# Um algoritmo ótimo linear para o problema de escalonamento em lote em redes OBS

**Gustavo B. Figueiredo**[1], **Nelson L. S. da Fonseca**[2]

[1]Grupo de Redes Ópticas e Wireless (GROW)
Núcleo de Pesquisa em Redes de Computadores – NUPERC
Universidade Salvador (UNIFACS)
Rua Ponciando de Oliveira, 126 - Rio Vermelho, 41950-275- Salvador- Bahia- Brasil.

[2]Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6176 – 13.084 -971 – Campinas – SP – Brasil.

{gustavo,nfonseca}@ic.unicamp.br

***Resumo.*** *No problema de escalonamento em lote em redes OBS, existe um compromisso entre a qualidade da solução em termos de probabilidade de bloqueio e o tempo de execução necessário para a obtenção das soluções. Este artigo apresenta um algoritmo ótimo com complexidade computacional linear para o problema de escalonamento em lote em redes OBS. Resultados obtidos via simulação mostram o bom desempenho do algoritmo quando comparado a outros apresentados na literatura.*

***Abstract.*** *In the channel batch scheduling problem for OBS networks, there is a trade-off between the quality of the solution in terms of blocking probability and the execution time required for obtaining such solutions. This paper presents an optimal algorithm with linear computational complexity to the problem of batch scheduling in OBS networks. Results obtained via simulations show the good performance of the algorithm when compared to other in the literature.*

## 1. Introdução

Em relação ao número de requisições processadas por vez, os algoritmos de escalonamento propostos para as redes OBS podem ser classificados em dois grupos: algoritmos de escalonamento gulosos [Yu et al. 2004, Xiong et al. 2000, Chen et al. 2004] e algoritmos de escalonamento em lote [Kaheel and Alnuweiri 2005, Charcranoon et al. 2003, Figueiredo et al. 2009]. Os algoritmos da primeira classe são ditos gulosos pois cada requisição é processada individualmente, usando somente as informações disponíveis no momento do processamento. De um modo geral, tais algoritmos possuem baixa complexidade computacional e armazenam um número pequeno de informações de estado.

Os algoritmos de escalonamento em lote, por sua vez, realizam o processamento de um conjunto de requisições ao invés de requisições individuais. A idéia é que ao processar mais do que uma requisição por vez, o algoritmo tenha informações suficientes para evitar que rajadas sejam descartadas pela adoção de uma estratégia gulosa equivocada.

Diferentes formas de modelar o problema de escalonamento em lote em redes OBS foram propostas. Isto implica em diferentes soluções, com níveis distintos de qualidade e complexidade computacional. Em [Kaheel and Alnuweiri 2005, Charcranoon et al.

2003] o problema de escalonamento em lote foi reduzido ao problema de *job scheduling* com máquinas não idênticas. O custo computacional de uma solução ótima nesta modelagem é proibitivamente alto, com complexidade exponencial no número de canais. Dessa forma, foram propostas algumas heurísticas cujo tempo de execução possui complexidade linear. Já em [Figueiredo et al. 2009] foi proposta uma redução ao problema de *job scheduling* com máquinas idênticas, o que permitiu a obtenção de uma solução ótima em tempo polinomial para o problema de escalonamento em lote onde as requisições possuem pesos que refletem suas prioridades de escalonamento.

É possível, assim, perceber um compromisso entre a qualidade da solução fornecida pelo algoritmo de escalonamento e o tempo de execução gasto para a obtenção das mesmas. Os algoritmos propostos em [Kaheel and Alnuweiri 2005, Charcranoon et al. 2003] possuem complexidade linear mas oferecem soluções heurísticas. E o algoritmo proposto em [Figueiredo et al. 2009] fornece solução ótima mas possui complexidade polinomial.

Um cenário ideal seria composto por um algoritmo com solução ótima e complexidade computacional linear. Neste artigo é apresentado como o problema de escalonamento em lote em redes OBS pode ser resolvido em tempo linear no número de requisições em processamento. Para tal, é apresentado o algoritmo GreedyOPT. O algoritmo é proposto para o cenário onde as requisições trafegando na rede possuem pesos idênticos e é obtido através da combinação de uma mudança na modelagem do problema e do emprego de um algoritmo ótimo para o problema de *job scheduling* proposto por [Bouzina and Emmons 1996]. É ainda realizada uma avaliação de desempenho do algoritmo onde se quantifica, além do desempenho dos algoritmos em relação à probabilidade de bloqueio, o impacto da mudança da modelagem do problema e o tempo de execução dos algoritmos.

O resto do artigo está organizado como segue: A Seção 2 apresenta conceitos e definições necessárias ao entendimento dos algoritmo apresentados. A Seção 3 apresenta os trabalhos relacionados. A Seção 4 apresenta o algoritmo GreedyOPT propriamente dito. A Seção 5 mostra os exemplos numéricos e por fim a Seção 6 apresenta as conclusões.

## 2. Algumas Definições

Denota-se por $G = (V, E)$ um grafo, onde $V(G)$ é o conjunto de vértices de $G$ e $E(G)$ seu conjunto de arestas. Seja $u \in V(G)$ um vértice de $G$, a *adjacência* de $u$ é definida como: $Adj(u) = \{v \in V(G); (u, v) \in E(G)\}$. Um *subgrafo* $H$ de $G$ é um grafo com $V(H) \subset V(G)$ e $E(H) \subset E(G)$. O *grau de* $u$ no subgrafo $H$, denotado por $d(u|H)$ corresponde ao número de vértices adjacentes a $u$ no grafo $H$. Dado um conjunto $V(H) \subseteq V(G)$, o subgrafo de $G$ induzido por $V(H)$ é o grafo $H = (V(H), E(H))$, onde $E(H) = \{(u, v) \in E(G); u, v \in V(H)\}$.

Uma *clique* é um conjunto $C \subset V(G)$ tal que $\forall u, v \in C; (u, v) \in E(G)$. Uma clique $C$ é dita *maximal* se não existir outra clique em $G$ que tenha $C$ como subconjunto.

Um vértice $v$ de $G$ é simplicial se sua vizinhança (conjunto de vértices adjacentes a $v$) induz uma clique. Um esquema de eliminação perfeito é uma ordenação $[v_n, ..., v_1]$ dos vértices de $G$ tal que cada vértice $v_i$ (com $1 \leq i \leq n-1$) é um vértice simplicial no subgrafo induzido por $V(G) \setminus v_{i+1}$

$G$ é um grafo de intervalos se existir uma correspondência biunívoca entre um conjunto de intervalos $\{I_v\}$, na reta real, e o conjunto de vértices, tal que para dois vértices distintos $u$ e $v$, tem-se que $(u, v) \in E(G) \Leftrightarrow I_v \cap I_u \neq \emptyset$. Os grafos de intervalos apresentam particularidades que os tornam atraentes para a solução de diversos problemas em combinatória. Dentre tais particularidades estão o seu reconhecimento e coloração em tempo linear, o que faz com que os algoritmos baseados em grafos com tais estruturas sejam extremamente rápidos.

Um exemplo típico da aplicação dos grafos de intervalos é a sua utilização na solução do problema de *job scheduling*, descrito a seguir: Seja $I = \{J_1 = (s_1, e_1, p_1), \ldots, J_n = (s_n, e_n, p_n)\}$ uma lista de $n$ tarefas, onde $(s_i, e_i)$ é o tempo de início e fim da tarefa $J_i$, e $p_i$ o seu peso. Tem-se um conjunto $W$ de máquinas com mesma capacidade de processamento. Todas as máquinas estão inicialmente livres desde o tempo 0 até mais infinito. O objetivo do problema é selecionar uma sub-lista $I' \subseteq I$ de tarefas de peso máximo, e alocar $I'$ nas $|W|$ máquinas. O escalonamento gerado deve satisfazer o fato de que em cada máquina não poder existir sobreposição de tempo entre as tarefas.

Caso não seja imposta nenhuma restrição sobre quais tarefas possam ser processadas em cada máquina, tem-se um problema de *job scheduling com máquinas idênticas*. Caso haja restrições de que algumas tarefas não possam ser processados em um determinado conjunto de máquinas, tem-se um problema de *job scheduling com máquinas não idênticas*.

O problema de escalonamento de canais em redes OBS pode ser reduzido ao problema de *job scheduling*, onde os canais e as requisições das redes OBS correspondem, respectivamente, às máquinas e tarefas no problema de *job scheduling*. No problema de escalonamento de lotes em redes OBS tem-se, formalmente, um conjunto $W$ de canais, e uma lista $I = \{J_1 = (s_1, e_1, p_1), \ldots, J_n = (s_n, e_n, p_n)\}$ de $n$ rajadas (correspondente a um lote), onde $(s_i, e_i)$ é o tempo de início e fim da rajada $J_i$ e $p_i$ seu peso correspondendo a sua prioridade de escalonamento. Tem-se também uma lista $S$ de rajadas já previamente alocadas nos canais. A lista $S$ corresponde a rajadas que foram processadas em lotes anteriores. *Objetiva-se, neste problema, alocar o conjunto de rajadas cuja soma dos pesos seja máxima (ou o maior número possível de rajadas caso todas possuam pesos idênticos) nos $|W|$ canais*. Duas rajadas em um mesmo canal não podem se sobrepor, ou seja, seus tempos de execução não podem ter uma intersecção.

Na Seção 3, apresenta-se uma modelagem do problema de escalonamento em redes OBS, através da formulação de um problema *job scheduling com máquinas não idênticas*, que é NP-Difícil [Kaheel and Alnuweiri 2005]. Na seção 4, mostra-se como o problema pode ser resolvido em tempo linear através de uma redução ao problema de *job scheduling com máquinas idênticas*.

## 3. Trabalhos relacionados

Em [Kaheel and Alnuweiri 2005], o problema de escalonamento de canais é resolvido através de uma formulação do problema de *job scheduling* com máquinas não idênticas. Neste caso, o conjunto $S$ de rajadas previamente alocadas representam intervalos de tempo nos quais algumas rajadas de $I$ não podem ser alocadas (ja que há intersecção com rajadas de $S$). Assim, Kaheel e Alnuweiri assumem uma modelagem direta do problema de escalonamento de canais para o problema de *job scheduling* em máquinas não

idênticas.

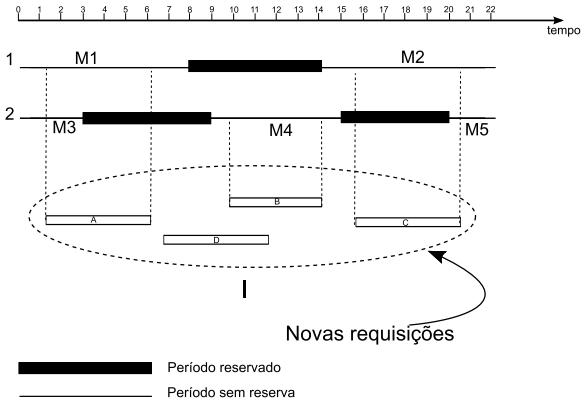


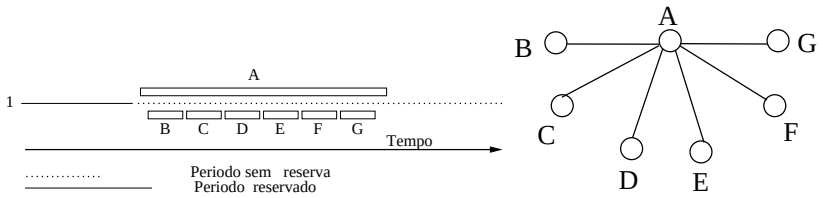

**Figura 1. Modelagem do problema de escalonamento em lote de canais como *job scheduling* com máquinas não idênticas.**

A Figura 1 ilustra tal modelagem. Na figura, existem dois canais de dados que podem ser utilizados para acomodar as requisições. É possível perceber que o canal 1 está reservado no intervalo de tempo compreendido entre os instantes 8 e 14 e os *voids* formados pelos intervalos $[0, 8)$ e $(14, \infty)$ podem ser usados para novas reservas. Dessa forma, os *voids* são modelados como máquinas distintas (M1 e M2) com períodos de funcionamento restrito. Da mesma forma, o canal 2 possui reservas nos intervalos $[3, 9]$ e $[15, 20]$. Assim, os períodos usados para acomodar novas reservas são: $[0, 3)$, $(9, 15)$ e $(20, \infty)$, o que na modelagem do problema de *job scheduling* seria equivalente a três máquinas distintas, M3, M4 e M5. Dessa forma, tem-se cinco máquinas com diferentes períodos de operação (diferentes capacidades).

Em [Arkin and Silverberg 1987], apresenta-se um algoritmo ótimo para *job scheduling* com máquinas não idênticas cuja complexidade computacional é da ordem de $O(n^{W+1})$, o que é proibitivamente alto em redes OBS [Kaheel and Alnuweiri 2005], dada a exponencialidade em $W$. Dessa forma, em [Kaheel and Alnuweiri 2005] são propostas 4 heurísticas para o problema de escalonamento em lote em redes OBS. A complexidade computacional dos algoritmos é de $O(nW \log(N))$, onde $n$ é o número de requisições sendo processadas, $W$ o número de canais e $N$ o número de reservas já realizadas. A seguir serão brevemente descritas as heurísticas propostas por [Kaheel and Alnuweiri 2005]. Em todas elas, as requisições são vistas como um grafo de intervalos $G$.

Os vértices $v_1, ..., v_n$ de $G$ são ditos ordenados segundo o critério *smallest-last* se $v_i$ tem o menor grau no subgrafo induzido pelos vértices $v_1, ..., v_i$ (sendo $v_n$ o vértice de $G$ com menor grau). No algoritmo **Smallest Vertex Ordering (SLV)**, os intervalos são alocados na ordem *smallest last*, ou seja, a requisição correspondente a $v_1$ é alocada ao primeiro comprimento de onda disponível ou descartada, depois $v_2$ e assim por diante até o processamento de $v_n$.

A idéia por trás do SLV é que tendo o grafo alguns poucos nós de grau elevado, o processamento prévio desses evitará a necessidade de se usar um número grande de comprimentos de onda. Contudo, ao contrário do que afirmam os autores de [Kaheel and Alnuweiri 2005], o processamento prévio das requisições com alto grau de intersecção pode ocasionar um número elevado de perdas.




(a) *Padrão de chegadas de requisições* (b) Grafo de intervalos correspondente

**Figura 2. Problema ocasionado no escalonamento em lote**

A Figura 2 ilustra o problema mencionado: suponha que as requisições se dispõem na forma apresentada na Figura 2(a). O grafo de intervalos correspondente é apresentado na Figura 2(b). Pode-se notar que o vértice "A" é o vértice com maior grau na ordenação *smallest last* (e portanto o primeiro a ser processado). Entretanto, ao se alocar o canal 1 à requisição A, todas as outras requisições serão descartadas.

Caso haja no grafo uma clique com tamanho $M$ e no máximo $k$ canais para alocação das requisições (com $M > k$), necessariamente $M - k$ requisições são descartadas. O algoritmo **Maximal Cliques First (MCF)** determina, além da ordem de processamento das requisições, quais requisições deverão ser descartadas caso necessário. Para isso, o algoritmo determina todas as cliques maximais de $G$ e as ordena, de forma crescente em relação ao tempo. Seja $\{C_1, C_2, ..., C_m, \}$ o conjunto de cliques maximais de $G$ ordenado tal que $C_i \prec C_j$ para $i < j$ (ou seja existe uma requisição em $C_i$ que possui tempo de início menor ou igual a cada uma das requisições em $C_j$). O algoritmo processa primeiro as requisições pertencentes à clique $C_1$ depois à $C_2$ e assim por diante. Se o tamanho de $C_j$ exceder o número de canais, requisições com o menor tempo de término são descartadas.

Assim como o algoritmo SLV, a estratégia adotada pelo algoritmo MCF pode não produzir os melhores resultados. Considere novamente a Figura 2, a primeira clique a ser processada é a clique formada pelos vértices "A" e "B". Como só existe um canal disponível, a requisição "B" é descartada e a requisição "A" é alocada no canal, o que faz com que todas as outras requisições sejam descartadas.

No algoritmo **Smallest Start-time First Ordering (SSF)**, as requisições são ordenadas de acordo com seu tempo de início. Assim, as requisições com menor tempo de início são processadas primeiro. A mesma situação de perdas apresentada na Figura 2 pode ocorrer no algoritmo SSF. A primeira requisição processada seria a requisição "A", o que resultaria nas perdas descritas.

No algoritmo **Largest Interval First Ordering (LIF)**, as requisições são ordenadas de acordo com o tamanho da rajada que consiste da diferença entre o instante de término e de início da requisição. As requisições que possuem maior tamanho são processadas primeiro. Assim como os algoritmos SLV, MCF e SSF, a situação descrita na Figura 2 pode ocasionar perdas se a requisição "A" possuir maior tamanho que as demais. É importante observar que mesmo que a soma dos tamanhos das demais requisições seja superior ao tamanho da requisição "A", elas não serão consideradas.

O problema com os algoritmos descritos é que eles são baseados em heurísticas que nem sempre produzem os melhores resultados. Assim, o desempenho dos algoritmos

depende da estrutura do grafo de intervalos associado ao lote de requisições, fazendo com que em alguns grafos os resultados sejam satisfatórios e em outros não. A seguir será discutido como esse problema pode ser resolvido de forma ótima com tempo de execução polinomial.

## 4. Algoritmo ótimo com complexidade linear no número de requisições processadas

Na seção 3, discutiu-se como o problema de escalonamento em lote de canais foi modelado como um problema de *job scheduling* com máquinas não idênticas [Kaheel and Alnuweiri 2005] e viu-se que tal modelagem leva a um algoritmo ótimo com complexidade exponencial no número de canais. Nesta seção é discutido como o problema pode ser resolvido em tempo linear.

A solução proposta para o problema de escalonamento em lote em redes OBS se dá em duas etapas. Na primeira etapa, é realizada uma redução ao problema de *job scheduling* com máquinas idênticas. Para garantir que todas as máquinas sejam idênticas, o conjunto $S$ de reservas já efetuadas cujas rajadas ainda não chegaram, é considerado na obtenção da nova solução, implicando no fato de que todos os canais estarão disponíveis para receber quaisquer requisições em qualquer instante de tempo. Para tal, é mantida uma estrutura de dados auxiliar na forma de um *heap* [Cormen et al. 1990] ordenado pelo tempo de término. Assim, o tempo necessário para mantê-lo é da ordem de $O(|S|\log(|S|))$.

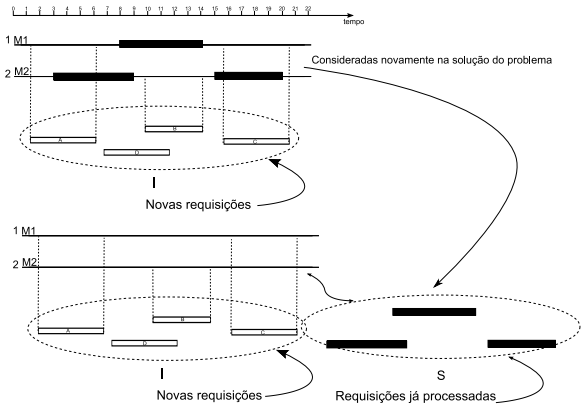



**Figura 3. Escalonamento em lote de canais com máquinas idênticas.**

Este processo é ilustrado na Figura 3, que mostra dois canais de dados usados para acomodar as requisições. Em um dado momento, o canal 1 está reservado no período $[8, 14]$ e o canal 2 nos períodos $[3, 9]$ e $[15, 20]$. Em seguida, chegam requisições de reservas A ($[1, 6]$), B ($[10, 14]$), C ($[16, 21]$) e D ($[7, 12]$). As requisições A, B, C e D são agrupadas em um lote, juntamente com as reservas previamente realizadas. Assim, ambos os canais podem ser usados para acomodar qualquer requisição, o que é determinado pela solução obtida pelo algoritmo de escalonamento.

O fato de não haver restrição sobre qual canal deve acomodar uma dada rajada, implica em uma redução direta ao problema de *job scheduling* com máquinas idênticas. Em última instância, tal redução permite que algoritmos rápidos e ótimos sejam usados na solução do problema, o que é feito na segunda parte da solução do problema.

Em [Bouzina and Emmons 1996] é apresentado um algoritmo para escalonamento de tarefas em um conjunto de $W$ máquinas idênticas cuja complexidade computacional é de $O(n \max(\log(n), |W|))$. Este algoritmo, denominado aqui GreedyOPT, será utilizado na segunda etapa da solução do problema de escalonamento de canais em redes OBS. A idéia fundamental do algoritmo baseia-se no Teorema 1.

**Teorema 1** *Dado um conjunto $I = \{J_1 = (s_1, e_1, 1), \ldots, J_n = (s_n, e_n, p)\}$ com $n$ requisições a serem escalonadas. Suponha que uma das requisições é encurtada, ou seja, seu tempo de início permanece inalterado, enquanto o tempo de término é antecipado. Se as demais requisições são inalteradas, então o número de requisições alocadas ou permanece inalterado ou é incrementado.*

**Proof** Ver [Figueiredo 2009]

A idéia do algoritmo é processar sequencialmente as requisições, tentando acomodá-las em um dos $|W|$ canais. Caso a requisição não possa ser acomodada, o algoritmo tenta substituí-la por alguma das requisições já acomodadas cujo tempo de término seja o maior. O algoritmo GreedyOPT é apresentado no Algoritmo 1.

**Algoritmo 1** GreedyOPT

**ENTRADA**

Um conjunto $W$ de canais de saída de um nó $i$, um conjunto $I$ de requisições a serem processadas em um lote e um conjunto $S$ de requisições já alocadas cujo período se intersecta com o período das requisições em $I$.

**SAÍDA**

Um conjunto $I'$ de cardinalidade máxima.

**GreedyOPT**

1: $N \leftarrow |I| + |S|$
2: Ordene todas as requisições de $I \cup S$ de acordo com o tempo de início de forma que $s_1 \leq s_2 \leq ... \leq s_N$.
3: Considere as requisições sequencialmente, adicionando-as ao conjunto $I'$
4: Caso não haja canal para acomodar a última requisição, remova de $I'$ a requisição com o maior tempo de término.

O algoritmo GreedyOPT resolve otimamente o problema de escalonamento em lote de canais em redes OBS quando todas as requisições sendo processadas na rede possuem peso unitário, como se pode constatar através do Teorema 2

**Teorema 2** *Seja $t_0$ o menor tempo em que mais que $|W|$ requisições se intersectam, seja $J$ o conjunto dessas requisições e $e_k = max_{j \in J} e_j$. Então, existe um escalonamento ótimo que não inclui $k$.*

**Proof** Ver [Bouzina and Emmons 1996]

#### 4.0.1. Complexidade computacional

**Teorema 3** *Seja $|W|$ o número de canais de saída de um nó OBS, $n$ o número de requisições a serem processadas em um lote, $S$ o conjunto de requisições já aloca-*

*das cujo período de tempo se sobrepõe às requisições do lote e* $s = |S|$*, a cardinalidade do conjunto* $S$*. A complexidade computacional do algoritmo GreedyOPT é de* $O(N\log(N) + N|W|) = O(N\max(\log(N), |W|))$*, onde* $N = n + s$*.*

**Proof** Demanda-se $O(s\log(s))$ para determinar quais as reservas ainda não utilizadas devem ser reprocessadas já que as mesmas podem ser organizadas como um *heap*. Demanda-se também $O(N\log(N))$ para realizar a ordenação das requisições do lote, bem como $O(N|W|)$ para realizar o teste de existência de canal para acomodar a nova requisição realizado na linha 4. Assim, a complexidade do GreedyOPT é de $O(s\log(s)+N\log(N)+N|W|)$. Dado que $s\log(s) \leq N\log(N)$, fazendo $s = N$ tem-se que a complexidade do algoritmo é de $O(2N\log(N)+N|W|) = O(N\log(N)+N|W|) = O(N(\log(N)+|W|))$. Fazendo-se a complexidade em função do termo que majora a soma $(\log(N) + |W|)$, tem-se $O(N\max(\log(N), |W|))$ ∎

## 5. Exemplos Numéricos

Para avaliar o desempenho dos algoritmos, foram realizadas simulações usando a ferramenta OB2S (*Optical Burst Switching Simulator*) desenvolvido na Universidade Salvador [Maranhão et al. 2007]. Em cada simulação, foram geradas 10.000 requisições de reserva de recursos para rajadas ópticas. Cada uma das simulações foi replicada 20 vezes usando diferentes sementes de geração de números aleatórios e os intervalos de confiança possuem um nível de confiança de $95\%$.

As simulações foram executadas em dois cenários distintos. No primeiro cenário, reproduz-se ambiente proposto em [Kaheel and Alnuweiri 2005]. No segundo cenário, avalia-se o desempenho dos algoritmos em topologias de redes reais com um conjunto de parâmetros mais realista. Os resultados são apresentados nas subseções a seguir.

### 5.1. Simulações em topologia com nó OBS concentrador

Como dito, neste primeiro conjunto de simulações verifica-se o comportamento dos algoritmos propostos em um cenário controlado com poucos parâmetros de ajuste, além de comparar os resultados obtidos com aqueles já publicados em [Kaheel and Alnuweiri 2005].

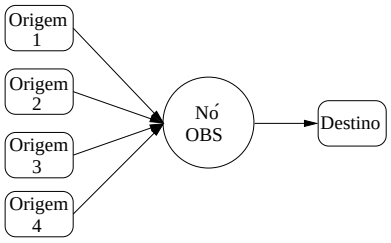
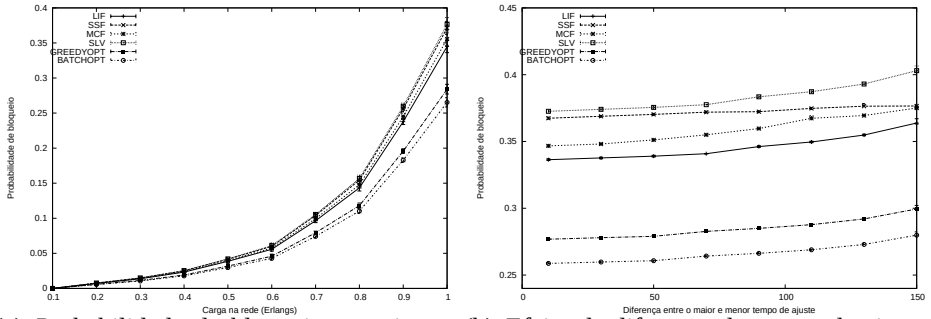



**Figura 4. Topologia usada no primeiro cenário de simulação.**

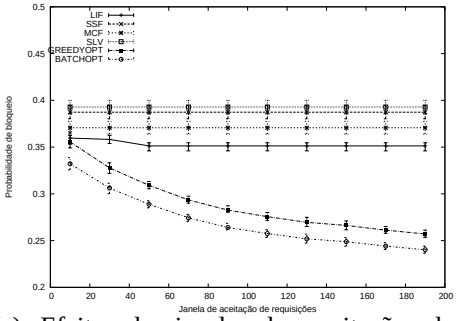
A Figura 4 apresenta a topologia usada no primeiro ambiente de simulação. Ela consiste de um único nó OBS de núcleo conectado a diversas fontes de tráfego e um único destino. Cada enlace de entrada possui dois comprimentos de onda separados (um para dados e um para controle). O enlace ligando o nó OBS de núcleo ao destino possui cinco comprimentos de onda (quatro para dados e um para controle) e cada um dos comprimentos de onda utilizados possui capacidade de 2.5 Gbps (OC-48).

Para estar em conformidade com os resultados discutidos em [Kaheel and Alnuweiri 2005], foi definida a constante $\tau$ como o tempo de transmissão de 1024 bits em um dos comprimentos de onda, ou seja, $\tau = \frac{1024}{2377728000} = 4.3e-7$ segundos. As rajadas são geradas pelas fontes de tráfego de acordo com uma distribuição de *Poisson* com tamanho médio das rajadas $\overline{b} = 81920$ bits. O tempo de ajuste é gerado de acordo com uma distribuição uniforme sobre o intervalo $[130\tau, 150\tau]$. A janela de aceitação de requisições possuiu um valor arbitrário de $100\tau$, o que dá aproximadamente $40\mu sec$.




(a) Probabilidade de bloqueio experimentada pelos algoritmos.

(b) Efeito da diferença do tempo de ajuste na probabilidade de bloqueio.




(c) Efeito da janela de aceitação de requisições na probabilidade.

**Figura 5. Resultados obtidos no cenário com nó concentrador.**

A Figura 5(a) mostra a probabilidade de bloqueio experimentada pelos algoritmos. É possível ver claramente que o algoritmo com menor probabilidade de bloqueio é o GreedyOPT.

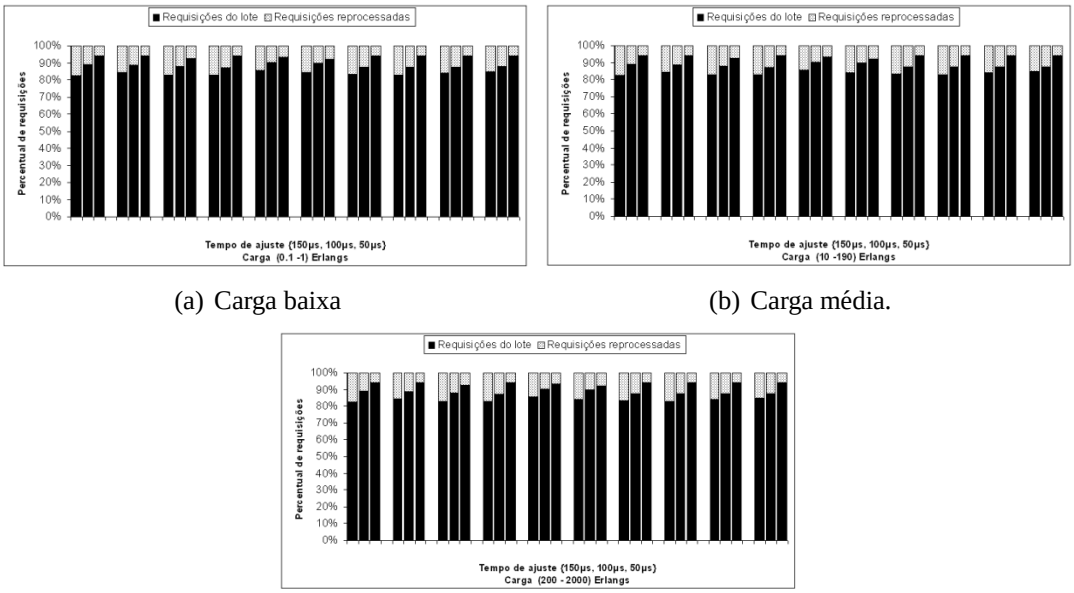
Foi avaliado, também, o efeito da diferença do tempo de ajuste na probabilidade de bloqueio. A diferença do tempo de ajuste foi definida como a diferença entre o maior e o menor tempo de ajuste gerados aleatoriamente, isto é, $T_{max} - T_{min}$. O valor de $T_{max}$ foi de $200\tau$ e o valor de $T_{min}$ foi gradativamente aumentado, diminuindo assim $T_{max} - T_{min}$. Neste caso, a carga de tráfego na rede foi de $99\%$ da capacidade do enlace.

A Figura 5(b) mostra que, de um modo geral, assim como observado nos experimentos em [Kaheel and Alnuweiri 2005], à medida em que se aumenta a diferença do tempo de ajuste, aumenta-se a probabilidade de bloqueio. É possível observar que a probabilidade de bloqueio experimentada pelo algoritmo GreedyOPT é inferior à dos demais algoritmos. O incremento da probabilidade de bloqueio à medida em que se aumenta a diferença entre os tempos de ajuste máximo e mínimo é um fenômeno comum em redes OBS que operam com o protocolo JET. Tal fenômeno é denominado *retro-*

*blocking* [Kaheel and H. 2004] e acontece devido ao uso de reserva retardada no protocolo JET. Assim, uma reserva, $r_i$, pode ser bloqueada por outra reserva, $r_{i+1}$, cujo tempo de início é anterior ao início da reserva $r_i$.

A Figura 5(c) mostra a probabilidade de bloqueio em função da janela de aceitação de requisições. Nesta avaliação, a diferença entre o maior e o menor tempo de ajuste foi de $50\tau$, a carga na rede foi de $99\%$ da capacidade do enlace e a janela de aceitação de requisição variou entre $10\tau$ e $190\tau$. É possível perceber que à exceção do algoritmo GreedyOPT, a probabilidade de bloqueio experimentada pelos algoritmos não depende do aumento da janela de aceitação de requisições. Na realidade, no caso dos algoritmos LIF, SSF, MCF e SLV, o sucesso ao alocar as requisições do lote depende muito mais do padrão de início e término das requisições do lote do que da quantidade de requisições.

No caso do algoritmo GreedyOPT, a probabilidade de bloqueio diminui com o aumento da janela de aceitação de requisições, pois à medida em que se aumenta a janela de aceitação de requisições, o número de requisições compondo cada lote aumenta. Como o algoritmos é ótimo, consegue sempre alocar o número máximo de requisições em cada lote, o que, ao final dos experimentos, resulta em uma probabilidade de bloqueio mais baixa.




(a) Carga baixa (b) Carga média.

(c) Carga alta.

**Figura 6. Percentual das requisições reprocessadas em cada lote.**

Dado que o algoritmo ótimo apresentado adiciona requisições já processadas cujas rajadas ainda não foram transmitidas ao lote em processamento, um experimento para medir o percentual de requisições reprocessadas em cada lote foi realizado. A estratégia de formação de lote JET-$\Delta$ [Figueiredo et al. 2009] foi usada com janela de aceitação de requisições igual a 1ms. Os resultados reportados foram calculados tomando-se a média aritmética em todos os nós da rede.

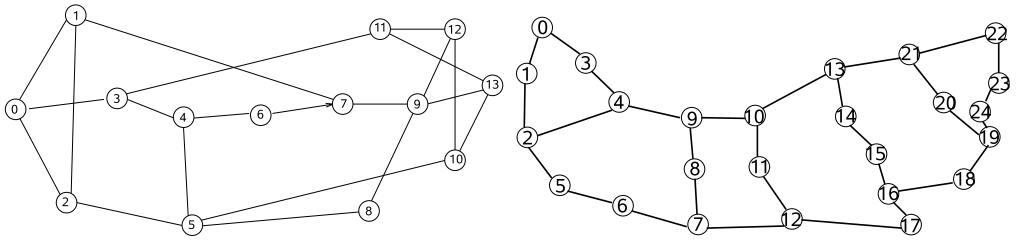
A Figura 6 mostra os resultados do percentual de reprocessamento dividido entre as requisições pertencentes ao lote (aquelas processadas pela primeira vez) e as

requisições reprocessadas. Os resultados são apresentados em função do tempo de ajuste contido nas requisições (sem levar em consideração a janela de aceitação de requisições). As barras verticais são dispostas em grupos de três. Cada barra dentro de um grupo representa o percentual de reprocessamento de requisições contendo um tempo de ajuste dentre $50\mu s$, $100\mu s$ e $150\mu s$. Cada grupo representa uma carga da rede. As simulações foram realizadas com três cenários de carga de tráfego: carga baixa, variando de $0.1$ a $1$ Erlangs com incremento de $0.1$ Erlangs; carga média, variando de $10$ a $190$ Erlangs, com incremento de $20$ Erlangs; e carga alta, variando de $200$ a $2000$ Erlangs, com incremento de $200$ Erlangs.

É possível perceber que, estatisticamente, não existe alteração no percentual de requisições reprocessadas em relação às requisições do lote (Figuras 6(a), 6(b) e 6(c)). Isto acontece pois à medida em que aumenta-se a carga na rede, mantendo-se inalterados os tempos de ajuste, aumenta-se proporcionalmente o número de requisições em cada lote e o número de requisições já processadas a espera da respectiva rajada. É possível também observar que o percentual de requisições reprocessadas aumenta à medida em que se aumenta o tempo de ajuste contido nas requisições. Isto acontece pois quanto maior o tempo de ajuste, mas tempo se passa entre o instante de término do processamento da requisição e a transmissão da rajada correspondente. Assim, a reserva fica mais tempo armazenada, permitindo, assim, que cheguem outras requisições e outros lotes sejam formados. Por fim, pode-se perceber que o percentual de requisições reprocessadas não ultrapassa $15\%$ das requisições do lote, mostrando que o fato de requisições serem consideradas novamente na solução do problema tem impacto relativamente pequeno no tempo de processsamento dos algoritmos.

### 5.2. Simulações com topologias de redes operacionais

Neste cenário, as simulações foram realizadas com as topologias das redes NSFNet e Abilene, apresentadas na Figura 7. Cada enlace da rede é constituído por uma fibra com 32 comprimentos de onda com capacidade de transmissão de 2.5 Gbps. O tempo de processamento do pacote de controle e configuração da malha de comutação dos comutadores é de $50\mu s$.

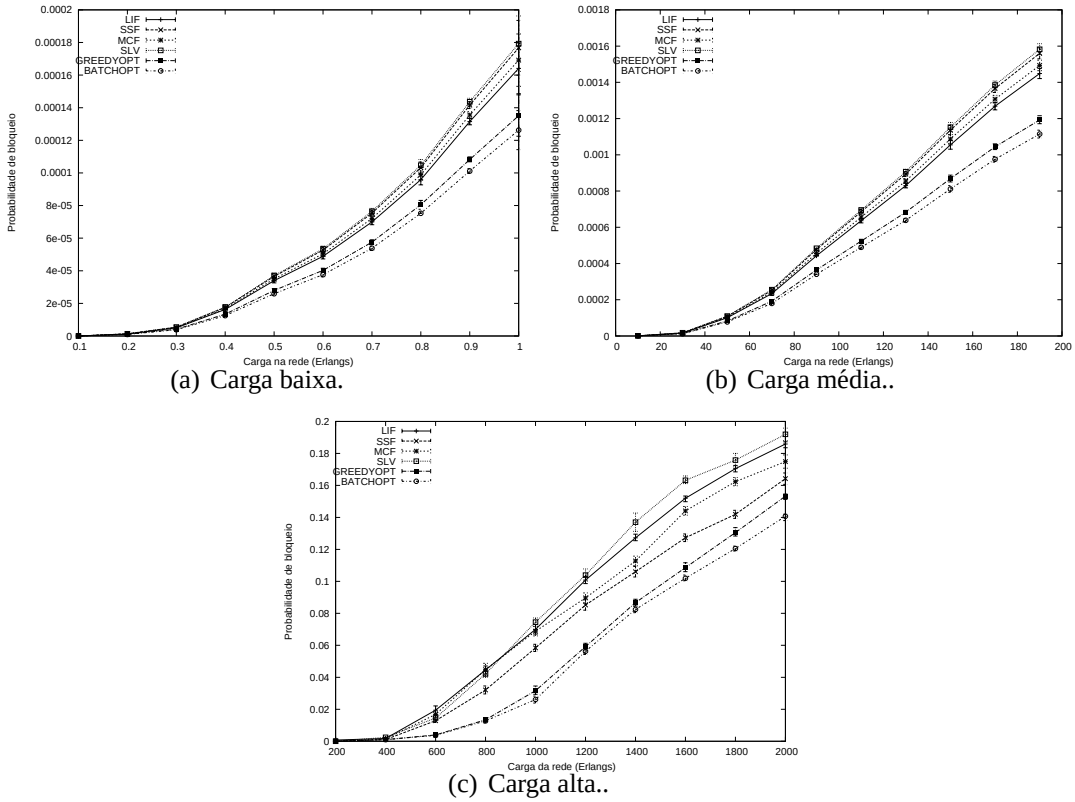


(a) *Backbone* NSFNet (b) *Backbone* Abilene

**Figura 7. Topologias usadas nas simulações.**

Todos os nós da rede são, potencialmente, uma origem ou um destino do tráfego, o que significa dizer que a cada requisição gerada, uma origem e um destino são sorteados de acordo com uma distribuição uniforme e uma rota é então criada entre o par. O tráfego é sempre gerado nos nós de origem e segue a distribuição de Poisson. O tamanho das rajadas varia de acordo com a distribuição exponencial negativa.

Foram realizadas simulações com tráfego com três diferentes intensidades de carga: carga baixa (carga variando de 0.1 a 1 Erlang), carga média (carga variando de 10 a 190 Erlangs) e carga alta (carga variando de 200 a 2000 Erlangs).




(a) Carga baixa. (b) Carga média..

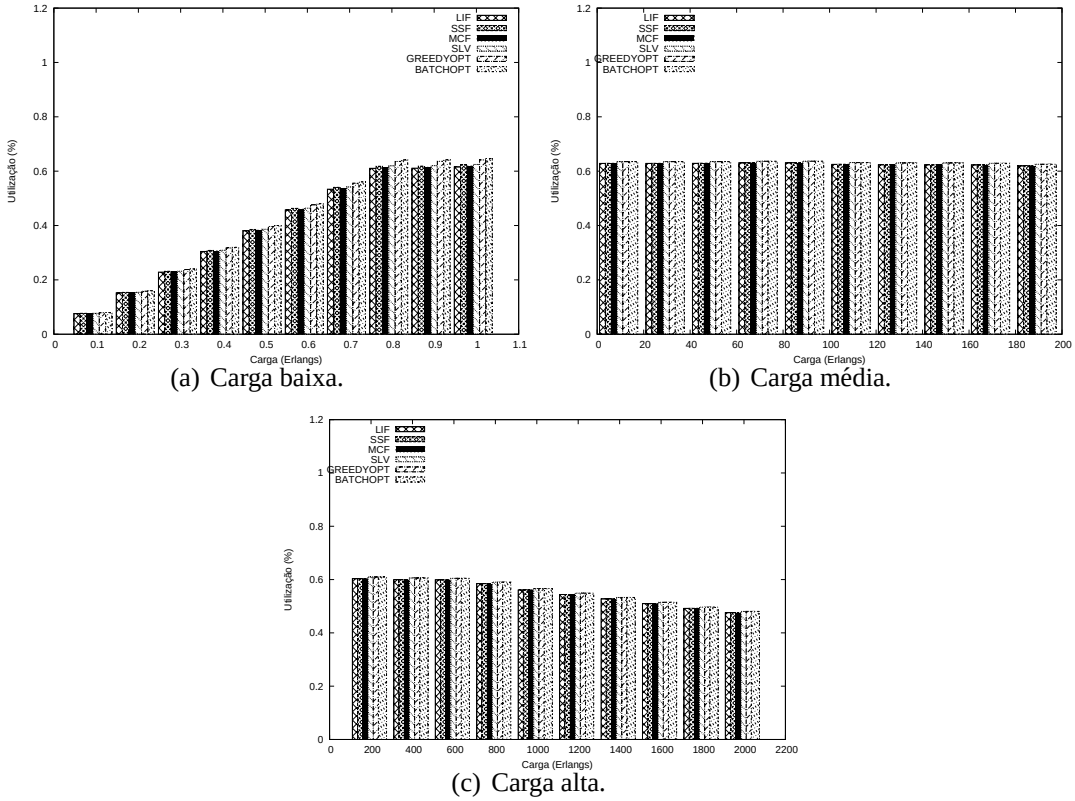
(c) Carga alta..

**Figura 8. Probabilidade de bloqueio.**

A Figura 8 mostra a probabilidade de bloqueio em função do aumento da carga na rede. Com o aumento da carga de tráfego na rede, o tamanho das cliques maximais do grafo de intervalos associado às requisições é aumentado, o que explica o crescimento das perdas. O algoritmo com a menor probabilidade de bloqueio foi o algoritmo GreedyOPT, com ganho de $35\%$ em relação ao SLV (algoritmo com pior desempenho). A seguir, vêm os algoritmos LIF, MCF e SSF com $12\%$, $8\%$ e $2\%$ de ganho na probabilidade de bloqueio em relação ao algoritmo SLV.

A Figura 9 apresenta a utilização efetiva da rede em função do aumento da carga de tráfego. É possível perceber que o algoritmo GreedyOPT produz a maior utilizações efetivas dentre os algoritmos avaliados. Uma maior utilização efetiva pode ser explicada por dois fatores: i) o algoritmo obteve menor probabilidade de bloqueio, fazendo com que um número maior de rajadas tenham sido transmitidas, e, consequentemente um maior tempo de duração das reservas bem sucedidas e ii) o algoritmo conseguiu atender rotas de tamanhos maiores, o que também se reflete em um aumento do tempo das reservas realizadas.

Além dos resultados numéricos, mediu-se também o tempo de execução dos algoritmos. As simulações foram realizadas em uma máquina Intel Pentium Core 2 Duo com 2.8Ghz e 4GB de memória RAM, executando o sistema operacional OpenSuse 11.1.




(a) Carga baixa. (b) Carga média.

(c) Carga alta.

**Figura 9. Utilização efetiva média da rede.**

O tempo de execução foi medido com o uso do comando *time*. Foram gerados 5 lotes de requisições, cada um contendo 500 requisições geradas aleatoriamente. Para cada lote foram realizadas 10 simulações e o tempo médio de execução de cada lote calculado, bem como a média do tempo de execução em todos os lotes.

O ganho relativo no tempo de execução de cada lote é apresentado na Tabela 1. O ganho relativo é definido como a diferença em percentual em relação ao valor de referência, que é o tempo de execução do algoritmo mais lento. É possível observar que o algoritmo mais lento em todos os cenários foi o algoritmo SLV. O algoritmo mais rápido foi o algoritmo SSF. O algoritmo GreedyOPT foi, no máximo $5\%$ mais lento que o al-

**Tabela 1. Ganho relativo no tempo médio de execução**

| | Lote 1 | Lote 2 | Lote 3 | Lote 4 | Lote 5 | Média |
|---|---|---|---|---|---|---|
| `Valor Referência` | $7.2\times10^{-2}$ | $8.0\times10^{-2}$ | $6.8\times10^{-2}$ | $7.6\times10^{-2}$ | $7.2\times10^{-2}$ | $7.3\times10^{-2}$ |
| `LIF` | 37% | 27% | 22% | 40% | 33% | 32% |
| `SSF` | 41% | 30% | 25% | 42% | 37% | 35% |
| `MCF` | 33% | 26% | 26% | 34% | 25% | 28% |
| `SLV` | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| `GreedyOPT` | 36% | 27% | 25% | 40% | 31% | 32% |
| `BATCHOPT` | 33% | 25% | 23% | 36% | 27% | 28% |

goritmo SSF e, em média $32\%$ mais rápido que o algoritmo SLV e $3\%$ mais lento que o algoritmo mais rápido, o SSF. Tais resultados mostram uma boa relação custo x benefício, já que o algoritmo GreedyOPT apresentou $35\%$ e $23\%$ de ganhos na probabilidade em relação aos algoritmos com pior e segunda melhor probabilidade de bloqueio, respectivamente. Além disso, o algoritmo GreedyOPT apresentou tempo de execução apenas $3\%$ mais lento do que o algoritmo mais rápido.

## 6. Conclusões e Trabalhos futuros

Este artigo apresentou uma solução em tempo linear para o problema de escalonamento em lote em redes OBS. A solução consiste na mudança na forma como o problema é modelado e na adoção de um algoritmo ótimo para o problema de *job scheduling* com máquinas idênticas. Resultados obtidos através de simulações mostram o bom desempenho do algoritmo quando comparado com outros da literatura.

## Referências

Arkin, E. M. and Silverberg, E. B. (1987). Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8.

Bouzina, K. I. and Emmons, H. (1996). Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3-4):379–393.

Charcranoon, S., El-Bawab, T. S., Cankaya, H. C., and Shin, J. D. (2003). Group scheduling for optical burst switched (obs) networks. In *Globecom*, pages 2745–2749.

Chen, Y., Qiao, C., and Xiang, Y. (2004). Optical burst switching (obs): A new area in optical networking research. *IEEE Network*, 18:16–23.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to algorithms*. MIT Press and McGraw-Hill.

Figueiredo, G. B. (2009). *Mecanismos de controle em redes de comutação de rajadas ópticas*. PhD thesis, Instituto de computação, Universidade Estadual de Campinas.

Figueiredo, G. B., Xavier, E. C., and da Fonseca, N. L. S. (2009). Um algoritmo ótimo para escalonamento de canais em lote em redes obs. In *Simpósio Brasileiro de Redes de Compuutadores*, pages 45–58.

Kaheel, A. and Alnuweiri, H. (2005). Batch scheduling algorithms for optical burst switching networks. *Lecture notes in Computer Science*, 3462/2005:90–101.

Kaheel, A. and H., A. (2004). Analytical evaluation of blocking probability in optical burst switching networks. In *IEEE International Conference on Communications*.

Maranhão, J., Soares, A., and Giozza, W. F. (2007). Estudo das arquiteturas de conversão de comprimento de onda em redes wdm com comutaçãoo de rajadas Ópticas. In *XXV SBRC*, pages 133–146.

Xiong, Y., Vandenhoute, M., and Cankaya, C. (2000). Control architecture in optical burst-switched wdm networks. *IEEE Journal of Selected Areas on Communications*, pages 1838–1851.

Yu, X., Li, J., Cao, X., Chen, Y., and Qiao, C. (2004). Traffic statistics and performance evaluation in optical burst switched networks. *Journal of Lightwave Technology*, 22(12):2722–2738.