CAMINHOS VIRTUAIS: APLICAÇÕES DEPENDENTES DA INTERNET TOLERANTES A FALHAS NAS ROTAS DE REDE

Jaime Cohen

jaime@convoy.com.br

Departamento de Informática - Universidade Estadual de Ponta Grossa

R. C. Cavalcanti, 4748, CEP 84031-900 Ponta Grossa, PR, Brasil

Fone: (0**42) 225-7544

RESUMO

Quando duas ou mais aplicações estão se comunicando através da Internet, pode ocorrer uma falha na comunicação entre elas mesmo que as aplicações estejam operacionais. O problema pode se encontrar em alguma conexão que faz parte da rota de rede que liga os dois nodos . Esta situação só irá alterar-se quando as tabelas de roteamento forem modificadas após um tempo considerável. O interessante disso é que apesar de ocorrerem falhas na rota de rede, podem existir ainda muitos caminhos físicos entre os dois nodos, disponíveis para transmitir a informação. Uma solução para minimizar o tempo de recuperação das conexões falhas foi proposta em DUARTE JR (1998), consistindo em desviar a rota por nodos da rede. O presente trabalho apresenta um novo critério para posicionar os desvios nas rotas e um algoritmo original para implementar este método. Além disso, apresentamos resultados experimentais mostrando fortes indícios de que o algoritmo pode reduzir os tempos de resposta das aplicações rodando em redes com presença de falhas.

Palavras-chave: Computação Tolerante a Falhas, Roteamento em Redes, Teoria dos Grafos, Conectividade.

ABSTRACT

Network applications communicating over the Internet may be temporary unable to exchange data due to a fault in a link somewhere on the IP route connecting the pair of nodes. The lack of communication will last until the network recovers from the failure, probably by changing the routing tables. An interesting fact related to this situation is that the Internet have enough redundant paths available to transmit the information. Based on these ideas, a solution to minimize the response time of the applications was proposed in DUARTE JR (1998) and it is based on relaying the fail route through a chosen node in the network. In the present work we introduce new criteria for placing the mencioned relaying nodes and an original algorithm to implement the method. Further, we present experimental results showing strong evidence that the algorithm can improve the response time of network applications running in the presence of network faults.

1 INTRODUÇÃO

As redes de computadores têm se tornado imprescindíveis para organizações e indivíduos. Falhas e problemas de desempenho causam prejuízos cada vez maiores. Assim sendo, soluções tolerantes a falhas para problemas em redes de computadores são de extrema

importância. Quem nunca tentou acessar uma página WEB de um site operacional e só conseguiu fazê-lo minutos ou horas depois? O problema poderia estar na rota de comunicação.

Considere um par de entidades no nível de aplicação, um cliente e um servidor, comunicando-se através da Internet. Se houver uma falha em algum ponto das rotas de rede (ou rotas IP – *Internet Protocol*) sendo usadas pelas entidades, a comunicação entre elas vai falhar mesmo que as entidades estejam sem falhas e mesmo que existam rotas físicas alternativas. O problema persistirá até que o protocolo de roteamento da camada de redes resolva o problema, determinando uma nova rota. Na situação atual, mesmo havendo caminhos físicos alternativos, estes não são usados, pois as aplicações não têm controle sobre as rotas que são determinadas pelo nível de rede. Além disso, o roteamento na Internet tem apresentado uma série de problemas e instabilidades (LABOVITZ,1998) (LABOVITZ, 1999) (PAXON, 1996).

Para tornar as aplicações de redes tolerantes a falhas, é necessário que haja novas formas para que clientes acessem os servidores e vice-versa. Assim, rotas alternativas para a comunicação entre aplicações devem usar entidades no nível de aplicação para desviar as rotas originais, permitindo que os PDU's (*Protocol Data Units*) cheguem aos seus destinos.

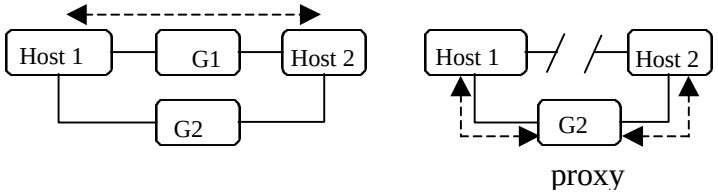
Usando o conceito das *proxies de roteamento* apresentado em DUARTE JR (1998), a aplicação tem um mecanismo simples que permite a implementação de um sistema de roteamento tolerante a falhas.

2 PROXIES DE ROTEAMENTO E ROTAS DE APLICAÇÃO

Uma solução para as aplicações serem tolerantes a falhas nas rotas de rede é permitir que a própria aplicação tente uma nova rota quando aquela não estiver funcionando. Uma *rota de aplicação* é definida como uma rota determinada por um protocolo de aplicação, sendo constituída por uma concatenação de rotas de rede (DUARTE JR., 1994). Uma *proxy de roteamento* é uma entidade que funciona como uma ponte entre duas entidades de aplicação. A rota a partir do cliente até a *proxy* e então ao servidor é uma rota de aplicação. Como as rotas de rede não são transitivas, a rota de rede entre A e B pode ser muito diferente da rota de aplicação entre A e B tendo um nodo C como *proxy de roteamento*. Este fato é crucial para o funcionamento do método. Veja a figura 1.

Um algoritmo para selecionar *proxies* num *backbone* foi apresentado em cite DUARTE JR (1998). No restante do trabalho, descreveremos um algoritmo original para este problema e mostramos experimentalmente a sua eficácia.

PAXON, V. “End-to-End Routing Behavior in theInternet”, **Proc. of the ACM SIGCOMM**, 1996.



A) Operação Normal B) Recuperação de Falha

Figura 1. Rotas de Aplicação.

3 CRITÉRIOS PARA O POSICIONAMENTO DE PROXIES DE ROTEAMENTO

Para que um algoritmo de posicionamento de *proxies* de roteamento possa ser desenvolvido e justificado, critérios devem ser estabelecidos para determinar onde as *proxies* deverão ser posicionadas. A nossa proposta foi a de elaborar um conjunto de critérios simples, cada um deles com claras explicações da sua relação com o problema e também razões do porquê uma implementação destes critérios deve levar a soluções eficientes.

Estes critérios consideram apenas a topologia física da rede, sem precisar avaliar as rotas de rede que são mais complexas e que mudam com maior freqüência. O algoritmo se baseia em parâmetros de conectividade da rede e também nos comprimentos das rotas de aplicação. As redes podem ser modeladas com custos nas arestas com o objetivo de representar a confiabilidade ou a banda das conexões.

3.1 Critérios de Conectividade e Distância

O principal critério para a ordenação das *proxies de roteamento* é baseado em conceitos de conectividade da Teoria dos Grafos. Os nodos da rede que pertencem a subgrafos com alta aresta-conectividade terão precedência para se tornarem proxies. Desta forma, aumentamos a chance de que o novo caminho de aplicação seja disjunto do caminho IP que se encontra falho. Outro aspecto para a escolha deste critério é que a *proxy* escolhida terá uma maior chance de ser reutilizada por outros pares de nodos que necessitem de uma rota alternativa.

Para evitar o quanto possível o uso de *proxies* muito distantes dos nodos que estão se comunicando, limites nos comprimentos dos caminhos de aplicação são impostos.

Um resumo dos critérios para escolha das *proxies de roteamento* segue abaixo:

- A proxy de roteamento deve pertencer a um componente com grande aresta-conectividade.
- O comprimento do caminho de aplicação não deve exceder um certo limite, que será especificado pelo algoritmo descrito adiante.

Além das idéias acima, duas regras são utilizadas para a resolução de empates:

- O tamanho do componente com alta conectividade contendo a proxy candidata deve ser maximizado. Desta forma, a proxy terá maiores chances de ser reutilizada no futuro.
- O comprimento do caminho de aplicação deve ser minimizado.

Para um exemplo simplificado de uma boa *proxy de roteamento* que segue os critérios descritos acima, veja a figura 2 a seguir.

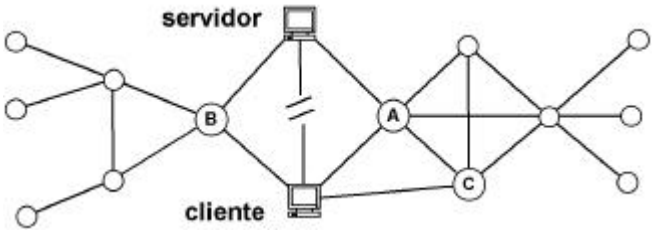


Figura 2. O link entre o servidor e o cliente está falho. O nodo A é o primeiro candidato para proxy de roteamento. Ele está no subgrafo mais conexo da rede e forma a rota de aplicação mais curta.

4 Definindo os Conceitos de Conectividade e Distância

Nesta seção serão definidos os conceitos sobre conectividade e distância em grafos que serão utilizados nas descrições dos algoritmos.

Comprimento da Rota de Aplicação. O comprimento de uma rota de aplicação formada por uma *proxy de roteamento* é definido abaixo:

Definição: Sejam *cliente*, *servidor* e *proxy* nodos da rede. Definimos *dist(proxy,cliente,servidor)* como sendo o comprimento da concatenação das rotas de rede entre o *cliente* e a *proxy* e entre a *proxy* e o *servidor*.

Vizinhança. O conceito de vizinhança definirá aqueles nodos que quando usados como *proxies de roteamento* irão formar caminhos de aplicação que não sejam longos. A definição segue:

*Definição: P*ara um inteiro *d* e nodos *cliente* e *servidor* da rede, definimos *d-vizinhança(cliente, servidor)* como sendo o conjunto de nodos *p* cujas rotas de aplicação entre *cliente* e *servidor* usando *p* como *proxy* possuem comprimento menor do que ou igual à *d*.

Número de Conectividade de um Vértice e Componentes com Máxima Conectividade. Os parâmetros de conectividade usados pelo algoritmo são definidos abaixo:

Definição: Seja *#C(v)* (com respeito a um grafo G) a aresta-conectividadede de um subgrafo não trivial de G contendo *v* e tal que a cardinalidade de qualquer corte separando nodos deste subgrafo seja maximizada. Também definimos *MCC(v)* como sendo o maior subgrafo

contendo *v* e tal que qualquer par de nodos deste subgrafo não pode ser separado por um corte de tamanho menor do que *#C(v)*. A figura 3 ilustra ambos os conceitos.

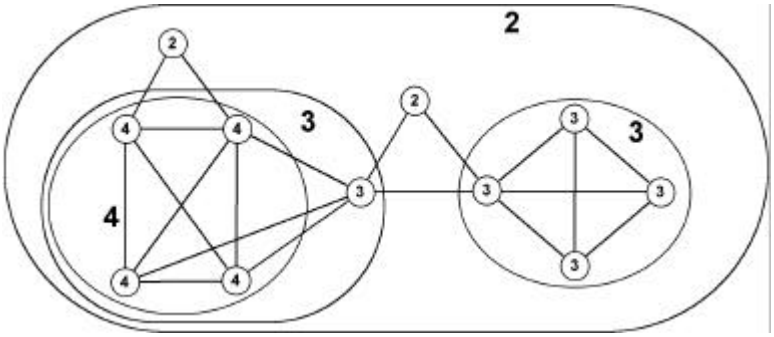


Fig. 3. Esta figura mostra o número de conectividade dos vértices, isto é, *#C(v)*, e os componentes *MCC(v)*.

Nas definições de *#C()* e *MCC()*, note que por subgrafo não trivial queremos dizer um subgrafo contendo dois ou mais vértices. Caso subgrafos triviais fossem permitidos na definição, o grau do vértice seria trivialmente o seu número de conectividade. Porém, o grau de um nodo não é um bom parâmetro para o problema abordado neste artigo. A razão é que podem haver muitos caminhos passando por um mesmo nodo e isso pode acontecer quando um vértice pertence a um subgrafo de baixa conectividade. Considere por exemplo um subgrafo induzido possuindo uma topologia de estrela. Neste caso, muitas rotas IP passarão pelo centro desta estrela e uma rota de aplicação poderá necessitar evitar passar por ele. Note também que o grau de um vértice *v* é uma cota superior para *#C(v)*.

5 Algoritmos para Localização de Proxies de Roteamento

O algoritmo principal que descreveremos encontra uma boa *proxy de roteamento* dentro de alguma vizinhança envolvendo o par de nodos tentando se comunicar. O critério de conectividade é usado e desempates são tratados pelo tamanho do subgrafo contendo o nodo e também pelo comprimento do caminho de aplicação.

A parte mais difícil do algoritmo é o calculo dos números de conectividade, *#C(v)*, e os respectivos *MCC(v)*. Nós propomos dois algoritmos com esta finalidade, que serão descritos em uma seção separada. O pseudocódigo do algoritmo principal é apresentado na figura 4 adiante.

Complexidade de Tempo. O tempo de execução do algoritmo da figura 4 é dado pelo tempo para o cálculo das funções *#C()* e *MCC()* e as distâncias entre os nodos. As distâncias podem ser calculadas através de duas execuções de um algoritmo para determinação de caminhos mínimos a partir de um vértice. A primeira execução é feita a partir do cliente e a outra

a partir do servidor. Utilizando-se o algoritmo de Dijkstra, o tempo será de $\Theta(n.log\ (n))$, assumindo que a rede é esparsa.

```
Algoritmo: Seleciona_Proxies (Grafo G = (V, E) ,  Nodos cliente, servidor)
Saída: lista ordenada de nodos candidatos a proxies de roteamento.
início
    Seja L uma lista vazia;
    Calcule os valores #C(v), MCC(v) e  dist(v, cliente, servidor)  para cada nodo v;
    Seja P o comprimento do caminho de rede usado entre o cliente e o servidor;
    Seja d = 2;
    enquanto   |L| <= |V| - 2   faça
        Ordene os nodos em V-L da vizinhança d*|P|-vizinhança(cliente, servidor)
            ...   em ordem não crescente e usando como chave
            ...   < #C(v),  | MCC(v) |,  – dist (v, cliente, servidor) >;
         Concatene os nodos no final da lista L;
        d := d + 1;    //  a vizinhança é aumentada para alcançar novos nodos
    fim_enquanto;
    retorne L;
fim.
```

Figura 4. Algoritmo principal para seleção de proxies de roteamento.

O tempo de execução do algoritmo como um todo dependerá do algoritmo usado para o cálculo das funções *#C()* e *MCC()*. Descreveremos a seguir duas possibilidades: a primeira calcula as duas funções de forma exata e tem um tempo de execução equivalente ao tempo da construção de uma Árvore de Cortes, também conhecida como Árvore de Gomory-Hu. A construção da Árvore de Cortes requer a execução de n-1 algoritmos de fluxo máximo em redes entre dois vértices. O tempo total de execução do algoritmo acima será então limitado por $O(n^4)$, considerando os algoritmos práticos para o problema de fluxos em redes. (COOK,1998), (LEEUWEN, 1990).

Com o intuito de evitar o tempo necessário para a construção da Árvore de Cortes, uma heurística é proposta para computar a função *#C(v)* de forma aproximada. Esta heurística não é capaz de computar a função *MCC()*, porém isto não é um problema, já que o algoritmo principal utiliza dois critérios de desempate na escolha das *proxies*. A grande vantagem desta heurística é que o seu tempo de execução é linear, considerando redes esparsas. A complexidade total do algoritmo diminui assim para $\Theta(n.log(n)+m)$.

5.1 O Algoritmo Exato para Computar *#C()* e *MCC()*

O seguinte lema é a chave para o cálculo de *#C():*

Lema 1. Dado um grafo G = (V,E) e um vértice $v \in V$, o valor de *#C(v)* é igual à máxima cardinalidade (ou capacidade) dentre todos os cortes mínimos separando *v* de cada outro vértice do grafo.

Prova. Considere um subgrafo H de G com a maior conectividade que define o valor de *#C(v).* Como H não é trivial, então necessariamente existe um vértice *u* de H, $u \neq v$. O corte mínimo separando *v* de *u* deve ser o de maior valor dentre os cortes mínimos separando *v* de outros vértices. Portanto, para computar *#C(v)* basta calcularmos os valores de todos os cortes mínimos entre *v* e os demais vértices.

O lema 1 nos diz que o problema de calcular os valores de *#C(v)*, para todo v em V, pode ser reduzido ao problema de encontrar o corte mínimo entre todos os pares de vértices do grafo. Este problema já foi largamente estudado, mas pouco progresso foi feito desde o trabalho de R.E. Gomory e T.C. Hu em 1961. Veja GOMORY (1961), HU (1982) e COOK (1998) para uma descrição deste método e também GUSFIELD (1990) para novas idéias sobre o problema.

Uma *Árvore de Cortes T* de um grafo *G* é definida como uma árvore ponderada tal que: (1) Os nodos de G correspondem aos nodos da árvore T. (2) O tamanho de um corte mínimo entre dois vértices quaisquer de G é igual ao peso da aresta de peso mínimo pertencente ao único caminho entre os vértices em T. (3) Um corte mínimo entre dois vértices de G é dado removendo-se a aresta de T citada no item acima e considerando os dois conjuntos de vértices induzidos pelas duas árvores formadas pela remoção da aresta.

A Árvore de Cortes permite uma rápida resposta à pergunta sobre qual é o custo de um corte mínimo entre qualquer par de vértices de *G*. Além disso, ela permite que se determine um corte mínimo de forma bastante simples. Para maiores detalhes veja HU (1982) e COOK (1998).

Lema 2. O valor *#C(v)* é igual ao maior valor de uma aresta adjacente à *v* em uma árvore de cortes T_c do grafo G.

Prova. Por definição, o valor de um corte mínimo do vértice *v* até um outro vértice *u* corresponde ao menor valor de uma aresta no caminho em T_c de *u* até *v.* Ainda, este valor deve ser menor do que ou igual ao valor da aresta adjacente à *v* do caminho. Portanto, a máxima capacidade de um corte mínimo separando *v* de qualquer outro vértice de *G*, deve ser igual ao valor de uma aresta adjacente a *v* em T_c com o maior valor.

Desta forma, podemos computar o valor de #C(v). O valor de *MCC(v)* pode ser computado analisando sub-árvores da árvore de cortes, podendo ser feito em tempo linear. Detalhes podem ser encontrados em COHEN (2001).

5.2 Uma Heurística de Tempo Linear para Calcular #C(v)

A heurística que propomos se baseia em uma idéia sugerida em um algoritmo descrito em KHULLER (1996) para um outro problema de conectividade. A heurística descrita aqui procura retirar, a cada passo, um pequeno número de aresta que formam um conjunto de subgrafos dois aresta-conexos. Cada componente conexo encontrado será formado por componentes 2-aresta-conexos ligados por caminhos simples. Os vértices dos componentes 2-aresta-conexos têm seus valores *#C(v)* acrescidos de 2 e os vértices de caminhos têm um acréscimo de 1 em seus *#C(v)*. Para fazer isso, a heurística remove, de cada componente conexo restante uma árvore de busca em profundidade e mais um conjunto de arestas para tentar formar subgrafos dois conexos. As arestas acrescentadas à árvore são chamadas de *arestas de cobertura*. As arestas de cobertura são encontradas usando uma idéia heurística que consiste em escolher a aresta que possui umas das pontas o mais alto possível na árvore de busca, tentando "cobrir" o máximo possível das arestas. O pseudocódigo segue abaixo:

Algoritmo: Heurística para o Cálculo de *#C(v)* de um Grafo G = (V, E)
início
  para cada v em G **faça**
    #C(v) = 0;
    se grau(v) = 0 **então** remova v de G;
  fim_para;
  enquanto |E| ≠ ∅ **faça**
    para cada componente conexo *c* de *G* **faça**
      T = uma árvore de busca em profundidade de *c;*
      para cada aresta *e* de T em pós-ordem **faça**
        se *e* não está coberta **então** encontre uma aresta de cobertura para *e*;
        para cada nodo *v* de *c* **faça**
          se *v* pertence a um ciclo de T + "arestas de cobertura"
          então *#C(v)* = *#C(v)* + 2
          senão *#C(v)* = *#C(v)* + 1;
        fim_para;
        remova de G as arestas de T e as arestas de cobertura;
        remova de G os vértices isolados;
    fim_para;
  fim_enquanto;
fim.

Esta heurística roda em tempo de pior caso *O(m/n(m+n))* ou *O(n) em grafos esparsos*. Detalhes podem ser encontrados em COHEN (2001).

5.3 Alguns Resultados Experimentais

Dois métodos foram apresentados para o cômputo da função *#C()* para os nodos de uma rede: um deles calcula a função de forma exata porém a heurística é muita mais rápida. É óbvio perguntarmos quão próximo dos valores ótimos a heurística calcula *#C()*. Abaixo apresentamos resultados mostrando que o erro cometido pela heurística no cálculo de *#C()* é menor do que 1 em média para cada nodo.

Para rodarmos os experimentos, quatro classes de grafos foram geradas, com 50, 100, 200 e 500 nodos, respectivamente. Cada classe contém 50 grafos e todos os resultados são médias sobre estes 50 grafos. O método utilizado para a geração dos grafos foi o Método de Waxman, descrito em ZEGURA (1997), um artigo dedicado ao estudo de modelos para a Internet.

A tabela 1 mostra os resultados comparativos entre os dois métodos. Uma coluna mostra a soma das diferenças sobre todos os nodos e a outra mostra a diferença média.

N	soma das diferenças	erro médio por
50	39,2	0,78
100	64,2	0,64
200	155,0	0,77
500	428,0	0,86

Tabela 1. Diferença empírica entre a heurística e o algoritmo exato.

A tabela 2 mostra que as *proxies de roteamento* têm grande probabilidade de recuperar a comunicação no caso de erros aleatórios nas redes. Para este experimento novamente os grafos foram gerados pelo modelo de Waxman para terem topologias relacionadas com *backbones* da Internet. Se pensarmos na diferença de tempo que existiria com a utilização de uma proxy de roteamento com o de vários *time-outs* que ocorrem quando a aplicação tenta fazer repetidas conexões com um nodo não alcançável, podemos notar que os resultados mostrados abaixo são bastante promissores.

A tabela mostra a porcentagem de pares de nodos que recuperam a comunicação após a falha de um *link* na rota IP entre eles. As médias foram tiradas considerando todos os pares de nodos e a falha de todos os possíveis *links*, um por vez. A segunda coluna se refere ao experimento em que uma segunda *proxy* foi escolhida quando a primeira tentativa falhou.

n	uma proxy	duas proxies
10	73 %	93 %
20	75 %	93%
30	62 %	87 %
50	59 %	81 %

Tabela 2. Porcentagem de pares de nodos que recuperam a comunicação utilizando o método proposto.

6 REFERÊNCIAS BIBLIOGRÁFICAS

COOK, W.J.; CUNNINGHAMA, W.H.; PULLEYBLANK, W.R. e SCHRIJVER, A. **"Combinatorial Optimization"**. John Wiley & Sons, Inc., 1998.

COHEN, J. **"Relatório de Projeto: Algoritmos para o problema da conectividade em grafos com aplicações ao gerenciamento integrado de redes de computadores"**. Relatório da Universidade Estadual de Ponta Grossa, disponível em www.deinfo.uepg.br/jaime, 2001.

DUARTE JR, E.P.; MANSFIELD, G.; NANYA, T. e NOGUCHI, S. "Improving the Dependability of Network Management Systems". **International Journal of Network Management**, vol. 8, núm. 4, 1998.

DUARTE JR, E.P.; MANSFIELD, G.; NOGUCHI, S. e MIYAZAKI, M. "Fault-Tolerant Network Management". **Proc. of ISACC'94**, Mexico, 1994.

GOMORY, R.E. e HU, T. C. "Multi-terminal network flows". **SIAM Journal on Applied Mathematics**, vol. 9, 1961.

GUSFIELD, D. "Very simple methods for all pairs network flow analysis". **SIAM Journal of Computing**, vol. 19, núm. 1, 1990.

HU, T.C. **"Combinatorial Algorithms"**, Addison-Wesley, 1982.

KHULLER, S. e RAGHAVACHARI, B. "Improved approximation algorithms for uniform connectivity problems". **Journal of Algorithms**, 21, 1996.

LABOVITZ, C.; AHUJA, A. e JAHANIAN, F. “Experimental Study of Internet Stability and Backbone Failures”. **Proceedings of the FTCS-29**, Madison, Wiscosin, 1998.

LABOVITZ, C.; MALAN, G. R. e JAHANIAN, F. “Internet Routing Instability”. **IEEE Transactions on Networking**, vol. 7, núm. 3, 1999.

PAXON, V. “End-to-End Routing Behavior in theInternet”, **Proc. of the ACM SIGCOMM**, 1996.

LEEUWEN, Editor J. van. “Handbook of Theoretical Computer Science: Algorithms and Complexity – Volume A”. **Elsevier e the MIT Press**, 1990.

ZEGURA, E. W.; CALVERT, K.L. e DONAHOO, M. J. “A quantitative comparison of graph-based models for Internet topology”, **IEEE/ACM Transactions on Networking**, 1997.