 5.3: Problemas gerados para teste.

Desta forma, na Tabela 5.3 pode ser observado que foram gerados exemplos tanto para grafos densos como para grafos esparsos, variando, também, o número de VPNs (serviços). Assim, foi possível verificar quais os fatores (número de nós, números de *links* ou número de *commodities*) que mais influenciam na complexidade do problema (tempo de execução).

Resultados usando uma modificação do algoritmo de busca em profundidade

Na primeira implementação do problema, o módulo achar caminhos usava o algoritmo de busca em profundidade para encontrar todos os caminhos possíveis para cada serviço na rede. Entretanto, esta abordagem inicial mostrou-se inadequada para ser aplicada, em

razão do grande número de caminhos selecionados para abordagens simples, que impactaram no tempo de execução dos problemas, inviabilizando o modelo proposto.

Dessa forma, o algoritmo de busca em profundidade foi modificado para permitir testar todos os problemas em questão. A modificação do algoritmo de busca em profundidade é descrita no apêndice A.

O módulo achar caminhos passou a utilizar uma modificação do algoritmo de busca em profundidade para achar os caminhos possíveis para cada serviço na rede. Com esta nova abordagem foram realizados três tipos de testes com os mesmos dados, variando-se a quantidade de caminhos escolhidos por serviço/*commodity*.

No primeiro teste, foram escolhidos os caminhos possíveis para cada *commodity*. Quanto maior a quantidade de caminhos, maior o poder de escolha do CPLEX, melhorando, assim, o valor do custo total para acomodar todas as *commodities*, mas, por outro lado, piora o tempo de execução do algoritmo. Os resultados são mostrados na Tabela 5.4.

Problema	Caminhos	Função Objetiva	Tempo Execução *
1	547	7191	00:00:00.18
2	872	9531	00:00:00.20
3	186	12615	00:00:00.12
4	265	15309	00:00:00.08
5	11228	3810	00:00:11.72
6	16441	4131	00:00:24.70
7	96636	14697	00:26:45.90
8	3506	18549	00:00:03.09
9	6509	33561	00:00:09.52
10	11468	22371	00:00:52.11
11	27015	64449	00:05:21.39
12	13013	15981	00:01:04.75
13	29144	Infactível	00:05:28.90
14	97412	Infactível	01:18:44.89

*hora:minuto:segundo

Tabela 5.4: Resultados explorando todos os caminhos possíveis.

No segundo teste, foram escolhidos 100 caminhos possíveis para cada *commodity* (Tabela 5.5). No terceiro teste, foram escolhidos 50 caminhos possíveis para cada *commodity* (Tabela 5.6). Foi estipulado que tempos de execução maiores que 5 minutos para encontrar os caminhos da solução não são apropriados para o dimensionamento dinâmico de rede. Os resultados indicam que o tempo de execução é desprezível para redes com até 20 nós e 30 *links*. Para as redes com 30 nós e 40 *links*, o tempo de execução aumenta, mas

Problema	Caminhos	Função Objetiva	Tempo Execução
1	547	7191	00:00:00.14
2	872	9531	00:00:00.20
3	186	12615	00:00:00.07
4	265	15309	00:00:00.08
5	2800	4518	00:00:01.09
6	4100	5982	00:00:01.99
7	6500	27585	00:00:06.65
8	3214	18774	00:00:02.60
9	6218	33825	00:00:08.39
10	4003	30252	00:00:06.17
11	9004	88089	00:00:36.84
12	3921	20394	00:00:05.19
13	9003	Infactível	00:00:26.42
14	9700	Infactível	00:00:33.65
15	9601	Infactível	00:00:34.87
16	4300	42294	00:00:07.37
17	4300	Infactível	00:00:08.40
18	9700	Infactível	00:00:40.33

Tabela 5.5: Resultados limitando em 100 caminhos por *commodity*.

continua sendo aceitável. Observa-se que o tempo de execução aumenta exponencialmente em função do número de arcos, como esperado.

Observa-se na Tabela 5.4 que, à medida que o número de enlaces da rede aumenta, o número de caminhos gerados aumenta exponencialmente. Isso implica num aumento na mesma proporção exponencial no tempo de execução do CPLEX, o que é um sério problema de escalabilidade. A operação *node splitting* aumenta ainda mais o número de enlaces da rede, o que implica em um aumento na complexidade do problema. Para contornar o fato do problema de *multicommodity flow* inteiro ser NP-difícil foi proposto um número limite de caminhos encontrados por serviço[MR99]. A Tabela 5.6 mostra os resultados obtidos ao limitar-se o número de caminhos por serviço para apenas 50. Comparando esses resultados com a Tabela 5.4, percebe-se que o tempo de execução foi de apenas alguns segundos para todos os 18 problemas, mesmo para aqueles que demoraram vários minutos na primeira abordagem. Entretanto, como era de se esperar, o valor da função objetiva não foi o valor ótimo em todos os casos. Isso acontece porque muitos caminhos possíveis são descartados, os quais poderiam ter um custo menor. A diferença entre o valor da função objetiva e o valor ótimo encontrado na primeira abordagem aumenta

Problema	Caminhos	Função Objetiva	Tempo Execução
1	547	7191	00:00:00.15
2	872	9531	00:00:00.20
3	186	12615	00:00:00.16
4	265	15309	00:00:00.10
5	1400	5679	00:00:00.42
6	2050	7521	00:00:00.67
7	3250	31668	00:00:02.07
8	2010	20310	00:00:01.19
9	4217	38364	00:00:04.85
10	2003	32775	00:00:01.76
11	4507	102999	00:00:08.60
12	1971	25044	00:00:01.67
13	4525	Infactível	00:00:06.47
14	4850	Infactível	00:00:08.25
15	4801	Infactível	00:00:07.83
16	2150	43584	00:00:02.18
17	2150	Infactível	00:00:02.45
18	4850	Infactível	00:00:08.61

Tabela 5.6: Resultados limitando em 50 caminhos por *commodity*.

proporcionalmente em relação ao número de enlaces da rede.

A Tabela 5.5 mostra os resultados obtidos ao se limitar o número de caminhos por serviço para 100. O tempo de execução foi de apenas alguns segundos para todos os 18 problemas. Como era de se esperar, o valor da função objetiva ficou mais próximo do valor ótimo do que o valor na abordagem com 50 caminhos por serviço, e, em muitos casos, essa diferença foi irrelevante ou nula. Apenas dobrando o número de caminhos por serviço, melhora-se o valor da função objetiva sem aumentar significativamente o tempo de execução.

Ao se limitar o número de caminhos por serviço obtem-se um maior grau de escalabilidade sem grandes perdas na precisão da solução ótima. O problema, que tinha uma complexidade exponencial com relação ao número de enlaces, passa a ter complexidade linear com relação ao número de serviços. Entretanto, por ser uma heurística, não se pode garantir que se encontrará uma solução factível, mesmo quando ela exista. Além disso, não se tem como calcular o quanto o valor ótimo é melhor que o valor encontrado na função objetiva. No entanto, a possibilidade de encontrar soluções subótimas em um tempo de execução hábil pode ser mais importante do que a obtenção da solução ótima.

Como exemplo, têm-se os problemas 15, 16, 17 e 18 que demorariam horas ou dias para serem resolvidos exatamente na primeira abordagem, mas, que, em apenas poucos segundos, pode-se obter uma aproximação do valor ótimo. Isso é importante em problemas de dimensionamento de rede dinâmico, nos quais o tempo de execução da otimização é um fator crucial.

Resultados usando uma generalização do algoritmo de Dijkstra

Constata-se, na abordagem utilizando o algoritmo de busca em profundidade, que limitando o número de caminhos por serviço, o problema da escalabilidade, para os problemas 15, 16, 17 e 18 está resolvido e o tempo de execução é satisfatório. Entretanto, o valor da função objetiva não é o valor ótimo. Isso acontece devido à deficiência do algoritmo de busca em profundidade em escolher caminhos aleatoriamente. Assim sendo, iniciou-se uma pesquisa por um algoritmo que escolhesse caminhos não aleatoriamente, que tivesse critérios na escolha dos caminhos.

Problema	Caminhos	Função Objetiva	Tempo Execução
1	2100	7191	00:00:02.10
2	3300	9531	00:00:03.67
3	453	12615	00:00:01.79
4	725	15309	00:00:02.63
5	2800	3810	00:00:02.92
6	4100	4131	00:00:04.66
7	6500	14697	00:00:08.57
8	4068	18549	00:00:07.19
9	8591	33561	00:00:26.28
10	3998	22371	00:00:07.45
11	8999	64449	00:00:29.52
12	3880	15981	00:00:06.76
13	8849	Infactível	00:00:25.13
14	9686	Infactível	00:00:23.87
15	9601	48450	00:00:21.49
16	4300	9600	00:00:05.56
17	4300	8397	00:00:07.39
18	9700	21732	00:00:17.34

Tabela 5.7: Resultados considerando 100 melhores caminhos entre os pontos finais das VPN's.

Este novo algoritmo deve obedecer a uma certa métrica na escolha dos caminhos,

como, por exemplo, o custo. Assim, utilizou-se o algoritmo clássico de Dijkstra, pois este usa uma métrica na escolha dos caminhos (custo), escolhendo, assim, caminhos mínimos. Era inviável, porém, utilizá-lo, pois o algoritmo clássico de Dijkstra encontra caminhos mínimos de um nó da rede para todos os outros, e no problema em questão deseja-se encontrar somente "k" caminhos mínimos entre dois pontos da rede. Assim sendo, foi encontrada, em [SS99], uma generalização do algoritmo de Dijkstra, que atendia a essa necessidade. Esse algoritmo é detalhado em 3.5.

Desta forma, uma segunda implementação do módulo achar caminhos mostrou-se necessária. Na nova implementação desse módulo foi empregada uma generalização do algoritmo de Dijkstra. Com essa nova abordagem, foram realizados três tipos de testes com os mesmos dados gerados da Tabela 5.3.

No primeiro teste, foram escolhidos 100 melhores caminhos possíveis para cada *commodity* (Tabela 5.7). No segundo teste, foram escolhidos 30 melhores caminhos possíveis (Tabela 5.8). No terceiro teste, foram escolhidos 10 melhores caminhos possíveis (Tabela 5.9.

Problema	Caminhos	Função Objetiva	Tempo Execução
1	630	7191	00:00:00.21
2	990	9531	00:00:00.33
3	453	12615	00:00:00.21
4	725	15309	00:00:00.31
5	840	3810	00:00:00.30
6	1230	4131	00:00:00.43
7	1950	14697	00:00:00.74
8	1231	18549	00:00:00.61
9	2637	33561	00:00:01.84
10	1202	22371	00:00:00.69
11	2707	64449	00:00:02.39
12	1186	15981	00:00:00.62
13	2692	Infactível	00:00:01.72
14	2900	Infactível	00:00:01.76
15	2881	48450	00:00:01.71
16	1290	9600	00:00:00.68
17	1290	8397	00:00:01.23
18	2910	21732	00:00:02.56

Tabela 5.8: Resultados considerando 30 melhores caminhos entre os pontos finais das VPN's.

A Tabela 5.7 mostra os resultados obtidos ao se limitar o número de caminhos por

serviço para 100 ($K = 100$). O tempo de execução é de apenas alguns segundos para todos os 18 problemas e o valor da função objetiva obtida é melhor que os valores das tabelas 5.5 e 5.6.

A Tabela 5.8 mostra os resultados obtidos ao se limitar o número de caminhos por serviço para 30 ($K = 30$). O tempo de execução é de apenas alguns segundos para todos os 18 problemas e o valor da função objetiva obtida é igual aos valores da tabela 5.7.

A Tabela 5.9 mostra os resultados obtidos ao se limitar o número de caminhos por serviço para 10 ($K = 10$). O tempo de execução é de apenas alguns centésimos de segundos para todos os 18 problemas e o valor da função objetiva obtida é igual aos valores das tabelas 5.7 e 5.8.

Problema	Caminhos	Função Objetiva	Tempo Execução
1	210	7191	00:00:00.11
2	330	9531	00:00:00.15
3	269	12615	00:00:00.14
4	408	15309	00:00:00.19
5	280	3810	00:00:00.16
6	410	4131	00:00:00.19
7	650	14697	00:00:00.30
8	408	18549	00:00:00.22
9	876	33561	00:00:00.49
10	403	22371	00:00:00.24
11	907	64449	00:00:00.66
12	410	15981	00:00:00.24
13	918	Infactível	00:00:00.51
14	968	Infactível	00:00:00.54
15	961	48450	00:00:00.52
16	430	9600	00:00:00.26
17	430	8397	00:00:00.44
18	970	21732	00:00:00.86

Tabela 5.9: Resultados considerando 10 melhores caminhos entre os pontos finais das VPN's.

Os resultados dos testes mostraram que, com a nova abordagem, o problema de escalabilidade está, de certa forma, resolvido e o problema que tinha uma complexidade exponencial com relação ao número de enlaces passa a ter uma complexidade linear com relação ao número de serviços. Entretanto, por ser uma heurística, não se pode garantir que esta encontra uma solução factível, mesmo quando ela exista. Além disso, não se tem como calcular o erro entre o valor calculado e o valor ótimo. A possibilidade, no

entanto, de encontrar soluções subótimas em um tempo de execução hábil pode ser mais importante do que a obtenção da solução ótima.

Com a abordagem utilizando a generalização do algoritmo de Dijkstra, tanto o tempo de execução quanto o valor da função objetiva ficaram satisfatórios. Isso demonstra que a metodologia desenvolvida é viável de ser empregada em problemas de dimensionamento de rede dinâmico.

Capítulo 6

Alocação de Recursos em Redes *MultiClasses*

A metodologia apresentada no capítulo anterior é estendida no presente capítulo para incluir requisitos de QoS na alocação de recursos para VPNs. Nesta nova abordagem é tratado e resolvido o problema da infactibilidade na alocação dos recursos para as VPNs na rede. Assim, para evitar a infactibilidade, algumas VPNs são rejeitadas. A rejeição é feita sempre visando a maximização do lucro ou ganho do VSP.

Para estabelecer uma rede virtual, os recursos precisam ser alocados ao longo da rede. Este trabalho torna-se mais complexo quando há a necessidade de diferenciar a alocação dos recursos para cada VPN devido as diferentes classes de serviços. Assim sendo, neste capítulo resolve-se o seguinte problema: *Como Alocar Recursos para VPNs com diferentes classes de serviços de maneira eficiente e em tempo real?*

O problema de alocação de recursos para VPNs em redes baseadas em *switchlets* consiste em alocar canais e *switchlets*. Deseja-se implementar VPNs de forma a maximizar o lucro ou ganho do *VPN Service Provider*.

Dessa forma, para calcular o lucro ou ganho nessa nova formulação é necessário acrescentar uma variável na função objetiva, correspondente ao preço da VPN (w_k). As outras variáveis (c_p, q^k e y_p^k) são iguais às da formulação do capítulo anterior.

A unidade de custo atribuída a um caminho é c_p, e a quantidade de fluxo requerida para o serviço k é q^k. Assim, tem-se que o total de custo atribuído ao serviço k para o caminho $p \in P(k)$ é $q^k c_p$. A unidade de preço atribuída a uma VPN (serviço) é w_k para todo serviço $k \in K$.

Exemplo: Usando o mesmo exemplo do capítulo 3, seção 3.3, tem-se: a rede da Figura 3.2 e as *commodities* 1 e 2. Com esta nova formulação, são acrescentadas duas métricas a essas *commodities* (classe de serviço e preço), como descrito abaixo:

- *Commodity 1:* Nó origem: 1; nó destino: 7; fluxo: 30; Classe: 1 e Preço: 3000;

- *Commodity 2:* Nó origem: 1; nó destino: 7; fluxo: 15; Classe: 4 e Preço: 1500;

A rede da Figura 3.2 têm um diâmetro de 3 *hops*. Por este motivo, para a *commodity* 1, que possui classe de serviço 1, são escolhidos caminhos que tenham um número menor ou igual ao diâmetro da rede. Assim, tem-se os conjuntos $P(k)$, $k = 1, 2$:

$$P(1) = \{p_{11}\} = \{[1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 7].$$

$$\begin{aligned} P(2) = \{p_{21}, p_{22}, p_{23}, p_{24}, p_{25}\} = \{&[1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 7], \\ &[1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7], \\ &[1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7], \\ &[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 4 \longrightarrow 7], \\ &[1 \longrightarrow 3 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7]\}. \end{aligned}$$

As unidades de custo correspondestes são: $c_{p_{11}} = 8$, $c_{p_{21}} = 8$, $c_{p_{22}} = 5$, $c_{p_{23}} = 5$, $c_{p_{24}} = 6$, $c_{p_{25}} = 6$.

A variável de decisão do problema y_p^k possui o valor 1, se o serviço k for atribuído ao caminho p; caso contrário y_p^k possui valor zero. As variáveis d_a, $a \in A$, são as capacidades dos arcos e as variáveis ∂_a^p, $a \in A$ e $p \in P(k)$, $k \in K$, possuem valor 1, se o arco a estiver contido no caminho p. Por outro lado, ∂_a^p possuem valor zero.

Assim, o problema de alocação de recursos para serviços *multiclasses* pode ser formulado da seguinte forma:

$$Maximize \sum_{k \in K} \sum_{p \in P(k)} (w_k - c_p q^k) y_p^k \tag{6.1}$$

$$subject\ to \sum_{k \in K} \sum_{p \in P(k)} q^k y_p^k \partial_a^p \leq d_a, \forall a \in A, \tag{6.2}$$

$$\sum_{p \in P(k)} y_p^k \leq 1, \forall k \in K, \tag{6.3}$$

$$y_p^k \in \{0, 1\}, \forall p \in P(k), \forall k \in K. \tag{6.4}$$

A função objetiva (6.1) é um somatório do conjunto: unidade de preço w_k do serviço $k \in K$ subtraído da unidade de custo c_p do caminho $p \in P(k)$ multiplicado pelo fluxo q^k requerido pelo serviço k. A variável de decisão y_p^k é igual a 1, se, e somente se, o serviço k for designado para o caminho p. O objetivo é maximizar esta soma com as seguintes restrições:

- A restrição (6.2) limita a soma de todos os fluxos que utilizam o arco a à capacidade d_a;
- A restrição (6.3) limita a designação do serviço k a somente um dos caminhos de $P(k)$. Esta restrição permite também que nenhum caminho $P(k)$ seja atribuído ao serviço k;
- A restrição (6.4) garante que nenhum fluxo é divido entre dois ou mais caminhos ou enviado parcialmente por um caminho. Esta restrição junto com a restrição (6.3) garantem que um fluxo é enviado integralmente por um único caminho.

O presente modelo é uma formulação por caminho do problema MCF inteiro, como no modelo apresentado no capítulo 5. As variáveis de decisão y_p^k atribuem caminhos para os serviços e possuem valores binários (0 ou 1).

Exemplo: A formulação do problema de *multicommodity flow* inteiro para as duas *commodities* da rede fica:

$$\begin{aligned} \textit{Maximize} \quad & (w_1 - c_{p_{11}} q^1) y^1_{p_{11}} + (w_2 - c_{p_{21}} q^2) y^2_{p_{21}} + (w_2 - c_{p_{22}} q^2) y^2_{p_{22}} + (w_2 - c_{p_{23}} q^2) y^2_{p_{23}} \\ & + (w_2 - c_{p_{24}} q^2) y^2_{p_{24}} + (w_2 - c_{p_{25}} q^2) y^2_{p_{25}} \\ & = 2760 y^1_{p_{11}} + 1380 y^2_{p_{21}} + 1425 y^2_{p_{22}} + 1425 y^2_{p_{23}} + 1410 y^2_{p_{24}} + 1410 y^2_{p_{25}} \end{aligned}$$

Subject to

$$\begin{aligned}
30y^1_{p_{11}} + 15y^2_{p_{21}} + 15y^2_{p_{22}} + 15y^2_{p_{23}} &\leq 30, \\
+15y^2_{p_{24}} + 15y^2_{p_{25}} &\leq 20, \\
30y^1_{p_{11}} + 15y^2_{p_{21}} &\leq 30, \\
+15y^2_{p_{22}} + 15y^2_{p_{23}} &\leq 20, \\
+15y^2_{p_{24}} + 15y^2_{p_{25}} &\leq 20, \\
30y^1_{p_{11}} + 15y^2_{p_{21}} + 15y^2_{p_{22}} + 15y^2_{p_{24}} &\leq 30, \\
+15y^2_{p_{23}} + 15y^2_{p_{25}} &\leq 25, \\
+15y^2_{p_{22}} + 15y^2_{p_{24}} &\leq 20, \\
+15y^2_{p_{23}} + 15y^2_{p_{25}} &\leq 20, \\
y^1_{p_{11}} &\leq 1, \\
y^2_{p_{21}} + y^2_{p_{22}} + y^2_{p_{23}} + y^2_{p_{24}} + y^2_{p_{25}} &\leq 1,
\end{aligned}$$

$$y_p^k \in \{0, 1\}, \ \forall p \in P(k), \ \forall k \in K.$$

Nas restrições do exemplo acima, as primeiras nove linhas correspondem aos arcos da rede. O lado direito das inequações correspondem às capacidades dos arcos. As colunas

do lado esquerdo representam os possíveis caminhos na rede. A instância p_{11} usa os arcos (1,2), (2,4) e (4,7), assim, a variável de decisão $y^1_{p_{11}}$ aparece nas restrições 1, 3 e 6. Os coeficientes das variáveis y^k_p's são a demanda de fluxo da *commodity* k. As linhas 10 e 11 correspondem às duas *commodities*, ou seja, na linha 10 aparecem as variáveis que satisfazem a primeira *commodity* e, na linha 11, as variáveis que satisfazem a segunda *commodity*. Estas duas linhas determinam que os fluxos irão trafegar por somente um caminho ou que não irão trafegar por nenhum dos caminhos (em razão da inequação ser ≤ 1), isso porque na última linha foi especificado que as variáveis de decisão assumem valores inteiros ou binários ($y^k_p \in \{0, 1\}$), tornando o problema inteiro.

Desta forma, a única diferença deste problema inteiro para um problema linear é a última restrição. Se as variáveis de decisão assumirem valores reais ($y^k_p \in [0, 1]$), o problema passa a ser linear.

Na presente formulação são incluídos requisitos de QoS na alocação dos recursos para VPNs. Nesta abordagem cada VPN pode ser caracterizada por uma Qualidade de Serviço. Para estabelecer a diferenciação de serviços foram utilizados 4 classes de serviços, das quais três possuem QoS e uma é *Best Effort*. As classes de serviços utilizadas são apresentadas na Tabela 6.1.

Classes de Serviços	Tipos de Dados
Classe 1	Voz
Classe 2	Vídeo
Classe 3	Dados
Classe 4	*Best Effort*

Tabela 6.1: Tipos de Classes de Serviços utilizados nesta abordagem.

Durante a alocação de recursos na rede para as VPNs, as características de classes de serviço são respeitadas. Por exemplo, a Figura 6.1 mostra a ordem de prioridade na alocação dos recursos na rede.

Se não houver recursos suficientes para alocação de todas as VPNs, algumas VPNs devem ser rejeitadas. Essa ordem de rejeição é contrária à ordem de alocação mostrada na Figura 6.1.

Como pode ser notado, no modelo matemático descrito no presente capítulo, é levado em consideração somente o preço e o fluxo da VPN, não se leva em consideração a classe de serviço da VPN. Entretanto, o valor da métrica preço é obtido em função da classe de serviço. Assim, indiretamente a classe de serviço influencia na decisão de controlar a admissão das VPNs.

Assim sendo, serviços com preços mais altos (voz e vídeo) sempre terão preferência

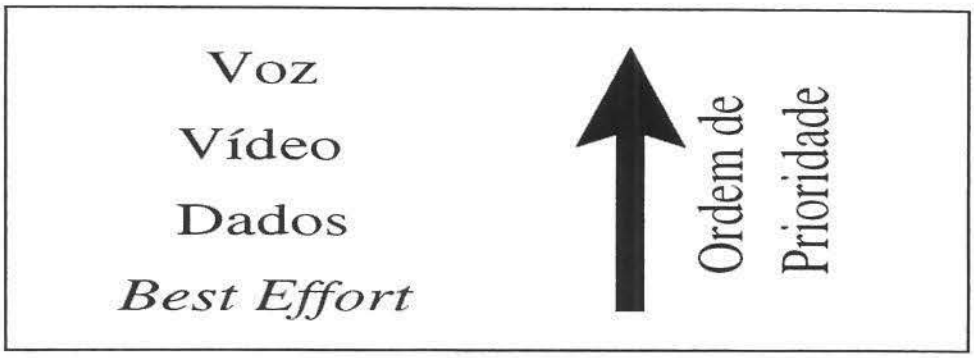

Figura 6.1: Ordem de prioridade na alocação de recursos na rede.

frente aos serviços de dados. Os serviços de dados sempre terão preferência frente os serviços *Best Effort*.

A métrica classe de serviço influencia diretamente em uma medida de QoS, conhecida como *delay* (atraso).

Como é sabido, o *delay* dos serviços de voz e vídeo deve ser menor que os serviços de dados, e o de dados menor que o serviço *Best Effort*. Essa análise é feita na hora de escolher os caminhos na rede.

Para encontrar os caminhos na rede foi utilizada uma generalização do algoritmo de Dijkstra, introduzido em 3.5. Na presente abordagem, continua sendo utilizado o valor de K (número limite) para limitar os caminhos possíveis encontrados para cada serviço. Agora, entretanto, esse valor de K é influenciado pela métrica de classe de serviço.

Dessa forma, os K caminhos encontrados por serviço devem obedecer certos critérios para serem escolhidos na hora de construir o problema de *multicommodity flow* a ser analisado (função objetiva, matriz, e limites).

Estes critérios são apresentados na Tabela 6.2.

Classes de Serviços	Dos K melhores caminhos possíveis encontrados
Classe 1 Classe 2	Os caminhos escolhidos devem possuir um número de *hops* menor ou igual ao diâmetro da rede.
Classe 3	Os caminhos escolhidos devem possuir um número de *hops* menor ou igual ao diâmetro da rede multiplicado por 1.3 (valor de relaxamento da classe de dados).
Classe 4	São usados todos os K caminhos encontrados. Não há restrição.

Tabela 6.2: Cálculo do *delay*.

Com esta abordagem, são usados caminhos respeitando sempre os *delays* das classes de

serviços. Assim, o problema de *multicommodity flow* é formulado levando em consideração a sensibilidade no atraso.

6.1 Descrição do Ambiente de Software

Para comprovar e validar a eficiência do modelo proposto foi utilizado o mesmo esquema de programas construído no capítulo anterior. Alguns módulos, entretanto, tiveram que ser modificados para tratar esta abordagem com QoS.

Módulos de programas alterados:

Módulo gera rede e VPNs(*commodities*)

Foi alterado o programa para gerar as VPNs. No arquivo das VPNs cada linha contém: nó de origem; nó de destino; fluxo (banda passante exigida pela VPNs); classe de serviço e preço da VPN. Os preços são gerados em função do fluxo e da classe de serviço, de acordo com a fórmula abaixo:

$$\boxed{\text{Preço} = [(\text{Fator do Custo x Fluxo}) \ / \ \text{Classe de Serviço}]}$$

O fator do custo foi estabelececido como 1500. Detalhes de implementação podem ser obtidos no apêndice A.

Módulo Achar Caminhos

A função deste módulo é achar os caminhos possíveis para cada *commodity* na rede. Para isso, foi utilizada uma modificação do algoritmo de Dijkstra, introduzido em 3.5. Entretanto, os caminhos escolhidos por esse algoritmo passam por uma análise e, dependendo da classe de serviço da VPN, alguns caminhos são descartados.

Como descrito anteriormente, o valor de K continua sendo usado para relacionar os K melhores caminhos, entretanto na hora de montar o problema de *multicommodity flow* inteiro (função objetiva, matriz e limites) nem todos os K caminhos são usados. Por exemplo, para as classes de serviço sensíveis ao atraso no tráfego de dados, são escolhidos caminhos com uma quantidade menor de nós. Esses critérios de escolha são explicados na tabela 6.2.

Nota-se, então, uma restrição na escolha dos caminhos por VPNs, que irá influenciar no processo de escolha do módulo *multicommodity flow*.

Módulo *multicommodity flow*

Esse módulo foi modificado para se adaptar à presente formulação, ou seja, para resolver o problema de maximização. Os resultados dos exemplos numéricos gerados são apresentados mais à frente.

6.2 Exemplo Numérico

Para ilustrar a solução do problema de alocação de recursos para VPNs *multiclasses* em redes baseadas em *switchlets*, utilizou-se a mesma rede ilustrada no capítulo 5 (Figura 5.1)[MR99]. A matriz de tráfego das VPNs é apresentada na tabela 5.1. As classes de serviço dessas VPNs estão especificadas na Tabela 6.3. O preço de cada VPN está na Tabela 6.4. A intenção é alocar recursos para essas VPNs na rede (Figura 5.1). A solução pode ser vista na Tabela 6.5.

Nós	1	2	3	4	5	6	7	8
1	-	1	1	2	4	2	1	1
2	1	-	2	4	4	4	2	1
3	2	1	-	4	4	4	1	2
4	4	1	4	-	4	2	4	1
5	2	1	2	2	-	1	1	2
6	4	1	2	1	4	-	2	4
7	4	1	1	1	2	1	-	1
8	1	1	2	1	2	1	4	-

Tabela 6.3: Matriz de classe de serviço. Os valores internos à matriz representam as diferentes classes de serviços, de acordo com a Tabela 6.1, para cada VPN.

Assim, observa-se o acréscimo de duas métricas (classe de serviço e preço) para cada VPN em relação ao mesmo exemplo do capítulo 5. A métrica classe de serviço foi inserida para implementar QoS. A métrica preço foi inserida para maximizar o lucro do VSP.

Cada caminho gerado corresponde a uma variável na função objetiva. A complexidade do algoritmo aumenta, particularmente, em função do número de *links* e do número de serviços.

Nós	1	2	3	4	5	6	7	8
1	-	9000	12000	2250	1500	2250	7500	12000
2	12000	-	5250	3000	1875	3375	4500	6000
3	750	10500	-	375	3000	2625	12000	5250
4	1125	12000	3000	-	4125	3750	1125	4500
5	6750	12000	750	2250	-	1500	6000	6750
6	1125	9000	1500	7500	2625	-	7500	6000
7	1875	19500	24000	4500	15750	15000	-	7500
8	12000	31500	19500	4500	23250	24000	1875	-

Tabela 6.4: Matriz de preço. Os valores internos à matriz representam os preços, em Reais (R$), para cada VPN.

Variáveis Analisadas	Valores Obtidos
Lucro do VSP	396759
Tempo de Execução	00:00:12.46 *
Número de Switches	8
Número de *Links*	28
Número de *Commodities*	56
Caminhos Gerados	195

*hora:minuto:segundo

Tabela 6.5: Resultado da Alocação das VPNs na rede da Figura 5.1. Valores gerados utilizando *switchlets* (*node splitting*) para serviços *multiclasses*.

6.3 Eficiência do método proposto

Para avaliar a viabilidade da adoção do método proposto em uma rede operacional, são verificados os mesmos problemas utilizados no capítulo 5, tabela 5.3. Essa tabela mostra 18 exemplos de redes, geradas com diferentes números de nós e *links*, variando, assim, o grau de conectividade da rede.

Desta forma, na Tabela 5.3 podem ser observado exemplos tanto para grafos densos como para grafos esparsos, variando, também, o número de serviços. Assim, foi possível verificar quais os fatores (número de nós, números de *links* ou número de *commodities*) que mais influenciam na complexidade do problema (tempo de execução).

Foram realizados três tipos de testes com os mesmos dados, variando-se a quantidade de caminhos escolhidos por VPN.

No primeiro teste, foram escolhidos os 100 melhores caminhos por VPN ($k = 100$). Quanto maior a quantidade de caminhos, maior o poder de escolha do CPLEX, melhorando, assim, o valor do custo total para acomodar todas as VPNs, mas, por outro lado, piora o tempo de execução do algoritmo. Os resultados são mostrados na Tabela 6.6.

Observa-se na Tabela 6.6 que os tempos de execução são apropriados para o dimensionamento dinâmico da rede (todos menores que 20 segundos), mas, por outro lado, se for comparado com algoritmos de roteamento dinâmico de rede (por exemplo, o protocolo PNNI em redes ATM, que, segundo o ATM Forum, deve demorar no máximo 8 segundos para conseguir estabelecer uma nova rota para uma conexão SVC[For96]) não é satisfatório. Assim, o tempo de execução deve ser melhorado.

Problema	Caminhos	Lucro do VSP	Tempo de Execução *
1	869	92934	00:00:02.03
2	1414	145719	00:00:03.18
3	331	130260	00:00:02.31
4	442	173691	00:00:03.34
5	1315	138756	00:00:02.85
6	1427	184713	00:00:04.01
7	2006	372204	00:00:06.41
8	1660	197451	00:00:04.65
9	4069	383814	00:00:12.74
10	2243	163206	00:00:05.80
11	5521	454176	00:00:17.98
12	2489	169644	00:00:06.22
13	5586	422250	00:00:18.71
14	6776	461343	00:00:19.07
15	5726	470175	00:00:15.61
16	2862	175977	00:00:06.15
17	1864	177213	00:00:07.35
18	4074	496776	00:00:15.63

*hora:minuto:segundo

Tabela 6.6: Resultados usando K=100.

Dessa forma, foi realizado um segundo teste, no qual foram escolhidos os 30 melhores caminhos por VPN ($k = 30$). Os resultados são apresentados na Tabela 6.7. Comparando os resultados com a Tabela 6.6, percebe-se que o tempo de execução não ultrapassou 9 segundos para todos os problemas, e o mais importante é que a função objetiva manteve-se idêntica aos resultados usando $k = 100$.

Problema	Caminhos	Lucro do VSP	Tempo de Execução
1	305	92934	00:00:01.79
2	502	145719	00:00:02.73
3	331	130260	00:00:02.76
4	442	173691	00:00:03.36
5	405	138756	00:00:02.38
6	447	184713	00:00:03.45
7	767	372204	00:00:05.42
8	888	197451	00:00:03.73
9	1877	383814	00:00:08.30
10	895	163206	00:00:03.90
11	2189	454176	00:00:09.53
12	1017	169644	00:00:03.90
13	2268	422250	00:00:09.88
14	2553	461343	00:00:09.35
15	2306	470175	00:00:09.14
16	1076	175977	00:00:03.96
17	629	177213	00:00:04.49
18	1394	496776	00:00:09.62

Tabela 6.7: Resultados usando K=30.

A Tabela 6.8 mostra os resultados obtidos ao se limitar o número de caminhos por VPN para 10 ($k = 10$). O tempo de execução não ultrapassou 8 segundos para todos os 18 problemas, entretantdo, como era de se esperar, o valor da função objetiva, comparado com as Tabelas 6.6 e 6.7, não foi igual em todos os casos. Isso acontece porque muitos caminhos possíveis são descartados, os quais poderiam ter um custo menor.

Observa-se, nos três exemplos, que, variando-se o número de VPNs para a mesma topologia de rede, o número de caminhos gerados aumenta linearmente. Isso implica num aumento na mesma proporção linear no tempo de execução total do algoritmo. Entretanto, por ser uma heurística, não se pode garantir que se encontra uma solução ótima. Além disso não se tem como calcular o quanto o valor ótimo é melhor que o valor encontrado na função objetiva. No entanto, a possibilidade de encontrar soluções subótimas em um tempo de execução hábil pode ser mais importante do que a obtenção da solução ótima. Isso é importante em problemas de dimensionamento dinâmico de rede, nos quais o tempo de execução da otimização é um fator crucial.

Por fim, constata-se, também, que não há problema de escalabilidade para os problemas em questão, na metodologia proposta, contornando-se de certa forma, a característica do problema de *multicommodity flow* inteiro ser NP-difícil.

Problema	Caminhos	Lucro do VSP	Tempo de Execução
1	134	92934	00:00:01.72
2	221	145719	00:00:02.65
3	242	130260	00:00:02.28
4	321	173691	00:00:03.30
5	145	138756	00:00:02.26
6	167	184713	00:00:03.31
7	367	372204	00:00:05.18
8	388	197451	00:00:03.45
9	803	383814	00:00:07.36
10	351	163206	00:00:03.50
11	834	454176	00:00:08.04
12	391	169644	00:00:03.53
13	873	422250	00:00:08.18
14	937	460758	00:00:07.96
15	908	470175	00:00:07.93
16	410	175977	00:00:03.53
17	257	177213	00:00:03.72
18	581	496776	00:00:08.11

Tabela 6.8: Resultados usando K=10.

Os resultados obtidos com a metodologia desenvolvida mostraram-se eficazes na efetivação do dimensionamento dinâmico de redes.

Capítulo 7

Conclusão

Switchlet representa um grande potencial na efetivação de redes programáveis. Apesar desta técnica ter sido desenvolvida para a tecnologia ATM, o conceito de *switchlets* pode ser extensível a outras tecnologias de comutadores.

Mecanismos automáticos para alocação de recursos na implementação de VPNs são de capital importância para projetos eficientes de redes virtuais.

Partindo dessa premissa, este trabalho introduziu métodos baseados em *multicommodity flow* para alocação eficiente de recursos em redes baseadas em *switchlets*:

- No primeiro método, as VPNs não foram diferenciadas, ou seja, não houve diferenciação de serviço. O objetivo com o modelo desenvolvido foi minimizar o custo e maximizar a utilização dos recursos na rede para alocar todas as VPNs do projeto.
- No segundo método, foi implementado tráfego com QoS junto com o serviço de melhor esforço (*Best-Effort*), permitindo, assim, analisar as VPNs de maneira diferenciada. O objetivo com o uso deste método foi maximizar o ganho ou o lucro do *VPN Service Provider* em alocar recursos na sua rede para as diferentes VPNs.

A otimização de recursos tratado na presente tese de mestrado é um problema NP-difícil, isto é, não é conhecido nenhum algoritmo capaz de resolver todas as instâncias desse problema em tempo polinomial (conjectura-se que tal algoritmo não exista). Entretanto, isso nem sempre significa que o problema seja intratável na prática, ou seja, é possível construir algoritmos que, apesar de exponenciais no pior caso, conseguem resolver um grande número de instâncias em tempo razoável.

Dessa forma, criaram-se modelos matemáticos para resolver o problema de *multicommodity flow* inteiro em tempo real. Para tornar o tempo de execução satisfatório, e, assim, viabilizar os métodos propostos, foi necessário utilizar alguns algoritmos conhecidos na literatura, a saber: *branch-and-bound*; busca em profundidade e generalização do algoritmo de Dijkstra.

Para certificação e validação dos métodos desenvolvidos foi implementado um conjunto de programas para analisar os problemas gerados para testes. Cada problema era constituído de uma rede (variando a sua quantidade de *links* e nós) e um conjunto de VPNs. Com os resultados obtidos foi possível chegar à conclusão de que os métodos desenvolvidos mostraram-se viáveis para implementação em tempo real e, consequentemente, para realizar o dimensionamento dinâmico de VPNs em redes de computadores.

Entretanto, pelo fato do problema ser NP-difícil, não se pode garantir que os modelos propostos encontrarão soluções para todas as instâncias desse problema. Pelo fato de se utilizarem métodos para melhorar a escalabilidade e diminuir o tempo de execução dos problemas (como: K=100; K=30 e K=10), também não se pode garantir que os modelos encontrarão soluções factíveis, mesmo quando existam. Além disso, não há como calcular o quanto o valor ótimo é melhor do que o valor encontrado na função objetiva. No entanto, a possibilidade de se encontrarem soluções subótimas em um tempo de execução hábil pode ser mais importante do que a obtenção da solução ótima, dado que o fator importante no dimensionamento dinâmico de redes é o tempo de execução dos algoritmos de alocação de recursos.

O trabalho desenvolvido nesta dissertação pode ser estendido de diversas formas:

- Neste trabalho foi utilizada a técnica, até então em evidência, de redes programáveis conhecida como *switchlets*. Assim, devido a existência de outras técnicas, como *spawning*[LC97], evidencia-se a possibilidade de se estender o método aqui proposto para estudo de *spawning*;

- Um campo de pesquisa em redes sob intensa investigação e desenvolvimento e na tentativa de inserir qualidade de serviço no mundo IP, neste sentido nota-se a possibilidade de verificar a viabilidade de integrar os métodos desenvolvidos com os protocolos de qualidade de Serviço (QoS), como: MPLS, DiffServ, etc;

- Situações comuns são aquelas onde a rede tem todos os seus *links* em situação de sobrecarga, ou quando a carga não está bem distribuída na rede, apresentando alguns *links* mais congestionados que outros. Muito embora a rede possa estar bem provisionada, os protocolos convencionais de roteamento dinâmico como RIP, OSPF e IS-IS, sempre baseiam a sua escolha de rota na métrica, ou seja, no menor caminho entre nós consecutivos. Essa situação pode ser especulada num cenário onde o caminho mais curto entre dois pontos passa por um *link* congestionado, implicando em falha no atendimento a certa requisição de QoS feita aos protocolos específicos para esse fim. Dessa forma, o trabalho desenvolvido pode ser estendido para contribuir na racionalização da distribuição de carga na rede, ajustando a carga nos diversos *links* de um domínio, por exemplo, de forma a retirar carga dos *links* mais congestionados e distribuí-la aos menos congestionados;

- De maneira geral, a característica deste trabalho em analisar somente as VPNs para alocação de recursos nas redes é relevante. Mas, por outro lado, é interessante saber como é feita a análise dos serviços que são trasmitidos internamente a essas VPNs. Esses serviços podem ser totalmente diferentes uns dos outros na mesma VPN. Por exemplo, quando um usuário contrata uma VPN de um VSP, geralmente uma determinada qualidade de serviço acompanha aquela VPN, o usuário, porém, pode decidir utilizar sua VPN para transportar diferentes serviços com diferentes requisições ou características (tráfego administrativo, tráfego de Voz, tráfego para a Internet). Assim, verifica-se a possibilidade de aplicar os métodos desenvolvidos em serviços levando em consideração diferentes tipos de tráfego em cada VPN;
- Utilizar a metodologia desenvolvida para criar um simulador. Este simulador terá como objetivo fornecer para o VSP a melhor maneira de imputar preços para as VPNs visando maximizar seu lucro e minimizar o custo total do projeto.

Bibliografia

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, e James B. Orlin. ***Network Flows: Theory, Algorithms and Applications***. Prentice Hall, 1993.

[Bis98] J. Biswas, et al. **The IEEE P1520 Standards Initiative for Programmable Network Interfaces.** *IEEE Communications Magazine, Special Issue on Programmable Networks*, Outubro de 1998.

[CDZ97] Kenneth L. Calvert, Matthew B. Doar, e Ellen W. Zegura. **Modeling Internet Topology**. *IEEE Communications Magazine*, Junho de 1997.

[CG99] Columbia University COMET Group. World Wide Web, `http://comet.ctr.columbia.edu/genesis/papers/OPENARCH_99/tsld001.htm`, 1999.

[CKMV99] A. T. Campbell, I. Katzela, K. Miki, e J. Vicente. **Open Signaling for ATM, Internet and Mobile Networks (OPENSIG'98)**. *Computer Communication Review (CCR)*, 29(1):97–108, Janeiro de 1999.

[CL95] Thomas M. Chen e Stephen S. Liu. ***ATM Switching Systems***. Artech House, Inc., 1995.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, e Ronald L. Rivest. ***Introduction to Algorithms***. McGraw-Hill Book Company, 1990.

[CMK+99a] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, e D. A. Villela. **The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures**. *Second International Conference on Open Architectures and Network Programming (OPENARCH)*, pp. 115–127, 1999.

[CMK+99b] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, e Daniel Villela. **A Survey of Programmable Networks**. *Computer Communication Review (CCR)*, 29(2):7–23, Abril de 1999.

[For96] ATM Forum. **Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)**, 1996.

[For00] Multiservice Switching Forum. World Wide Web, `http://www.msforum.org`, 2000.

[Gro00] Open Signalling Working Group. `http://comet.columbia.edu/opensig/`, 2000.

[ILO99] ILOG. ***ILOG CPLEX 6.5, User's Manual***. ILOG Press, Março de 1999.

[Isa00] Rebecca Isaacs. **Lightweight, Dynamic and Programmable Virtual Private Networks**. *OPENARCH*, 2000.

[JRF01] Antônio P. Castro Jr., Alexandre T. Rios, e Nelson L. S. Fonseca. **Alocação de Recursos para Redes Virtuais Privadas em Redes Baseadas em *Switchlets***. *SBRC*, (19):684–695, Maio de 2001.

[Laz97] Aurel A. Lazar. **Programming Telecommunication Networks**. *IEEE Network*, 11(5):8–18, Setembro/Outubro de 1997.

[LC97] Aurel A. Lazar e A. T. Campbell. **Spawning Network Architectures**. Technical report, Center for Telecommunications Research, Columbia University, 1997.

[Ltd95] Longman Group Ltd. ***Longman: Dictionary of Contemporary English***. Longman Group Ltd, terceira edição, 1995.

[ML97] J. E. Van Der Merwe e I. M. Leslie. **Switchlets and Dynamic Virtual ATM Networks**. *Proc Integrated Network Management V*, pp. 355–368, Maio de 1997.

[ML98] J. E. Van Der Merwe e I. M. Leslie. **Service-Specific Control Architectures for ATM**. *IEEE Journal on Selected Areas in Communication*, 16(3):424–436, Abril de 1998.

[MPS98] E. Q. V. Martins, M. M. B. Pascoal, e J. L. E. Santos. **The K shortest paths problem**. Technical report, Department of Mathematics, University of Coimbra, Portugal, Junho de 1998.

[MR99] Debasis Mitra e K. G. Ramakrishnan. **A Case Study of Multiservice, Multipriority Traffic Engineering Design for Data Networks**. *Proc. IEEE, GLOBECOM99*, pp. 1071–1083, 1999.

[MS99] David McDysan e Darren Spohn. ***ATM Theory and Applications***. McGraw-Hill Series on Computer Communications, 1999.

[Net96] Ipsilon Networks. **IP Switching: The Intelligence of Routing, the Performance of Switching**. World Wide Web, `http://www.ipsilon.com/productinfo/techwp1.html`, 1996.

[Nic98] K. Nicholas. **Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers**. IETF Internet Draft, `http://search.ietf.org/internet-drafts/`, Agosto de 1998.

[P1500] IEEE Standard for Application Programming Interfaces for Networks P1520. `http://www.ieee-pin.org/`, 2000.

[Pro96] DARPA Active Network Program. World Wide Web, `http://www.darpa.mil/ito/research/anets/projects.html`, 1996.

[Roo97] S. Rooney. **An Innovative Control Architecture for ATM Networks**. *Integrated Network Management V*, pp. 369–380, Maio de 1997.

[Roo98] S. Rooney. ***The Structure of Open ATM Control Architectures***. PhD thesis, University of Cambridge, Computer Laboratory, Fevereiro de 1998.

[RSW99] J-P. Redlich, M. Suzuki, e S. Weinstein. **Virtual Networks in the Internet**. *Second IEEE International Conference on Open Architecture and Network Programming*, pp. 108–114, 1999.

[SQ99] Stardust.com e QoSforum.com. **The Need for QoS**. Technical report, Stardust Technologies, Inc., `http://www.stardust.com` e `http://www.qosforum.com`, Julho de 1999.

[SS99] Michael Sig e Mikkel Sigurd. **MultiCommodity Flow with Integer Solutions**. Technical report, Department of Computer Science, University of Copenhagen, Maio de 1999.

[Tan97] Andrew S. Tanenbaum. ***Redes de Computadores***. Editora Campus, terceira edição, 1997.

[TSS+97] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, e Gary J. Minden. **A Survey of Active Network Research**. *IEEE Communications Magazine*, 35(1):80–86, Janeiro de 1997.

Apêndice A

Detalhes de Implementação

Neste apêndice, mostram-se alguns detalhes na implementação que ficaram implícitos nos capítulos 5 e 6. Alguns desses detalhes foram empregados devido ao fato de haver restrições particulares dos recursos computacionais utilizados.

A.1 Gerador de Topologia de Rede

As estruturas topológicas das redes são tipicamente modeladas usando grafos: com os nós representando os comutadores e os arcos representando conexões direcionadas (*links*) entre os comutadores. Através de grafos é possível modelar caminhos (sequência de nós e *links*) onde fluxos de informações são trocadas entre os nós. Informações adicionais sobre a rede podem ser inseridas na estrutura topológica, simplesmente, associando informações aos nós e *links*. Exemplos de informações associadas aos nós: capacidade de *throughput* e custo. Exemplos de informações associadas aos arcos: banda passante e custo.

Nesse sentido, foi necessário utilizar um gerador de redes consistente e que se adequasse ao formato do grafo requerido pelo presente trabalho, com informações nos nós e arcos, a saber: banda passante e custo.

Assim, foram pesquisados alguns programas conhecidos na literatura para gerar topologias de redes, como: *The Georgia Tech Internetwork Topology Models* (GT-ITM)[CDZ97]; *Tiers*[CDZ97] e *randgraph*. Os dois primeiros programas são geradores construídos pela *Georgia Tech* e utilizam estruturas complexas para gerar as topologias de rede (por exemplo, redes com estruturas hierárquicas, redes PNNI de mesmo grupo e de grupos diferentes, etc), os quais, entretanto, fornecem informações somente para os *links*. O terceiro é um gerador mais simples que gera as topologias de redes usando números pseudo-aleatórios, mas fornecem informação tanto para os *links* como para os nós da rede.

Após uma análise desses programas, foi constatado que para empregá-los seria necessário alterar os seus códigos fontes, visto que os mesmos não geravam redes com os

formatos requeridos. Levando em consideração que o fato de gerar topologias de redes complexas não tem uma importância significativa para o presente trabalho, pois o mais importante é avaliar a viabilidade da adoção dos métodos propostos, decidiu-se por implementar um novo gerador pseudo-aleatório, gerando, assim, topologias de redes simples, que respeitam o formato exigido.

Dessa forma, para gerar os exemplos numéricos foi implementado um gerador de redes. Com este programa, foram gerados 18 redes com diferentes números de nós e *links*, variando, assim, o seu grau de conectividade. Os exemplos gerados estão na tabela 5.3.

Os números pseudo-aleatórios são obtidos usando uma função, da linguagem C, conhecida como `rand()`. Como ilustrado abaixo:

```
RAND_MAX 100
int randomi(float valor){
   return ((int) (valor*rand()/(RAND_MAX+1.0)));
}
```

A função acima gera números randômicos entre 0 e `RAND_MAX`. Para melhorar a aleatoridade da função `rand()`, foi usada a função `srand(stime)`. Dessa forma, a função `rand()` passa a utilizar a hora atual do calendário do sistema como semente para obter um número aleatório:

```
ltime = time(NULL);
stime = (unsigned) ltime/2;
srand(stime);
```

Além de utilizar esta função para gerar os nós e *links*, ela, também, foi utilizada para gerar os custos. Já, para o valor da banda passante esta função não foi utilizada. A informação da banda passante para os *links* possui três valores pré-definidos, a saber:

155Mbps - 311Mbps - 622Mbps

Esses valores foram distribuídos de maneira pseudo-aleatória, ou seja, 50% dos *links* assumiam o valor de 155, 30% assumiam o valor de 311 e 20% assumiam o valor de 622. O valor do custo acompanhou essa distribuição, no sentido de que quanto maior a banda passante do *link* maior será o seu custo. Para isso, foi utilizado o valor de `RAND_MAX` com o objetivo de limitar o escopo da função `rand()`. Assim, para *links* onde a banda passante era de 155 o `RAND_MAX` foi 10, para *links* de 311 o `RAND_MAX` foi 15 e para *links* de 622 o `RAND_MAX` foi 25.

A.2 Gerador de VPNs

Foi construído um programa para gerar as VPNs (fluxos). As mesmas funções utilizadas para obter a aleatoridade na implementação do programa para gerar topologias de redes foram aplicadas no gerador de VPNs.

Para o problema do capítulo 5, cada VPN possui as seguintes informações:

```
Nó de Origem - Nó de Destino - Banda Passante
```

Para o problema do capítulo 6, cada VPN possui as seguintes informações:

```
Nó de Origem - Nó de Destino - Banda Passante - Classe de Serviço - Preço
```

O nó de origem e o nó de destino são obtidos através de valores randômicos. A informação de banda passante assume três valores (3Mbps, 6Mbps e 12Mbps). Estes valores foram distribuídos de maneira pseudo-aleatória, pois de 100 VPNs, 50 assumiam o valor 3, 30 assumiam o valor 6 e 20 assumiam o valor 12. Assim, foi feita a distribuição do valor da banda passante (50% - 3, 30% - 6, 20% - 12).

Já as informações de classe de serviços possuem quatro valores pré-definidos (1, 2 ,3 e 4), como oportunamente explicado no capítulo 6. Estes quatro valores foram distribuídos randomicamente entre as VPNs.

Os preços são gerados em função da banda passante e da classe de serviço, de acordo com a fórmula abaixo:

$$\text{Preço} = [(\text{Fator do Custo x Banda Passante}) \ / \ \text{Classe de Serviço}]$$

Nota-se que a classe de serviço é inversamente proporcional ao preço. Foi considerado que a classe de serviço 1 (voz) e 2 (vídeo) deveriam ser iguais na hora de gerar o preço. Isso porque o preço influencia na admissão das VPNs (como pode ser observado no modelo matemático formulado no capítulo 6). Assim, na implementação, foram geradas somente 3 classes de serviços, a saber:

Classes de Serviços	Tipos de Dados
Classe 1	Voz e Vídeo
Classe 2	Dados
Classe 4	*Best Effort*

Tabela A.1: Classes de Serviços utilizadas na implementação.

O fator do custo foi estabelececido como 1500 (valor em Reais, 1500,00). Esse número foi escolhido após observações dos valores do custo de trafegar uma VPN por um determinado caminho. O objetivo foi evitar que o VSP tivesse prejuízo ao trafegar as VPNs na sua rede, ou seja:

O preço da VPN tem que ser maior que o custo de trafegar a VPN na rede.

A.3 Modificação do algoritmo de busca em profundidade

Uma modificação do algoritmo de busca em profundidade foi utilizada para encontrar os caminhos possíveis entre a origem e o destino de uma VPN.

Nesta modificação, os caminhos escolhidos para cada VPN obedecem a uma restrição, isto é, são escolhidos somente caminhos com um número de nós (*hops*) menor ou igual ao total de nós da rede dividido por dois. Dessa maneira, a quantidade de caminhos selecionados é um subconjunto do total de caminhos possíveis encontrados pelo algoritmo de busca em profundidade.

Esta modificação foi adotada em virtude do grande número de caminhos obtidos para abordagens simples, que impactaram no tempo de execução dos problemas, inviabilizando o modelo proposto. Mas o fator que mais influenciou na adoção desta restrição foi do espaço de memória ser insuficiente para armazenar a grande quantidade de caminhos para cada VPN. Estava ocorrendo constantemente problema de *buffer overflow* no módulo achar caminhos.

Esse algoritmo é uma modificação do algoritmo de busca em profundidade descrito em [CLR90]. Esta modificação é apresentada pelo pseudocódigo da Figura A.1.

A.4 Achando o Diâmetro da Rede

Para cálcular o diâmetro da rede foi empregado o algoritmo de busca em largura[CLR90].

Achar o valor do diâmetro da rede significa: encontrar o menor número de nós (*hops*) os quais permitam que todos os nós da rede alcancem todos os outros nós da rede.

Para encontrar o valor do diâmetro da rede foi necessário realizar uma busca em largura de todos os nós da rede para todos os outros nós. Para realizar a busca, esse algoritmo monta uma quantidade de estruturas em árvores igual ao número de nós da rede. O maior valor da altura dentre todas as árvores é o diâmetro da rede.

O algoritmo utilizado para achar o diâmetro da rede é baseado no algoritmo de busca em largura descrito em [CLR90]. Este algoritmo é apresentado pelo pseudocódigo da Figura A.2.

```
Algoritmo → Modificação do Algoritmo de Busca em Profundidade

 para cada nó u ∈ G(N) faça
   color[u] ⟵ 1
   π[u] ⟵ 0
 para cada VPN k ∈ K faça
   qtd_nós ⟵ 0 #variável global#
   ATCP(u,w) #onde:  u=nó de origem e w=nó de destino#

 ATCP(u,w)
   color[u] ⟵ 2
   qtd_nós ⟵ qtd_nós + 1
   se (qtd_nós ≤ N/2) então
     para cada nó v ∈ Adj[u] faça
       se (color[v] = 1) então
         π[u] ⟵ v
         se v = w
          então trilhar caminho em π
          senão ATCP(v,w)
   π[u] ⟵ 0
   color[u] ⟵ 1
   qtd_nós ⟵ qtd_nós - 1
 fim ATCP

fim algoritmo
```

Figura A.1: A Figura mostra o pseudocódigo da modificação feita no algoritmo de busca em profundidade.

```
Algoritmo → Achar o diâmetro da rede

nível_árvore  vetor
nó_visitado  vetor
fila_nó   vetor
diametro   vetor
ler arquivo com a topologia da rede
para j=1 até total de nós faça
  k ⟵ 0
  nó_visitado[j] ⟵ 1
  nível_árvore[j] ⟵ 0
  fila_nó[k] ⟵ j
  faça enquanto fila_nó ≠ 0
    nó ⟵ fila_nó[0]
    para cada nó w ∈ Adj[nó] faça
      se nó_visitado[w] = 0 então
        nó_visitado[w] ⟵ 1
        nível_árvore[w] ⟵ nível_árvore[nó] + 1
        fila_nó[k+1] ⟵ w
        k = k + 1
    para i = 0 até total de nós faça
      fila_nó[i] ⟵ fila_nó[i+1]
    k ⟵ k - 1
    nó_visitado[nó] ⟵ 2
  maior ⟵ 0
  para i = 0 até total de nós faça
    se maior < nível_árvore[i] então
      maior ⟵ nível_árvore[i]
  diâmetro[j] ⟵ maior
  j ⟵ j + 1
maior ⟵ 0
para i = 0 até total de nós faça
se maior < diâmetro[i] então
  maior ⟵ diâmetro[i]

fim algoritmo
```

Figura A.2: A Figura mostra o pseudocódigo do algoritmo para encontrar o diâmetro da rede.