Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Computer Communications 27 (2004) 13–26

computer communications

www.elsevier.com/locate/comcom

Dynamic bandwidth allocation in multi-class connection-oriented networks

Samer Taha*,[1], Mohsen Kavehrad

Electrical Engineering Department, The Pennsylvania State University, University Park, State College, PA 16802, USA

Received 2 August 2002; revised 17 April 2003; accepted 30 April 2003

Abstract

Multi-class network is becoming a more attractive solution to provide Quality-of-Service guarantee, as more quality-demanding applications are emerging. This research considers networks that provide connection-oriented services, as in ATM and MPLS technologies, for example. A common scheme for Dynamic Bandwidth Allocation (DBA) in connection-oriented communications is to dynamically segregate bandwidth between different traffic categories. These categories can represent topological Virtual Paths or different Classes-of-Service with different Quality-of-Service requirements. Bandwidth segregation can be efficient if and only if demands for different Classes-of-Service or different Virtual Paths can be predicted accurately. This led us to develop a novel algorithm that has a much wider vision in allocating resources than classical distributed algorithms. We call the proposed algorithm, Virtual Demand Distribution (VDD) algorithm. This algorithm utilizes signaling packets to broadcast information that enhances the performance of DBA algorithms. Mathematical analysis of multi-class connection-oriented networks and performance analysis/comparisons of the proposed VDD algorithm and a Simple Distributed algorithm are presented.

Keywords: Dynamic bandwidth allocation; Quality-of-Service; Multi-class Network

1. Introduction

Dynamic Bandwidth Allocation (DBA) for networks carrying multi-class traffic has been investigated within different contexts. We are concerned with architectures that work on the call level. Most of the work at this level was directed to dynamically allocate bandwidth to different Virtual Paths (VPs). The VP concept is a basic characteristic of ATM and ISDN networks, which provide connection-oriented services. VPs can be designed based on topological optimization. Also, bandwidth can be segregated between different CoSs with different QoS requirements, as in Ref. [1].

Bandwidth segregation is an effective way of managing resources to serve different CoSs of traffic with different QoS requirements. However, as was shown in Ref. [2], allocating bandwidth to VPs may lead to a lower efficiency (throughput) if bandwidth is not carefully (optimally) allocated. We point out that the basic reason for non-optimal allocation of bandwidth is the difficulty and inaccuracy in predicting demands for different CoSs. One of the essential reasons for this inaccuracy is the nature of most DBA algorithms, which are absolutely distributed.

Much work has been done recently on traffic engineering over MPLS architectures Ref. [3]. However, most of the efforts were directed to dynamically distribute load on alternative paths in order to efficiently utilize resources while achieving the best possible (QoS), like the work in Refs. [4,5]. Although CoSs were also supported in these studies, little efforts were directed to optimize the allocation of bandwidth among CoSs in a common link.

We believe DBA, traffic engineering, and routing are related tasks and cannot optimally perform without cooperation with each other. Routing and load distribution on alternative paths—Label Switched Paths (LSPs) in MPLS context—can be viewed as a necessary process on small time scales to efficiently utilize scattered resources. This process takes place at the edge of an MPLS network (ingress nodes). But at the same time, there can be competition on bandwidth between different LSPs, which may represent different CoSs, at each link in the network,

* Corresponding author. Tel.: +1-503-613-0270; fax: +1-503-613-6494.
E-mail address: samer.m.taha@intel.com (S. Taha).
[1] Current address: Mail Stop RA3-250, LTD Automation—Ronler Acres 3, Intel Corporation, Oregon, USA.

0140-3664/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0140-3664(03)00145-2

both at the core and at the edge of an MPLS network. As the aggregated demand for different CoSs/LSPs varies slowly over time, a DBA process is necessary to optimally distribute bandwidth between competing CoSs/LSPs based on a specific objective function. This objective function usually takes into consideration user-utilities and revenues, which are functions of QoS, pricing model, and regulations governing different CoSs. From here on, we will use the concept of CoSs to refer to both LSPs and classes of different levels of QoS in any connection-oriented multi-class network.

In the context of connection-oriented services, usually objective functions can be designed at higher levels, like controlling the blocking rates of calls. We can assume that a lower level control will take the responsibility of computing the required bandwidth by each call, based on the CoS of that call, the QoS specifications of that CoS, and the optimization criteria defined for that CoS. This computed bandwidth is usually called 'effective bandwidth' Ref. [6]. So, these low level controls may, for example, minimize the average delay of sessions within a specific CoS or minimize the average packet loss rate for sessions within another CoS, and so on. On the other hand, a higher-level control is needed to allocate (segregate) bandwidth between the CoSs in an optimal way. We recommend that these two levels of control be carried at different layers, this would achieve two objectives: first, routing stability; since this dynamic segregation of bandwidth would usually be a slow process and thus QoS routing algorithms can have valid and meaningful information about the status of the network. Second, controlling call blocking rates of different CoSs would become a possible task. Controlling the call blocking rates is expected to be an important issue, especially when advanced pricing systems will be introduced Ref. [7].

When we consider different DBA algorithms, we can classify them to centralized and decentralized algorithms. Centralized algorithms provide a single point of failure and require huge amounts of information, which is, usually, broadcasted using flooding protocols. Also, their ability to provide a solution for a large-scale network with varying demands in an appropriate time is questionable. On the other hand, decentralized algorithms are implementable, but cannot provide an optimal solution because they don't have a global vision of the network.

The major contribution of this paper is that we propose and evaluate a novel algorithm that performs a slow DBA to allocate bandwidth globally over all backbone links of a multi-class connection-oriented network. We call this proposed algorithm Virtual Demand Distribution (VDD) algorithm. The VDD algorithm is a distributed algorithm that can approach global optimality. The VDD collects more information than classical distributed algorithms, but the amount of collected information is still small, compared with the amount of information that any centralized algorithm needs in order to achieve global optimality. This extra information is obtained by utilizing the signaling messages used for establishing new connections in any multi-class network providing connection-oriented services. Also, to provide stability for routing algorithms we avoid using rapid DBA (on a per-session level) control. Instead, our slow DBA system deals with aggregates of traffic and responds to variations in demands on a relatively slow basis. The word 'relatively' here means with respect to the speed of updating routing tables.

We approach the development of this algorithm throughout two stages of analysis. In the first stage, we analyze the dynamics of DBA implemented in a single link, assuming stationary traffic with variable-size calls and known statistics. In this stage, we demonstrate the effect of transient responses on predicting blocking rates of calls when a Complete Partitioning (CP) schema is used and we develop an algorithm for optimal resource allocation by extending the recursive algorithm proposed in page 25 of Ref. [8]. In the second stage, we analyze a multi-link multi-class connection-oriented loss network and emphasize the difficulty in accurately predicting demands in such an environment. In this stage we develop a framework for analyzing demands and call blocking rates of non-stationary traffic in a multi-link network and then we develop a novel algorithm to accurately predict demands for different CoSs in a multi-link multi-class loss network. These accurately predicted demands could then be used to allocate resources based on the algorithm that was developed in the first stage. The algorithms developed in these two stages combined together form the proposed VDD algorithm. To evaluate the performance of the proposed VDD algorithm we used Bones Designer, a sophisticated simulation tool, and built a multi-node network with a configuration shown in Fig. 1. We built this simulation to emulate an as real a network as possible, with 48 h of non-stationary varying traffic generated at each node. A detailed description of this simulation is provided in Section 5. In addition to VDD algorithm, we simulated performance of an equivalent Simple Distributed (SD) algorithm, applied to two CoSs. As we mentioned above, in Section 2 we start by analyzing DBA algorithms and call blocking rates in a single communication link.

2. Single-link analysis

When different traffic categories or classes share a common link, three possible resource-sharing techniques can be considered. These are Complete Sharing (CS), Virtual Partitioning (VPT), and Complete Partitioning (CP). In CS, no control over the performance of different CoSs can be exercised. However, it is the simplest possible technique and works well under light loads. But, a well-designed network should not be lightly loaded all the time. In CP, a strict control over the performance of any CoS can be exercised, but wastage of resources is an issue to worry about. Under high load conditions, CP works very well if

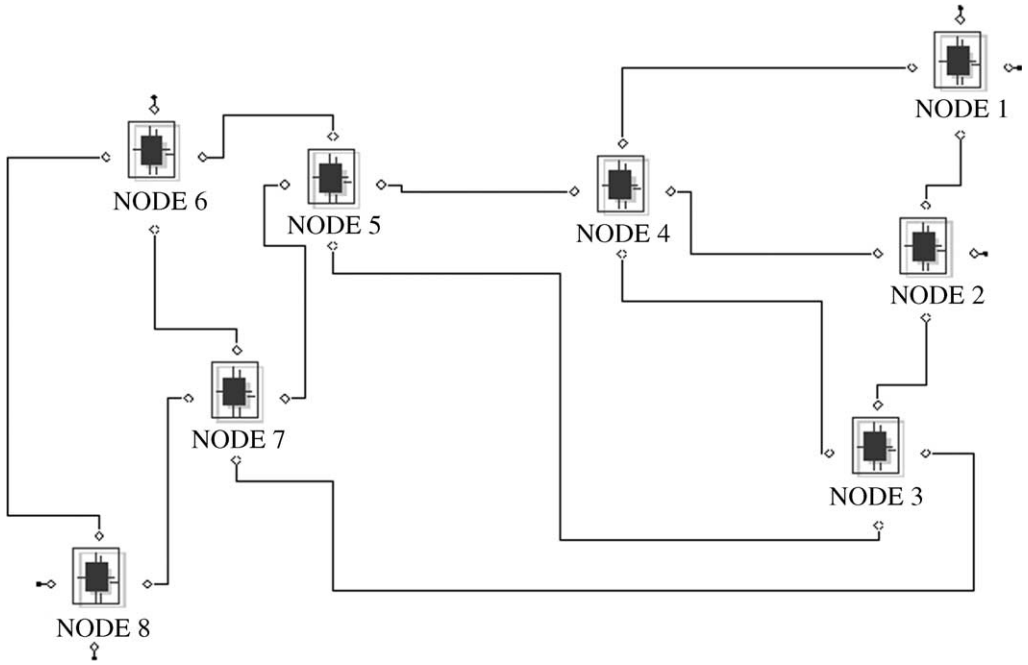

Fig. 1. The configuration of the network used in simulating the VDD and other algorithms, (simulation tool: Bones Designer).

and only if demands on different CoSs can be estimated accurately. VPT can be considered as a mixture of both CS and CP techniques. Resources are not wasted under VPT technique, but because of the possible sharing of resources by different CoSs and because performance depends on allocated capacities to different CoSs, no strict control over the blocking rates of different CoSs can be achieved, especially when the traffic is non-stationary. Trunk-Reservation DBA is an example of a VPT scheme.

In this paper, we are interested in a highly-utilized backbone network supporting different CoSs which are sensitive to call blocking rates. Because of the differences in these CoSs due to differentiation in treating different sessions of these CoSs at the packet level, or because some CoSs may represent VPs with different geographical significance, a differentiation between the blocking rates of these CoSs is desired. Thus, we will adopt a CP scheme, where capacities will be dynamically allocated in a non-stationary traffic environment. To approach the analysis of call blocking rates estimates in a multi-link multi-class network carrying non-stationary traffic and adopting a DBA under a CP scheme, we start in Section 2.1 by analyzing the call blocking rates in a single-link with one CoS carrying stationary traffic, then we will continue building on the results of this subsection in the rest of the paper.

2.1. Call-blocking probability for one CoS of stationary traffic with variable size bandwidth requests

We consider here predicting the call blocking rate experienced by calls with stationary Poisson arrival process, stationary exponential holding time, and variable-size requests with known distribution. Such systems are known as stochastic Knapsack systems. In Appendix A, we review several formulas and algorithms developed so far with different model assumptions, and we show that there is no closed-form formula for predicting call blocking rates in such systems, neither in an exact nor in an approximate form. Thus, we found that the following recursive algorithm is the best to build upon for deriving a DBA algorithm with the objective of controlling the call blocking rates of different CoSs. This algorithm assumes a model where there is a limited number of bandwidth sizes, such that a size-k session requests b_k units of bandwidth with a probability of $P_b(b_k)$, $\rho_k = \lambda_k/\mu_k$, λ_k is the mean arrival rate of size-k calls, μ_k^{-1} is the mean holding time of size-k calls, M is the number of different bandwidth sizes, and C is the capacity of the link which equals multiples of units of bandwidth. The expected call blocking rate of size-k calls (B_k) can be found as follows:

Algorithm 1.

Set $g(0) = 1$, $g(j) = 0$ for $j < 0$.
For $h = 1,2,\ldots,C$
{

$$g(h) = (1/h)\sum_{k=1}^{M}(b_k)\rho_k g(h - b_k). \quad (1)$$

}
Set

$$G = \sum_{h=0}^{C} g(h). \quad (2)$$

For $h = 0,1,\ldots,C$
{

$$q(h) = g(h)/G. \tag{3}$$

}

$$B_k = \sum_{h=C-b_k+1}^{C} q(h). \tag{4}$$

In Section 2.2, we use Algorithm 1 to derive an algorithm that allocates bandwidth between CoSs of non-stationary traffic sharing a common link with the objective of controlling the ratios between call blocking rates of different CoSs. For simplicity, and without loss of generality, we consider two CoSs.

2.2. Controlling call-blocking probabilities for two CoSs of non-stationary traffic with variable size bandwidth requests

Our objective is to dynamically allocate bandwidth between two CoSs, such that the ratio of class-2 blocking rate (B_p^2) to class-1 blocking rate (B_p^1) is β, over the decision interval t_d. A class-1 call requests a bandwidth b_m^1 with probability $P_b^1(b_m^1)$, a class-2 call requests a bandwidth b_n^2 with probability $P_b^2(b_n^2)$. Class-1 mean arrival rate is A_1 calls/s and class-2 mean arrival rate is A_2 calls/s. Class-1 mean holding time is S_1^{-1} s, class-2 mean holding time is S_2^{-1}. We also define $\bar{b}^1$: the average bandwidth size of class-1 calls, $\bar{b}^2$: the average bandwidth size of class-2 calls, $\lambda_m^1 = P_b^1(b_m^1)A_1$, $\lambda_k^2 = P_b^2(b_n^2)A_2$ calls/s, $\mu_m^1 = S_1$, $\forall m$, and $\mu_n^2 = S_2$, $\forall n$, $m = 1,2,\ldots,M_1$ and $n = 1,2,\ldots,M_2$, where M_1 is the number of possible bandwidth sizes at class-1 and M_2 is the number of possible bandwidth sizes at class-2. In general, we assume non-stationary stochastic process where the mean arrival rate of class-k is $A_k(t)$, t is time. However, the holding time is assumed to be a stationary process with constant mean value ($1/S_k$) seconds. We assume that

$$\frac{\partial A_k(t)}{\partial t} \approx 0.0 \text{ for } t_0 \le t \le t_0 + t_d, \tag{5}$$

thus the system is assumed to be under stationary traffic throughout the decision interval t_d.

An important phenomenon arises here, at each decision instance a situation where the allocated capacity is less than the utilization of class-k may occur for a duration of time equals to ζ_k seconds, where ζ_k is the time for class-k utilization to drop below the newly allocated capacity. This is because we assume that the system cannot drop any active session, in order to maintain QoS parameters promised to users. During this time interval (ζ_k) all class-k arriving calls will be blocked. Expected value of $\zeta_k(\bar{\zeta}_k)$ can be exactly found if all calls utilizing class-k require the same amount of bandwidth. In this case, the problem becomes finding the expected time for the number of class-k active calls to drop, say from ω_k to ω_k', while no new calls are accepted. This is a death process, thus:

$$\bar{\zeta}_k = \sum_{m=\omega_k'}^{\omega_k} \frac{1}{mS_k} \text{ seconds}, \tag{6}$$

where the expression in Eq. (6) can be approximated as follows:

$$\bar{\zeta}_k \approx \frac{(\omega_k - \omega_k')}{S_k(\omega_k + \omega_k')/2} \text{ seconds}. \tag{7}$$

The approximation in Eq. (7) is accurate as long as $(\omega_k - \omega_k')/\omega_k' \ll 1$, which is the case in a realistic backbone network, especially as long as the assumption in Eq. (5) is true. Note, ω_k' is determined by the capacity allocation decision for class-k.

However, in our case, sessions utilizing class-k require different amounts of bandwidth. Thus, based on the available information to the system, ω_k and ω_k' can be approximated as the utilization and the allocated capacity, respectively, divided by the average size of the bandwidth requests. Let Q_1 be the allocated capacity (bps) to class-1, $(W_t - Q_1)$ is the allocated capacity (bps) to class-2. Let u_k be the utilization (bps) of class-k just before calculating a new bandwidth allocation decision. We can define the following approximations for the length of the transient period:

$$\bar{\zeta}_1 \approx \frac{(u_1/\bar{b}^1 - Q_1/\bar{b}^1)}{(S_1)(u_1/\bar{b}^1 + Q_1/\bar{b}^1)/2} U(u_1 - Q_1) \text{ seconds}, \tag{8}$$

$$\bar{\zeta}_2 \approx \frac{(u_2/\bar{b}^2 - [W_t - Q_1]/\bar{b}^2)}{(S_2)(u_2/\bar{b}^2 + [W_t - Q_1]/\bar{b}^2)/2} \times U(u_2 - [W_t - Q_1]) \text{ seconds}. \tag{9}$$

Based on the assumption in Eq. (5), and assuming that A_1 and A_2 can be accurately estimated every t_d seconds, an algorithm (Algorithm 2) for bandwidth allocation between the two CoSs can be developed by incorporating the transient effects in Eqs. (8) and (9), and by extending Algorithm 1. The objective of Algorithm 2 is stated in Eq. (10).

Every t_d seconds, find Q_1 such that:

$$\frac{\text{Number of class} - 2 \text{ blocked calls throughout } t_d \text{ seconds}}{\text{Number of class} - 1 \text{ blocked calls throughout } t_d \text{ seconds}} = \beta. \tag{10}$$

Note, in Algorithm 2, all bandwidth quantities must be integer numbers, the allocation decision is updated every t_d seconds, $Q_1(n)$ is the nth allocation decision.

Algorithm 2.

Estimate A_1 and A_2.
Measure u_1 and u_2.

Set $Q_1' = Q_1(n-1)$
While ($|\delta| > \epsilon$)
{
Set $g_1(0) = 1$, $g_2(0) = 1$, $g_1(j) = 0$ for $j < 0$, $g_2(j) = 0$ for $j < 0$.
For $h = 1, 2, \ldots, Q_1'$
{

$$g_1(h) = (1/h)\sum_{k=1}^{M_1}(b_k^1)\left(\frac{P_b^1(b_k^1)A_1}{S_1}\right)g_1(h-b_k^1). \quad (11)$$

}
For $h = 1, 2, \ldots, [W_t - Q_1']$
{

$$g_2(h) = (1/h)\sum_{k=1}^{M_2}(b_k^1)\left(\frac{P_b^2(b_k^2)A_1}{S_2}\right)g_2(h-b_k^2) \quad (12)$$

}
Set

$$G_1 = \sum_{h=0}^{Q_1'} g_1(h). \quad (13)$$

Set

$$G_2 = \sum_{h=0}^{[W_t-Q_1']} g_2(h). \quad (14)$$

For $h = 0, 1, \ldots, Q_1'$
{

$$q_1(h) = g_1(h)/G_1. \quad (15)$$

}
For $h = 0, 1, \ldots, [W_t - Q_1']$
{

$$q_2(h) = g_2(h)/G_2. \quad (16)$$

}

$$B_p^1 = \sum_{k=1}^{M_1} P_b^1(b_k^1) \sum_{h=Q_1'-b_k^1+1}^{Q_1'} q_1(h). \quad (17)$$

$$B_p^2 = \sum_{k=1}^{M_2} P_b^2(b_k^2) \sum_{h=Q_1'-b_k^2+1}^{[W_t-Q_1']} q_2(h). \quad (18)$$

$$\phi = \frac{(A_2B_p^2(t_d - \bar{\zeta}_2) + A_2\bar{\zeta}_2)/A_2t_d}{A_1B_p^1(t_d - \bar{\zeta}_1) + A_1\bar{\zeta}_1/A_1t_d},$$
($\bar{\zeta}_1$ from Eq. (8) and $\bar{\zeta}_2$ from Eq. (9)) (19)

If ($\phi > \beta$
{
$Q_1' = Q_1' - 1$.
}
If $\phi < \beta$
{
$Q_1' = Q_1' + 1$.
}
$\delta = \phi - \beta$.
}
$Q_1(n) = Q_1'$

For understanding Eqs. (11–14), reader is advised to refer to Ref. [8]. Eqs. (17) and (18), define the average blocking rates for the two CoSs. The second summation finds the probability of blocking a specific size request and the first summation finds the average probability of blocking any-size request. In Eq. (19), the effect of the transient periods, approximated in Eq. (8) and (9), is included. Class-k blocking probability found in Eqs. (17) and (18) is meaningful after the end of the transient period, if it exists. Nominator in Eq. (19) divides the expected number of blocked class-2 calls throughout the decision period (t_d) by the number of all calls expected throughout the same period. Denominator of Eq. (19) does the same, but for class-1. Note also, in Eq. (19), the values of $\bar{\zeta}_1$ and $\bar{\zeta}_2$ belong to one the following three cases, ($\bar{\zeta}_1 = 0, \bar{\zeta}_2 = 0$), ($\bar{\zeta}_1 = 0, \bar{\zeta}_2 > 0$), or ($\bar{\zeta}_1 > 0, \bar{\zeta}_2 = 0$). It is clear that the inclusion of the transient response in the calculations of the DBA equations will work as a slowing tool. It will slow down the changes in the allocated bandwidth if a sudden and large change occurs in the demands. This is desirable, since by this, the network will not make dramatic changes in the allocations unless new demands pattern are long-lived. ϵ is a parameter that defines the desired accuracy in forcing the differentiation ratio β.

In this paper, we try to emphasize the importance of accurate predictions of demands. We show, by utilizing the above algorithm, that estimating average arrival rates of two CoSs based on local information is not efficient, even if networks have simple structures. To overcome this problem, which is a serious one in CP schemes, we propose a novel algorithm that utilizes signaling messages in connection-oriented networks with reservations to approach global optimality in predicting demands at each router/switch.

3. Loss network analysis

In this section, we provide a loss network analysis for a multi-class network, for simplicity and without loss of generality we will restrict the analysis to two CoSs. However, this analysis can be extended to any number of CoSs.

Let set N be the set of all nodes in a network that supports two connection-oriented CoSs.
Let $i, j \in N$, where $i \neq j$.
Let l^{ij} be the output port connecting node i to j.

A flow is defined as a sequence of output ports visited by the traffic of one or more sessions carrying information between two nodes. $f_{v,n}^{ij}$ is the nth flow moving through l^{ij} and belongs to class v, $v = 1, 2$. Two traffics connecting the same two nodes, but each one moving through different intermediate nodes are considered as two different flows. $a(f_{v,n}^{ij})$ is the average arrival rate of calls forming the flow $f_{v,n}^{ij}$, the arrival process is assumed to be Poisson. For each l^{ij} we define a tree T_v^{ij}, which is a set that contains all $f_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$, where σ_v^{ij} is the total number of flows at l^{ij} that belong to class v. Any flow that belongs to class v and passes through l^{ij} is a member of T_v^{ij}. For every tree T_v^{ij}, we define roots R_v^{ij} and branches B_v^{ij}. R_v^{ij} is a set that contains $r_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$ where $r_{v,n}^{ij}$ is the part of $f_{v,n}^{ij}$ that starts at the source of $f_{v,n}^{ij}$ and ends at node i. B_v^{ij} is a set that contains $b_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$ where $b_{v,n}^{ij}$ is the part of $f_{v,n}^{ij}$ that starts at a node j and ends at the destination of $f_{v,n}^{ij}$. We define $T^1(T_v^{ij})$ which is a group of trees. A tree $T_v^{xy} \in T^1(T_v^{ij})$ if and only if there exists at least one flow $f_{v,n}^{ij} \in T_v^{ij}$ such that $l^{xy} \in f_{v,n}^{ij}$. We call the group $T^1(T_v^{ij})$ the group of son-trees (first generation trees) of T_v^{ij}. Similarly, $T^2(T_v^{ij})$ is the group of grandson-trees (second generation trees) of T_v^{ij}. A tree $T_v^{ab} \in T^2(T_v^{ij})$ if and only if there exists at least one flow $f_{v,n}^{ij} \in T_v^{ij}$ such that $l^{xy} \in f_{v,n}^{ij}$ and there exists at least one flow $f_{v,k}^{xy} \in T_v^{xy}$ such that $l^{ab} \in f_{v,k}^{xy}$. In a similar way, we can define $T^z(T_v^{ij})$, where z can take any value between 0 and ∞. Note that $T^0(T_v^{ij}) = \{T_v^{ij}\}$.

Claim 1. *There exists a bandwidth allocation algorithm that can perform as optimal as a centralized algorithm in a multi-class connection-oriented network while working in a distributed way and requiring an amount of information much less than the global demand matrix.*

A detailed proof for the above claim is documented in Appendix B. So far, we claim the existence of an algorithm that runs in a distributed way but still can approach global optimality by collecting an amount of information much less than the global demand matrix. We believe VDD is a very strong candidate to be such an algorithm.

In a practical environment, the demand of any flow is not known a priori. Demands can be measured, and based on measured demands over an averaging window of Δt starting at time t_0, demands can be predicted throughout $(t_0 + \Delta t, t_0 + 2\Delta t)$. The accuracy of this prediction depends on the nature/correlation of the variation in demands and on Δt. As Δt increases, above a certain value, the correlation between successive intervals decreases. On the other hand, as Δt decreases below a certain value, there will not be enough time to estimate the average arrival rate of calls and hence estimate the demand. In Ref. [9], it was shown that the variations in aggregated demands are usually slow enough such that a decision interval of 1 h would even be faster than necessary. Measuring demand needs time and so does broadcasting information. In addition to timing considerations, there are complicated forms of dependency among decisions computed at different output ports throughout the whole multi-class network. With all these considerations, and based on previous discussions, we developed the VDD algorithm. In the Section 4, we explain and state the VDD algorithm.

4. The virtual demand distribution (VDD) algorithm

Let t_d be the decision interval.
Let t_u be the update interval.
$T_d = mt_u$, where $m \geq$ (maximum number of hops), this is a necessary condition.

The following requirements are necessary for any network for it to be able to support the VDD algorithm:

1. Signaling packets (call-setup packets) must be used to establish any new connection.
2. Packets belonging to a specific session have to follow the same path that was assigned initially by the network.
3. Acknowledgments have to follow the same path—in the reverse direction—that was followed by the call-setup packets. Acknowledgments have to keep all the information collected by the call-setup packets.
4. Call-setup packets belonging to different flows and moving through the same output port must have different labels, or can be identified and assigned different labels.
5. Call-setup packet must continue its journey to the destination edge device, even if the requested bandwidth is rejected by an intermediate node.

The above requirements can be fulfilled by ATM networks Ref. [10]. MPLS architecture can easily support the above requirements too. Each call-setup packet records the following information at each output port it moves through:

1. Whether or not this output port has accepted this new call under the specified CoS.
2. The percentage increase or decrease of the expected blocking rate—based on the current updated decision—with respect to the current measured blocking rate under the specified CoS. We call this the Virtual Percentage Change in Blocking Rate (VPCBR).

This information must be written in the call-setup packet in a way that reflects the order in which the output ports have been visited by. Also, any output port, say l^{ij}, must know its order when it writes information in any call-setup packet and bind this order $O_{L_{v,n}^{ij}}^{ij}$ with the label $L_{v,n}^{ij}$ of that call-setup packet. $L_{v,n}^{ij}$ is the label assigned to flow $f_{v,n}^{ij}$ at output port l^{ij}. This binding needs to be done one time when a new flow starts to move through an output port.

At each output port, the following data is retrieved from each Acknowledgment (ACK) that moves through it. To simplify the notation, we will drop the superscript ij. Also,

we define $l_{v,n}(k)$ to be the kth output port visited by the nth flow $f_{v,n}$ at the current output port at which the algorithm is running, $k = 1, 2, \ldots, Y_{L_{v,n}}$, where $Y_{L_{v,n}}$ is the number of output ports visited by the flow labeled by $L_{v,n}$.

Assume that the following data is going to be collected and worked with independently, at each output port. A linked-list data structure is to be created, each label $L_{v,n}$ is assigned one list. Each element of the list assigned to $L_{v,n}$ contains four counters and two real numbers. $CA_{L_{v,n}}(k)$ to count number of accepted vth class calls, $CR_{L_{v,n}}(k)$ to count number of rejected vth class calls, $d_{L_{v,n}}(k)$ to record VPCBR value of the vth CoS at the kth output port of the flow labeled $L_{v,n}$, $k = 1, 2, \ldots, Y_{L_{v,n}}$; $v = 1, 2$.

Briefly, the basic idea in the VDD algorithm is to let every output port computes and declares a *virtual* decision every t_u seconds. The decision is *virtual* because it will not be implemented physically. It can be interpreted as if the output port is 'saying': 'I'm planning to implement this decision, so, other output ports try to optimize your decisions based on my decision'. However, we do not distribute the virtual decision itself. Rather, we distribute the expected effect of this virtual decision on the current blocking rate at this specific output port. Note that, by this idea, we give the network a chance to converge to the optimal allocation without physically changing the allocations more frequently, since rapid changes in the allocations will not be helpful in stabilizing QoS routing in a multi-class network. In the following, we will first present a pseudo-code of the algorithm and then we explain how it works in more details.

While (network is running)
{
initialize all counters;
$x = 0$; set $clock = 0$;
wait t_r seconds;
  for ($x = 1$ to m)
  {
  $t = 0$;
    while ($t \le t_u$)
    {
      if (ACK arrived)
      {
      read label $L_{v,n}$;
        $\forall k = 0, 1, 2, \ldots, Y_{L_{v,n}}$ **do**:
        {
        read $d_{L_{v,n}}(k)$;
        $CA_{L_{v,n}}(k) = CA_{L_{v,n}}(k) + 1$, if the call was accepted at $l_{v,n}(k)$;
        $CR_{L_{v,n}}(k) = CR_{L_{v,n}}(k) + 1$, if the call was rejected at $l_{v,n}(k)$;
        }
      }
    }
    Virtual_Decision(1);
  }
  if (this link is a DownLink)
  {compute and declare the semi-final-1 VPCBR // *which equals the last computed VPCBR*}
  else (wait until all semi-final-1 VPCBRs are received from all $l_{v,n}(O_{L_{v,n}} + 1)$, $n = 1, 2, \ldots, \sigma_v$; $v = 1, 2$)
  {**Virtual_Decision**(2) \\ *this is a semi-final-1 VPCBR*}
  wait t_{pr} seconds;
  if (this link is a DownLink)
  {**Virtual_Decision**(1) \\ *this is a semi-final-2 VPCBR*}
  else (wait until all semi-final-2 VPCBRs are received from all $l_{v,n}(O_{L_{v,n}} + 1)$, $n = 1, 2, \ldots, \sigma_v$; $v = 1, 2$)
  {**Virtual_Decision**(3) \\ *this is a semi-final-2 VPCBR*}
  if (this link is an UpLink)
  {**Virtual_Decision**(4) \\ *this is a final VPCBR*}
  else (wait until all final VPCBRs are received from all $l_{v,n}(O_{L_{v,n}} - 1)$, $n = 1, 2, \ldots, \sigma_v$; $v = 1, 2$)
  {**Virtual_Decision**(3) \\ *this is a final VPCBR*}
  when (**clock** $= t_d$){physically implement Q_1}
}

Virtual_Decision(e) \\ this is a function that accepts an integer (1,2,3, or 4)
{
  switch (e){
  case 1: $\{A = 1, B = O_{L_{v,n}} - 1\}$
  case 2: $\{A = O_{L_{v,n}} + 1, B = Y_{v,n}\}$
  case 3: $\{A = 1, B = Y_{v,n}$; such that $k \neq O_{L_{v,n}}\}$
  case 4: $\{A = 2, B = Y_{L_{v,n}}\}$
}

$$A_v = \sum_{n=1}^{\sigma_v} \left(\frac{CA_{L_{v,n}}(1) + CR_{L_{v,n}}(1)}{xt_u} \prod_{k=A}^{B} (1 - \psi_{L_{v,n}}(k)) \right); \quad v = 1, 2. \tag{20}$$

Where:

$$\psi_{L_{v,n}}(k) = \begin{cases} \left(\dfrac{CR_{L_{v,n}}(k)}{CA_{L_{v,n}}(k) + CR_{L_{v,n}}(k)} \right) \times (1 + d_{L_{v,n}}(k)), & \text{if } CR_{L_{v,n}}(k) > 0 \\ d_{L_{v,n}}(k), & \text{if } CR_{L_{v,n}}(k) = 0 \end{cases}; \tag{21}$$

Feed A_1 and A_2, computed in Eq. (20), into **Algorithm 1** and find Q_1; \\ *new virtual bandwidth allocation decision*

$$d^v = \frac{(\mathrm{EBR}_v - \mathrm{CBR}_v)}{\mathrm{CBR}_v}; \text{ \\\\new VPCBR at this output port}$$

Declare the VPCBR \\ i.e. the d^v;
Where: EBR_1 = Expected Blocking Rate under 1st CoS due to the current virtual decision. Feed A_1 and Q_1 into **Algorithm 1** to get EBR_1.

EBR_2 = Expected Blocking Rate under 2nd CoS due to the current virtual decision. Feed A_2 and $[W_t - Q_1]$ into **Algorithm 1** to get EBR_2.

CBR_1 = Current Blocking Rate measured under 1st CoS.

CBR_2 = Current Blocking Rate measured under 2nd CoS.

The algorithm can be explained as follows:

After every physical implementation of bandwidth allocation, the algorithm waits for t_r seconds. This is to wait for the transient response to approximately die. t_r can be designed to be an adaptive parameter, but since we are following the slow variations of demands, we can simply set t_r based on the worst case scenario. As we explained earlier, during a transient period, all incoming calls will be rejected until the utilization drops below the new allocated bandwidth. These rejected calls if accounted for, will feed the algorithm with wrong information, which is why the algorithm must wait until the transient period ends. The time needed for the transient period to end can be approximately estimated as in Eqs. (8) and (9).

Throughout every update interval t_u, information is collected at each output port from the returning ACK packets. For example, at output port l^{ij}, the algorithm collects the rate of accepting and rejecting calls at each output port related to T_v^{ij}, $v = 1, 2$. At the same time, the algorithm collects all the VPCBRs declared at all output ports related to T_v^{ij}, $v = 1, 2$. During the first m updates, the algorithm considers only information collected from all output ports related to R_v^{ij}, $v = 1, 2$. Here, the algorithm does not look at output ports related to B_v^{ij}, $v = 1, 2$; it is clear that, if the algorithm does look at both the roots and branches of the tree T_v^{ij} at the same time, oscillation in the decisions can occur. The idea of looking only at the roots of the tree during the first m updates can be interpreted as providing a chance for the new demands to 'dig' through the network. This is why we required the satisfaction of the condition $m \geq$ (maximum number of hops), so information about new demands originating at the UpLinks will have a chance to reach the most-distanced possible DownLinks.

At clock $= xt_u$, where $x \leq m$, algorithm computes the expected demand on both CoSs by simply multiplying the total arrival rate (both accepted and rejected calls) and the end-to-end blocking probability for $r_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$; $v = 1, 2$, then adding the results (Eq. (20)). Note that, the total arrival rate for a specific flow is the same at all output ports visited by this flow.

At each output port related to any $r_{v,n}^{ij}$, the expected blocking rate is computed based on the measured blocking rate up to clock $= (x)(t_u)$ and the declared VPCBR at that output port at clock $= (x-1)t_u$ (Eq. (21)). Note that, in Eq. (21) the expected blocking rate at a specific output port is considered to be the VPCBR itself if the measured blocking rate were zero. The expected demand is then used to compute a new virtual bandwidth allocation Q_1, based on the desired objective function. Then, this new virtual allocation will be used to compute and declare the new VPCBR at clock $= (x)(t_u)$ at l^{ij}. The word 'declare' means that if any signaling packet moves through l^{ij} after clock $= xt_u$, it will carry this new VPCBR to other output ports.

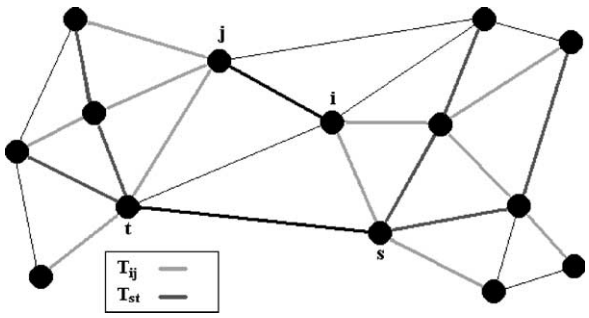

Fig. 2. A schematic view of a network that shows two trees only.

For clock $> (m)(t_u)$, the finalization stage starts. A serious problem must be faced in this stage, that is, which output ports will implement their final decisions first and in what order? By looking at the simple case shown in Fig. 2, one can recognize the dependency that all output ports' decisions have on each other. After careful study and intensive simulations, we reach the following protocols that need to be followed in the final stage.

Once the mth decision is declared, all DownLinks declare their semi-final-1 decisions, which are basically the mth decisions themselves at these DownLinks. The semi-final-1 decision is computed in a similar way to the decisions that are made during the first m updates, with a major difference that, at l^{ij}, only output ports related to B_v^{ij}, $v = 1, 2$, are considered in the computations. So, in the semi-final-1 decision, the algorithm looks at output ports after l^{ij}. This is why, at a DownLink, the semi-final-1 decision is the same as the last regular decision, since there are no output ports after a DownLink. In general, any output port l^{ij} needs to wait until the first output port of every $b_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$; $v = 1, 2$, declares its semi-final-1 VPCBR. Note that, when an output port computes the semi-final-1 VPCBR, the VPCBRs collected from B_v^{ij}, $v = 1, 2$, are the semi-final-1 VPCBRs. The declaration of semi-final-1 decisions may be viewed as if there is a decision wave propagating from downstream nodes to upstream nodes, we call this wave a semi-final-1 decision wave.

Once a DownLink (say l^{ij}) declares its semi-final-1 VPCBR, it waits for T_{pr} seconds (propagation time), which is the required/expected time for the semi-final-1 decision wave to reach all the UpLinks. This time depends on the maximum number of hops and on the signaling rate throughout the backbone network, which can be easily estimated. So, after t_{pr} seconds, all DownLinks compute and declare the semi-final-2 decisions which are computed similarly to the semi-final-1 decisions with a major difference that, at l^{ij}, all output ports related to T_v^{ij}, $v = 1, 2$, are considered in the computations. In general, any output port l^{ij} needs to wait until the first output port of every $b_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$; $v = 1, 2$, declares its semi-final-2 VPCBR. Note that, when l^{ij} computes the semi-final-2

VPCBR, the VPCBRs collected from B_v^{ij}, $v = 1, 2$, are the semi-final-2 VPCBRs, while the VPCBRs collected from R_v^{ij}, $v = 1, 2$, are the semi-final-1 VPCBRs. The declaration of semi-final-2 decisions may be viewed as if there is a decision wave propagating from downstream nodes to upstream nodes, we call this the semi-final-2 decision wave.

Once an UpLink (say l^{ij}) receives all semi-final-2 VPCBRs from the first output port of every $b_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$; $v = 1, 2$, it declares the final decision. In general, any output port l^{ij} needs to wait until the last output port of every $r_{v,n}^{ij}$, $n = 1, 2, \ldots, \sigma_v^{ij}$; $v = 1, 2$, declares its final VPCBR. The final VPCBRs are computed similarly to the semi-final-2 VPCBRs.

Note that, when l^{ij} computes the final VPCBR, the VPCBRs collected from B_v^{ij}, $v = 1, 2$, are the semi-final-2 VPCBRs, while the VPCBRs collected from R_v^{ij}, $v = 1, 2$, are the final VPCBRs. The declaration of final decisions may be viewed as if there is a decision wave propagating from upstream nodes to downstream nodes, we call this the final decision wave. At a synchronized instance, clock $= t_d$, all output ports physically implement the final decisions that were computed through the final decision wave. $(t_d - mt_u - t_r)$ should be greater than $3t_{\mathrm{pr}}$ to ensure that all output ports compute the final VPCBR.

So, the finalization stage is composed of three decision waves, after pumping virtual demands through the network for m update intervals, which is enough to ensure that the effects of new demands can reach the most-distanced node in the network. The first decision wave (semi-final-1) starts to provide the output ports with information about how downstream links responded to the offered virtual demands. The second and third decision waves, (semi-final-2) and (final), work as correction waves for some possible wrong information that might be generated by the first decision wave, as we explain later.

One may argue that information is being distributed by the signaling packets and not by independent flooding packets, and that may result in missing some information about specific flows if they are not active and do not have signaling packets during a specific t_u interval. Actually, this is what the VDD algorithm can perfectly overcome, since the expected demand is computed based on measurements of the average arrival rate of different flows and based on updated information about latest virtual decisions of all related output ports. One can easily realize that, if a specific flow is not active or completely dead through a specific t_u interval, then this means the contribution of this flow to the total expected demand is insignificant. Hence, even if the algorithm did not receive updated information about this path, the effect will be absolutely negligible.

5. Performance analysis

In this section, we present the results of testing the performance of the VDD algorithm. We compare the performance of the algorithm with a SD algorithm that uses locally estimated demands. So, both VDD and SDD algorithm utilize Algorithm 2. However, SD estimates average arrival rate of both CoSs using local measurements, while VDD utilizes signaling messages in a coordinated mechanism to gain wider vision and higher accuracy in estimating average arrival rate of both CoSs every decision interval.

5.1. Simulation settings

Fig. 1 shows configuration of the simulated network. The sophisticated simulation tool (Bones Designer) was used to build a two-class multi-node network. This network was built to emulate a real backbone network. Each node represents a router; each router is connected to a LAN—not shown in the figure—via a GateWay. Each LAN generates a stream of calls (sessions) based on a Poisson random process. Traffic generated at a specific LAN is distributed on all possible destinations forming a group of flows based on fixed routing (shortest path). The average arrival rate of sessions under each flow varies slowly, following a sinusoidal shape with a period of 48 h. The phase of this sinusoid is different from one flow to another. Another slower sinusoidal function with a period of 196 h was used to classify sessions into 1st class and 2nd class under each flow at each LAN, such that the maximum of the sinusoid corresponds to 10% of the sessions being 1st class and 90% being 2nd class. At the minimum of sinusoid, the opposite occurs. Thus, a non-stationary traffic/demand is generated for a period of 48 h.

Average arrival rate of different flows was set to congest the network, in order to test the performance of different algorithms. The requested bandwidth was generated following a Binomial distribution with $\bar{b}^1 = \bar{b}^2 = 1.0$ Mbps. The requested bandwidth was quantized with a step of 100 Kbps. Minimum requested bandwidth was limited to 100 Kbps and the maximum to 10 Mbps. Thus, $M_1 = M_2 = 100$. Holding time was modeled as an exponential distribution with $S_1^{-1} = S_2^{-1} = 10$ min. The links' bandwidth, $W_t = 3.0$ Gbps.

For the VDD algorithm, we used the following settings $t_d = 20$ min, $t_u = 2$ min, $t_r = 4$ min, $t_{\mathrm{pr}} = 1$ min, and $m = 6$. For the SD algorithm, the decision interval was set to be equal to t_d, in order to make the comparison valid. The SD algorithm counts the number of 1st and 2nd CoSs calls arriving throughout the t_d interval, in order to estimate the average arrival rate of both CoSs.

As we mentioned earlier, the goal of VDD algorithm is to provide accurate predictions of demands of different CoSs. It is up to the network designer to choose the objective function to be optimized. For this investigation, we chose a simple objective function to demonstrate how well the algorithm can perform. So the VDD algorithm is not the DBA itself, but rather, it is a tool to provide accurate

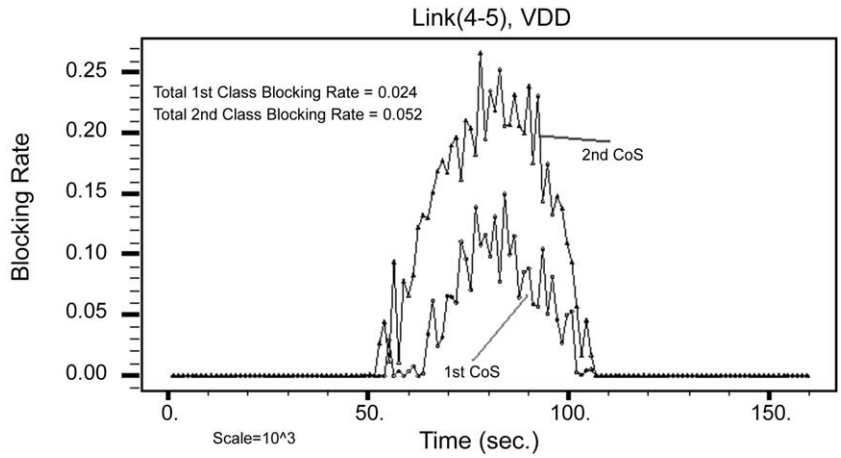

Fig. 3. Blocking rates under the VDD at link 4-5.

prediction of demands that can be used by DBA algorithms. In this simulation, we simply chose $\beta = 2$.

5.2. Simulation results

Since our objective is to force the ratio of the 2nd class blocking rate to the 1st class blocking rate to be two at any output port ($\beta = 2$), we selected l^{45} to be monitored, since it is in the middle of the network. Fig. 3 shows the blocking rate of both the 1st and 2nd CoS versus time at l^{45} for the VDD algorithm, Fig. 4 shows the performance of SD algorithm. It is clear that the VDD algorithm outperforms the SD algorithm, proving the wider vision that VDD enjoys. Fig. 4 shows the VDD approximately maintaining $\beta = 2$ almost all the time, while SD deviated from the value $\beta = 2$ most of the time. Fig. 5 shows utilization and bandwidth allocation decisions for the VDD algorithm at l^{45}, it can be noticed how the algorithm shapes the utilization of both CoSs gradually to enforce the ratio $\beta = 2$.

To clarify the importance of global vision, Fig. 6 shows utilizations and bandwidth allocation decisions for the VDD algorithm at l^{57}. It is interesting to compare this figure to Fig. 7, which shows the equivalent performance of the SD algorithm. Note, how the bandwidth allocation decisions in the SD algorithm mistakenly congested the 2nd CoS during a specific time interval while there are free resources available, as can be noticed by comparing the 1st class utilization to the allocated bandwidth for the 1st CoS. This can happen simply because the SD algorithm cannot estimate how many of the arriving calls are being accepted by the DownLinks. However, the VDD algorithm is able to evenly distribute the extra resources.

In another experiment, we broke link l^{56} at a specific instance and intentionally increased the percentage of 1st class calls generated at node 1 and moved through l^{56} to make the effect of breaking l^{56} more recognizable. Fig. 8 shows the performance of the VDD algorithm while Fig. 9 shows the performance of the SD algorithm. Again, we can see how the SD algorithm failed to notice the effect of breaking l^{56}, while the VDD algorithm perfectly estimated the effect of this event and allocated resources, accordingly.

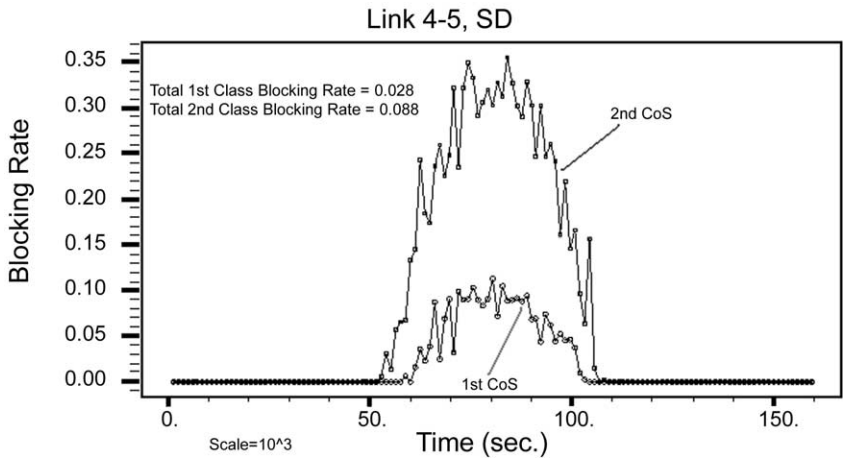

Fig. 4. Blocking rates under the SD at link 4-5.

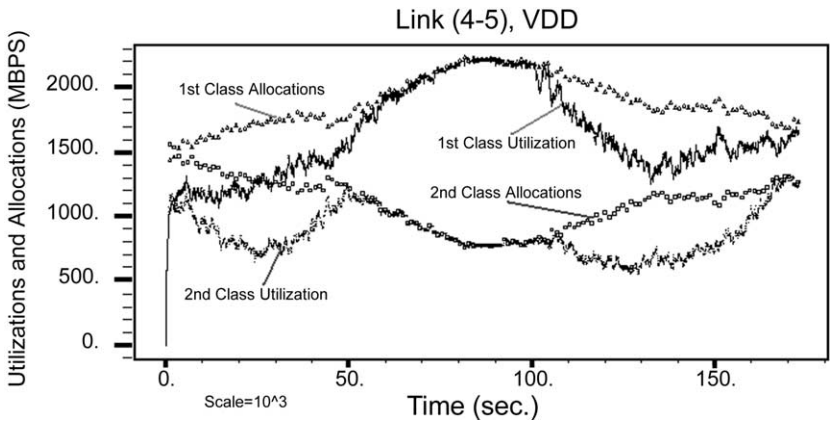

Fig. 5. Utilizations and allocations under the VDD algorithm at link 4-5.

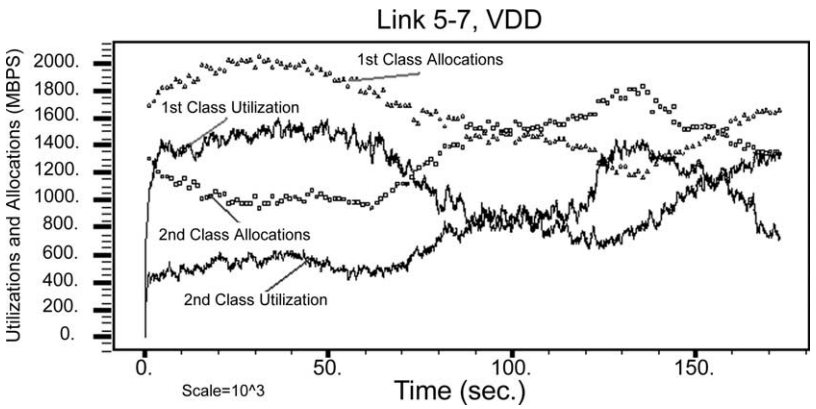

Fig. 6. Utilizations and allocations under the VDD algorithm at link 5-7.

5.3. Tradeoffs regarding the VDD algorithm

The most important feature of VDD algorithm is its ability to have a wider vision when implementing a bandwidth allocation decision. VDD algorithm does not require extra signaling packets, which is a very desirable criterion. It can be considered as a robust algorithm, since it is not affected by failures in some parts of the network, and it will always try to provide the best prediction of demands based on the most recent available information. Dynamic resource allocation based on prediction of demands has been proposed in many previous works, the work in Ref. [11,12] is an example of that.

As the Internet increasingly becomes the largest and the most complicated machine in the world, and as the world converges into one community, designing and policing

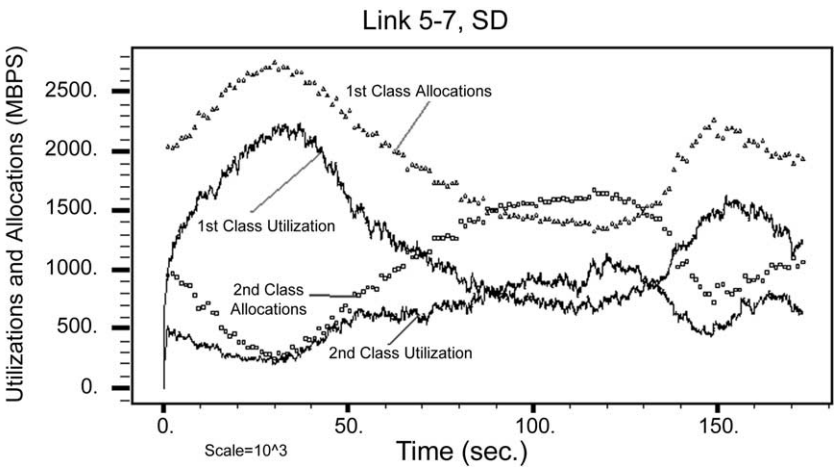

Fig. 7. Utilizations and allocations under the SD algorithm at link 5-7.

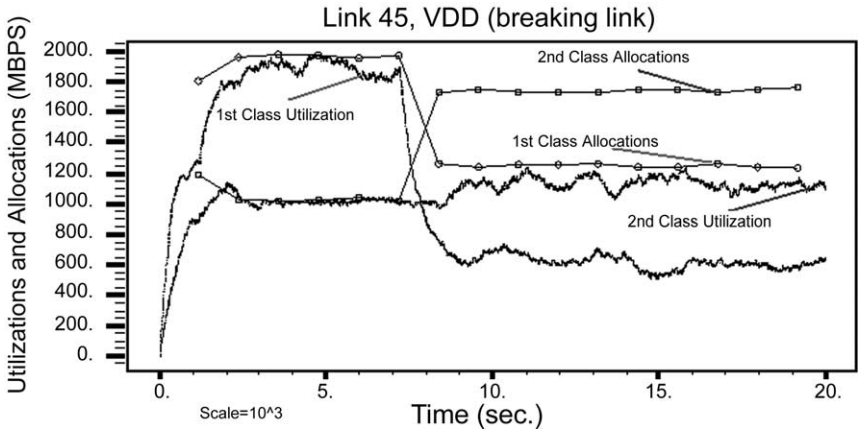

Fig. 8. Performance of the VDD when link 5-6 was broken.

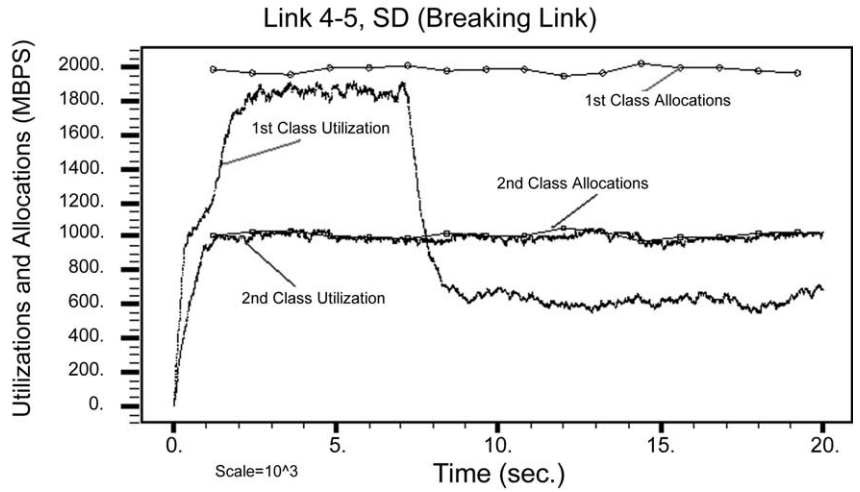

Fig. 9. Performance of the SD algorithm when link 5-6 was broken.

the network based on local geographical information will no longer provide adequate results. The VDD algorithm collects information about demands and congestion from relevant flows that can start and end at any location in the world, thus avoiding this problem.

On the other hand, VDD algorithm adds more overhead on the signaling packets and requires software upgrades at all backbone routers where this algorithm would be implemented. But this overhead is acceptable or even lower than what was proposed in some previous works. Like the frameworks proposed in Ref. [13,14], where negotiation protocols with much more overhead than what we proposed in this work were proposed. When the network is underutilized, the VDD algorithm could be considered as an over-engineered approach. The complexity of optimizing an objective function may become high if the network supports many CoSs, however, the complexity involved in the mechanism of VDD would not increase significantly as the number of CoSs or VPs increases.

6. Conclusion

The importance of accurate predictions of demands in a multi-class multi-nodes connection-oriented network with bandwidth segregation led us to develop a novel bandwidth allocation algorithm, the VDD. Simulations showed that the VDD outperforms an equivalent SD algorithm. Results under non-stationary two-class traffic with Poisson arrivals in a multi-node network showed that VDD was able to near-optimally allocate resources to the two CoSs based on the selected objective function. Although we were not able to mathematically prove that the VDD algorithm can produce globally optimized decisions, we showed by network analysis the existence of an algorithm that can issue globally optimized decisions while operating in a distributed manner, by collecting extra information about the network. Such information still would be much less than the information needed by a centralized algorithm. The potential of the VDD algorithm to be used as a benchmark in large network analysis increases its importance.

Toward the development of the VDD algorithm, we analyzed the prediction of call blocking rates in a single-link adopting a CP scheme with two CoSs of non-stationary traffic, and developed a recursive algorithm to allocate bandwidth between the two CoSs with the objective of enforcing a specific ratio between the call blocking rates of the two CoSs. This recursive algorithm considered the blocking transient effects that may be experienced by calls in a CP scheme. Then, the mathematical analysis was extended to multi-class multi-node connection-oriented networks adopting CP scheme under non-stationary traffic conditions. Then the VDD algorithm was motivated by this loss networks analysis and evaluated by extensive simulations.

Appendix A

In this appendix, we review previous work on predicting call blocking rate experienced by calls with stationary Poisson arrival process, stationary exponential holding time, and variable-size requests with known distribution. Such systems are known as stochastic Knapsack systems. We show that the recursive algorithm (Algorithm 1), which is detailed in Section 2.1, is the most suitable to our needs in this research. The formulas and algorithms in this section were derived in a clear way in Ref. [8].

Let the requested bandwidth be a positive real value (b) with a density function $f_b(b)$, B_p: blocking probability, λ: mean arrival rate, μ: mean service rate, and W_t: total capacity. A closed form formula for B_p was found when $f_b(b)$ is uniform and $0 \le b \le W_t$. The probability of blocking a request of size b is:

$$B_p(b) = 1 - \frac{J_0(2\sqrt{\rho(W_t - b)/W_t})}{J_0(2\sqrt{\rho})} \tag{A1}$$

where $\rho = \lambda/\mu$, $J_0(\cdot)$ is the modified Bessel function of order 0. Thus, we can write:

$$B_p = \frac{1}{W_t}\int_0^{W_t} B_p(b)\mathrm{d}b. \tag{A2}$$

The problem in Eq. (A1) is the assumption that $f_b(b)$ is uniform over $0 \le b \le W_t$. In any real network, usually the maximum possible value for b is a small fraction of W_t. In addition, the distribution of requested bandwidth is far from uniform.

The other alternative is to consider a limited number of bandwidth sizes, such that a size-k session requests b_k units of bandwidth with a probability of $P_b(b_k)$. In the context of our analysis, we have $\lambda_k = \lambda P_b(b_k)$ and $\mu_k = \mu$, $\forall k$. Let $\vec{b} = [b_1 b_2 \ldots b_M]$, $\vec{n} = [n_1 n_2 \ldots n_M]$, where n_i is the number of size-i active calls in the link. An exact expression was found, not closed form though, for the probability of blocking a size-k call ($B_p(b_k)$):

$$B_p(b_k) = 1 - \frac{\sum_{\vec{n}\in S_k} \prod_{j=1}^{M} \rho_j^{n_j}/n_j!}{\sum_{\vec{n}\in S} \prod_{j=1}^{M} \rho_j^{n_j}/n_j!}, \tag{A3}$$

where $S_k = \{\vec{n} \in S : \vec{b}\cdot\vec{n} \le W_t - b_k\}$ and $S = \{\vec{n} \in I^M : \vec{b}\cdot\vec{n} \le W_t\}$. I is the set of non-negative integers. So, we can write:

$$B_p = \sum_{k=1}^{M} P_b(b_k) B_p(b_k). \tag{A4}$$

The expression in Eq. (A3) cannot be computed easily for reasonable values of M and W_t. A recursive algorithm exits to compute the expression in Eq. (A3). It was shown in Ref. [8] that this recursive algorithm works well for large values of M and W_t. This is important, because in a real backbone we expect large values of W_t for sure. This recursive algorithm is detailed in Section 2.1. The formula in Eq. (A3) gives an exact answer for the blocking rates for very reasonable model assumptions that suite very well the problem under study in this work. Thus, the recursive algorithm that solves this formula was selected to be a building block in the novel algorithm (VDD) proposed in this paper.

One more technique that needs to be explored is the asymptotic approximation. This approximation becomes more accurate as W_t increases, which is a desired behavior in our analysis. However, unfortunately, it seems no closed form formula can be derived from this approximation. The following equations constitute the approximation:

$$B_p(b_k) = b_k \frac{\delta}{W_t}, \tag{A5}$$

$$\delta = \frac{e^{\frac{-(\alpha/\sigma)}{2\sigma^2}}}{\sigma\int_{-\infty}^{\alpha/\sigma} e^{\frac{-x^2}{2\sigma^2}}\mathrm{d}x}, \tag{A6}$$

$$\sigma^2 = \frac{\sum_{k=1}^{M} b_k^2 \rho_k}{W_t}, \tag{A7}$$

$$\alpha = \sqrt{W_t} - \frac{\sum_{k=1}^{M} b_k \rho_k}{\sqrt{W_t}}. \tag{A8}$$

Another problem in this approximation is that it gives accurate results as long as $|\alpha| \le 1.0$. If this approximation is to be used in a practical situation, $|\alpha|$ can reach high values, especially when the demand is higher than total serving power. Our simulations showed that the approximation is inaccurate, at all, for large values of $|\alpha|$.

As we can see, a closed-form formula that can provide us with the expected blocking rate in the case of variable size bandwidth requests is not available, neither in an exact nor in an approximate form. Thus, based on the above review of previous work, it is clear that the recursive algorithm, Algorithm 1 in Section 2.1, is the best approach to build upon, in order to find an approach for bandwidth allocation between CoSs with the objective of forcing specific differentiation ratios between the call blocking rates of these different CoSs.

Appendix B

Proof of Claim 1 (stated in Section 2). In the following we mean by an output port related to a tree that there is a flow member of this tree such that this output port is a member of this flow, by a decision a bandwidth allocation decision. Consider any output port l^{ij} in a connection-oriented network that supports two CoSs.

(1) Let l^{st} be any output port such that $l^{st} \neq l^{ij}$ where $s, t, i, j \in N$, $s \neq t$, $i \neq j$.
(2) Assume there is at least one $f_{v,k}^{ij} \in T_v^{ij}$ such that $l^{st} \in f_{v,k}^{ij}$. Then, the effect that $T^r(T_v^{st})$ may have on the decision at l^{ij} is implied in the state/information at l^{st}, where $r = 0, 1, 2, \ldots, \infty$.
(3) If there is no $f_{v,k}^{ij} \in T_v^{ij}$ such that $l^{st} \in f_{v,k}^{ij}$, then state/information at l^{st} either has no effect on the decision at l^{ij} or have an indirect effect if and only if there exists a $T^r(T_v^{ij})$ such that there exists a tree $\in T^r(T_v^{ij})$ such that l^{st} is related to this tree, where $r = 1, 2, \ldots, \infty$. Note r starts from 1 and not from 0.
(4) For the state/allocation at l^{st} to have no effect on the decision at l^{ij} then $\forall v$, $v = 1, 2$; $\forall r$, $r = 0, 1, 2, \ldots, \infty$, there is no $T^r(T_v^{ij})$ such that l^{st} is related to any tree $\in T^r(T_v^{ij})$. This scenario is shown in Fig. 2.
(5) So, there should exist an algorithm that only needs to know $a(f_{v,n}^{ij})$ $\forall f_{v,n}^{ij} \in T_v^{ij}$, $\forall n$, $n = 1, 2, \ldots, \sigma_v^{ij}$; $\forall v$, $v = 1, 2$ and the state/information at each output port (say l^{xy}) related to any flow $\in (T_1^{ij}$ or $T_2^{ij})$, in order to issue an optimal decision at l^{ij}. Note that, the state/information at each output port l^{xy} is in itself a function of all $f_{v,n}^{xy} \in T_v^{xy}$ and state/information at each port related to any flow $\in (T_1^{xy}$ or $T_2^{xy})$. □

The state/information at an output port is a general term that, in the worst case, can be an information package containing the demands of all flows composing the tree at that output port. Note that, when the state/information at an output port is designed for the worst case, we are actually implementing a centralized algorithm with one improvement, that demands on irrelevant flows are not considered by the algorithm. In the best case, state/information can be brief information about some parameters of interest like utilization and blocking rates of the two CoSs. Worst case and best case here are in terms of complexity.

References

[1] J. Chan, D. Tsang, Comparison of static and dynamic bandwidth allocation schemes for multiple QoS classes in ATM network, Singapore ICCS '94 Proceedings 2 (1994) 732–782.
[2] A. Maunder, P. Min, Dynamic bandwidth allocation to virtual paths IPCCC '98 IEEE International (1998) 94–100.
[3] E. Rosen, A. Viswanathan, R. Callon, Multi-protocol label switching architecture, IETF (2001) RFC 3031.
[4] K. Kar, M. Kodialam, T.V. Lakshman, Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications, IEEE JSAC (2000) 2566–2579.
[5] E. Dinan, D.O. Awduche, B. Jabbari, Analytical framework for dynamic traffic partitioning in MPLS networks, ICC 2000 3 (2000) 1604–1608.
[6] R. Cue'rin, H. Ahmadi, M. Naghshineh, Equivalent capacity and its application to bandwidth allocation in high-speed networks, IEEE JSAC 9 (7) (1991) 968–981.
[7] I.C. Paschalidis, J.N. Tsitsiklis, Congestion-dependent pricing of network services, IEEE/ACM Transactions on Networking 8 (2) (2000) 171–184.
[8] K.W. Ross, Multiservice Loss Models for Broadband Telecommunication Networks, Springer, London, 1995.
[9] A. Odlyzko, The economics of the internet: utility, utilization, pricing, and quality of service, AT&T Labs-Research (1998).
[10] W. Stallings, Data and Computer Communications, Fifth ed., Prentice-Hall, Englewood Cliffs, NJ, 1997.
[11] S. Gupta, S. Bose, R. Harris, L. Berry, Distributed dynamic bandwidth allocation and management for self-healing broadband networks with multi-class traffic, IEEE GLOBECOM'98 2 (1998) 1166–1171.
[12] H. Saito, Trials of dynamic bandwidth allocation in ATM networks, Proceedings of IEEE ATM Workshop, 1997, pp. 141–146.
[13] E.W. Fulp, D.S. Reeves, On-line dynamic bandwidth allocation, Proceedings of the International Conference on Network Protocols, 1997, pp. 143–141.
[14] S. Jordan, H. Jiang, Connection establishment in high-speed networks, IEEE JSAC 13 (7) (1995) 1150–1161.