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

Antônio Pires de Castro Jr, Alexandre Theotonio T. Rios, Nelson Luis S. Fonseca
Instituto de Computação - Universidade Estadual de Campinas
Caixa Postal 6176, 13083-970, Campinas - SP - Brasil
e-mail: {apcastro, alexrios, nfonseca}@ic.unicamp.br

Resumo

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 multiplexador.

Este trabalho 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.

Abstract

Network programming is key to the deployment of new telecommunication services. *Switchlets* is a network programming technique which allows a fine granularity for the allocation of the resources of a switch.

This paper introduces a technique for the allocation of resources to Virtual Private Networks (VPN) in networks based on *switchlets*. The technique is based on *multicommodity flow* and it can be implemented in real time.

Palavras-Chaves

Redes programáveis, Dimensionamento de Redes, Avaliação de Desempenho, Redes Virtuais Privadas.

1 Introdução

A habilidade para rapidamente criar, desenvolver e gerenciar novos serviços em telecomunicações é o principal fator que direciona a comunidade de pesquisa em redes programáveis. O impacto do resultado neste campo de pesquisa irá influenciar usuários, provedores de serviços e fabricantes de equipamentos no setor de telecomunicações.

A separação do hardware de comunicação (*switching fabrics, routing engines*) do software de controle é fundamental para a programabilidade da rede. Atualmente 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.

Switchlets, uma técnica de redes programáveis[2, 4, 3], surgiu numa tentativa de facilitar o controle e particionamento dos recursos de comutadores, simplificando o desenvolvimento de novos serviços, de modo a introduzi-los de forma automatizada. Uma *switchlet* é um subconjunto de recursos e de softwares de sinalização de um multiplexador que pode ser alocado/programado independentemente das outras *switchlets* do multiplexador.

Redes Virtuais (VN)[6] são entidades de serviços customizadas para determinado tipo de tráfego e/ou determinado grupo de usuários (VPN). As redes virtuais, então, são necessárias para diminuir o custo e a complexidade de prover uma rede física dedicada ao tráfego de serviços distintos.

As redes virtuais privadas são redes virtuais proprietárias que utilizam a infra-estrutura da rede pública para interligar as estações de um grupo de usuários, tipicamente estações de uma corporação. As VPNs devem oferecer escalabilidade, custo efetivo e a possibilidade de controlar e gerenciar a criação das redes privadas sobre a infra-estrutura pública. Assim sendo, surge a necessidade de alocar recursos de uma infra-estrutura física, para VPNs visando a maximização do uso eficiente dos recursos da rede.

Tendo em vista que o desenvolvimento de arquiteturas de sinalização de redes é, em muitas situações, manual, consumindo tempo e custo no processo, estão sendo pesquisadas técnicas para tornar as redes programáveis, ou seja, para facilitar a rápida criação, desenvolvimento e gerenciamento de arquiteturas virtuais *on-the-fly*.

O presente artigo introduz um método para a alocação eficiente dos recursos de uma rede baseada em *switchlets*, a fim de permitir a coexistência de diferentes redes virtuais privadas[6]. Propõe-se um modelo baseado em *multicommodity flow* para o problema de alocação de recursos em redes baseadas em *switchlets*. Evidencia-se, também, que o método proposto é factível de ser implementado em tempo real.

Serão apresentados, primeiramente, os conceitos e as metodologias utilizadas para a resolução do problema de particionamento de recursos, a saber: *switchlets*, na seção 2; *node splitting*, na seção 3; e *multicommodity flow* para programação inteira, na seção 4. Alguns exemplos numéricos serão apresentados na seção 6. E, por fim, conclusões serão derivadas na seção 7.

2 *Switchlets*

Uma *switchlet* é um subconjunto dos recursos de sinalização e de buffers de um multiplexador (switch).[9] *Switchlets* de um mesmo multiplexador 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.

A técnica de *switchlet* apresenta-se como uma nova alternativa para controlar e gerenciar redes ATM, dado que permiti que diferentes arquiteturas de controle[7] estejam operacionais simultaneamente.

Switchlets, em diferentes multiplexadores ATM, podem ser combinadas para formar uma rede virtual ATM. Um meio físico pode ter várias redes virtuais. Cada rede virtual criada dessa maneira pode, potencialmente, usar diferentes mecanismos de controle e de gerenciamento, os quais são coletivamente chamados de arquitetura de controle.[3, 9]

Um exemplo é mostrado na Figura 1, que retrata uma rede com cinco switches, e com três

redes virtuais.

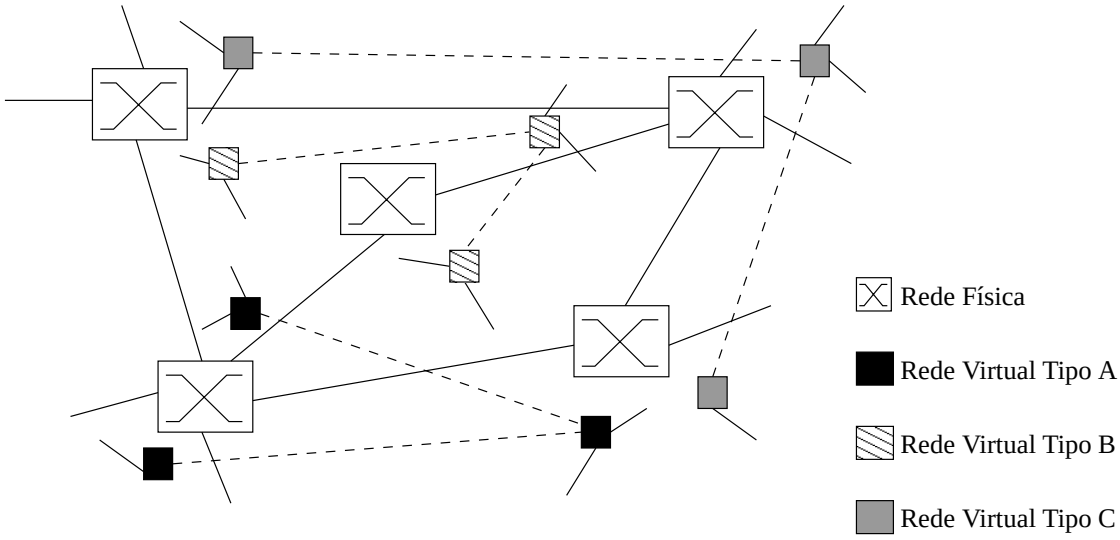

Figura 1: Rede Virtual ATM com diferentes arquiteturas de controle.

Switchlets permitem novas arquiteturas de controle serem introduzidas em rede de comunicações 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.[10]

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

O particionamento de um multiplexador em *switchlets* requer a especificação das *switchlets* como o subconjunto de recursos físicos disponíveis em um multiplexador. No presente artigo, o único recurso do multiplexador em questão, é a banda passante.

3 *Node Splitting*

Conforme mencionado anteriormente, as *switchlets* de um multiplexador podem ser alocadas independentemente. Desse modo, a granularidade do problema de alocação de recursos engloba subconjuntos de recursos de um multiplexador. Para representar os multiplexadores de uma rede baseada em *switchlets* por um grafo, é necessário observar a restrição de que a banda passante dos canais que chegam e que saem de um multiplexador é compartilhada por todas as *switchlets* do multiplexador.

Para representar as redes lógicas criadas usando *switchlets*, aplica-se a técnica de *node splitting*[1] nos vértices do grafo que representa a rede de multiplexadores.

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 2).

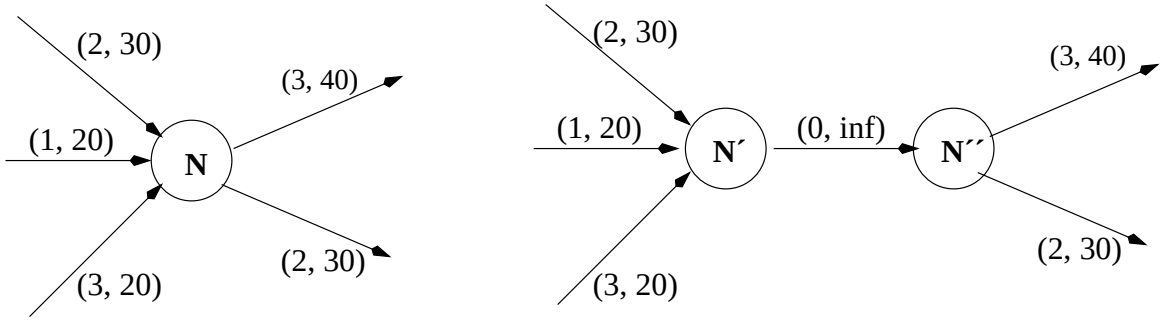

Figura 2: Exemplo de *node splitting*.

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.

4 Problemas de *Multicommodity Flow*

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 minimizar o custo total do projeto. Utiliza-se, neste artigo, uma solução baseada em *multicommodity flow*.

Esta seção descreve o problema de *multicommodity flow* (MCF)[1, 8]. Este problema distribui o tráfego da rede de forma a minimizar o custo ou o retardo. 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.

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.

Seja uma rede direcionada G com: um conjunto de nós N; um conjunto de arcos A e um conjunto de serviços 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 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 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 *multicommodity flow* inteiro[8] pode ser formulado da seguinte forma:

$$Minimize \sum_{k \in K} \sum_{p \in P(k)} c_p q^k y_p^k \quad (1)$$

$$subject\ to \sum_{k \in K} \sum_{p \in P(k)} q^k y_p^k \partial_a^p \leq d_a, \forall a \in A, \quad (2)$$

$$\sum_{p \in P(k)} y_p^k = 1, \forall k \in K, \quad (3)$$

$$y_p^k \in \{0,1\}, \forall p \in P(k), \forall k \in K. \quad (4)$$

A função objetiva (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 (2) limita a soma de todos os fluxos que utilizam o arco a à capacidade d_a;
- A restrição (3) limita a designação do serviço k à somente um dos caminhos de $P(k)$;
- A restrição (4) garante que nenhum fluxo é divido entre dois ou mais caminhos ou enviado parcialmente por um caminho. Esta restrição junto com a restrição (3) garante 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*.

O algoritmo *branch-and-bound* é um método eficaz para resolver problemas de otimização NP-difícil, que é o caso do problema tratado neste artigo. Para resolver um problema, o algoritmo *branch-and-bound* analisa o espaço de soluções 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.

O software CPLEX *Mixed Integer optimizer* versão 6.5 resolve modelos nos quais uma ou mais variáveis são inteiras. O CPLEX Mixed Integer optimizer, ou MIP optimizer, explora o algoritmo de *branch-and-bound*.

5 Metodologia

Foi construído um esquema de programas para comprovar e validar o algoritmo desenvolvido. Esta seção apresenta o modelo utilizado para resolver o problema de MCF usando *switchlets*. A relação entre os diversos conceitos e as técnicas apresentadas nas seções anteriores são apresentadas na presente seção.

5.1 Descrição dos Módulos

Foram desenvolvidos quatro módulos de programas:

Módulo gera rede e serviços(*commodities*)

Este módulo é constituído de dois programas: um para gerar a topologia das redes e outro para gerar os serviços. Tanto a topologia, quanto os serviços são obtidos com base em valores aleatórios fornecidos por um gerador de números pseudo-aleatórios.

Assim, são gerados dois arquivos: um para a rede e outro para os serviços/*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 aquivo dos serviços cada linha contém: nó de origem; nó de destino e fluxo (banda passante exigida pelo serviço).

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:

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

Tanto o módulo "gera rede e serviços", 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*, foi utilizada uma modificação do algoritmo de busca em profundidade.

Módulo CPLEX

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. No nosso caso, todas as variáveis foram setadas como binárias. O CPLEX *Mixed Integer optimizer*, ou o otimizador MIP, explora o algoritmo de *branch-and-bound*.

Esse módulo recebe um arquivo com a formulação do problema de *multicommodity flow* inteiro, chama as funções do módulo MIP do CPLEX para resolver o problema e fornecer os resultados. Alguns resultados dos exemplos numéricos gerados são apresentados mais à frente.

5.2 Descrição do Problema

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

Desse modo, o principal objetivo deste artigo é maximizar o uso de recursos (banda passante) em redes ATM, com diferentes tipos de serviços. Existe, também, a necessidade da rede conseguir prover as requisições dos usuários de maneira dinâmica, ou seja, *on-the-fly*. Este último é alcançado com o uso de *switchlets*.

5.3 Descrição do Ambiente para geração de exemplos numéricos

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.

6 Exemplos Numéricos

6.1 Um Exemplo

Para ilustrar a solução do problema de alocação de recursos para VPNs em redes baseadas em *switchlets*, utilizou-se a rede ilustrada na Figura 3[5], na qual os rótulos nos arcos representam os recursos. A solução pode ser vista na Tabela 1.

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

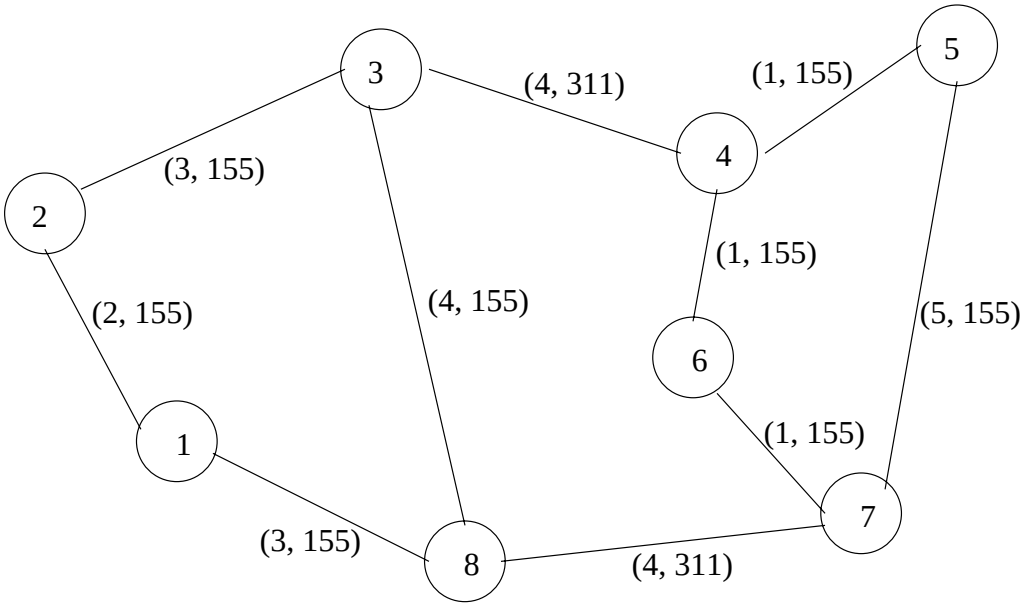

Figura 3: Topologia da Rede.

Variáveis Analisadas	Valores Obtidos
Custo Total	32496
Tempo de Execução	02s
Número de Switches	8
Número de Links	48
Número de *Commodities*	56
Caminhos Gerados	1528

Tabela 1: Valores gerados utilizando *switchlets* (*node splitting*).

6.2 Eficiência do método proposto

Para avaliar a viabilidade da adoção do método proposto neste artigo em uma rede operacional, foram gerados 17 redes com diferentes números de nós e links, variando, assim, o grau de conectividade da rede. A Tabela 2 lista os exemplos gerados.

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 todos os possíveis caminhos 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 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

Tabela 2: Problemas para teste.

Problema	Caminhos	Função Objetivo	Tempo Execução
1	547	7191	00s
2	872	9531	00s
3	186	12615	00s
4	265	15309	00s
5	11228	3810	26s
6	16441	4131	54s
7	96636	14697	1832s
8	3506	18549	03s
9	6509	33561	10s
10	11468	22371	51s
11	27015	64449	323s
12	13013	15981	64s
13	29144	Infactível	328s
14	97412	Infactível	4670s

Tabela 3: Explorando todos os caminhos possíveis.

Problema	Caminhos	Função Objetivo	Tempo Execução
1	547	7191	00s
2	872	9531	00s
3	186	12615	00s
4	265	15309	00s
5	2800	4518	01s
6	4100	5982	01s
7	6500	27585	05s
8	3214	18774	03s
9	6218	33825	08s
10	4003	30252	05s
11	9004	88089	35s
12	3921	20394	05s
13	9003	Infactível	26s
14	9700	Infactível	32s
15	9601	Infactível	34s
16	4300	42294	06s
17	4300	Infactível	07s

Tabela 4: Limitando em 100 caminhos por *commodity*.

No segundo teste, foram escolhidos 100 caminhos possíveis para cada *commodity* (Tabela 4). No terceiro teste, foram escolhidos 50 caminhos possíveis para cada *commodity* (Tabela 5). 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. 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 continua sendo aceitável. Observa-se que o tempo de execução aumenta exponencialmente em função do número de arcos, como esperado.

Observa-se na Tabela 3 que, à medida que o número de enlaces da rede aumenta, o número de caminhos gerados aumenta exponencialmente. Isso implica um 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 a NP-dificuldade do problema de *multi-commodity flow* inteiro, é proposto um número limite de caminhos encontrados por serviço[5]. A Tabela 5 mostra os resultados obtidos ao limitar o número de caminhos por serviço para apenas 50. Comparando esses resultados com a Tabela 3, percebe-se que o tempo de execução foi de apenas alguns segundos para todos os 15 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 proporcionalmente em relação ao número de enlaces da rede.

A Tabela 4 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 17 problemas.

Problema	Caminhos	Função Objetivo	Tempo Execução
1	547	7191	00s
2	872	9531	00s
3	186	12615	00s
4	265	15309	00s
5	1400	5679	00s
6	2050	7521	00s
7	3250	31668	02s
8	2010	20310	01s
9	4217	38364	04s
10	2003	32775	01s
11	4507	102999	08s
12	1971	25044	01s
13	4525	Infactível	06s
14	4850	Infactível	07s
15	4801	Infactível	07s

Tabela 5: Limitando em 50 caminhos por *commodity*.

Foram incluídos dois problemas, além dos 15 anteriores nessa abordagem, ambos com número relativamente grande de enlaces. 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 resolve-se o problema da 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, temos os problemas 16 e 17 que demorariam horas ou dias para serem resolvidos exatamente, 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.

7 Conclusão

A tecnologia de *switchlet* representa um grande potencial na efetivação de redes programáveis. Apesar desta tecnologia ter sido desenvolvida para a tecnologia ATM, o conceito de *switchlets* é extensível a outras tecnologias de multiplexadores (comutadores).

Mecanismos automáticos para alocação de recursos para implementação de VPNs são de capital importância para implementação eficiente de redes virtuais.

Este trabalho introduziu um método baseado em *multicommodity flow* para alocação de recursos em redes baseados em *switchlets*. Mostrou-se que o método é viável para implementação em tempo real.

Como trabalho futuro sugere-se o uso do algoritmo modificado de Dijkstra para encontrar os K caminhos de menor custo e usar esses caminhos na otimização.

Neste artigo, escolheu-se apenas os K primeiros caminhos encontrados pela busca em profundidade. Além disso, sugere-se estender o método introduzido neste artigo para integrar tráfego com Qualidade de Serviço(QoS) junto com o serviço do melhor esforço(*Best-Efford*)[5].

Agradecimentos

Este trabalho foi parcialmente financiado pelo CNPq e pelo PRONEX. Os autores agradecem os comentários dos revisores anônimos.

Referências

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Networking Flow - Theory, Algorithms and Application*. Prentice Hall, 1993.

[2] A. T. Campbell, H. G. De Meer, M. E. Kounavis, J. Vicente K. Miki, and D. A. Villela. A Survey of Programmable Networks. *Computer Communication Review (CCR)*, 29(2):7–23, Abril 1999.

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

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

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

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

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

[8] Michael Sig and Mikkel Sigurd. MultiCommodity Flow with Integer Solutions. Technical report, Department of Computer Science, University of Copenhagen, May 1999.

[9] J. E. van der Merwe and I. M. Leslie. Switchlets and Dynamic Virtual ATM Networks. *Proc Integrated Network Management V*, pages 355–368, Maio 1997.

[10] J. E. van der Merwe and I. M. Leslie. Service-Specific Control Architectures for ATM. *IEEE Journal*, 16(3):424–436, Abril 1998.