國立交通大學

電信工程學系

碩 士 論 文

以局部收集的網路狀態為依據之服務品質繞徑法

Localized State-Dependent Control in QoS Routing

研 究 生：顏廷任

指導教授：廖維國　教授

中 華 民 國 九 十 五 年 十二 月

以局部收集的網路狀態為依據之服務品質繞徑法

Localized State-Dependent Control in QoS Routing

研 究 生：顏廷任　　Student：Ting-Jen Yen

指導教授：廖維國　　Advisor：Wei-Kuo Liao

國 立 交 通 大 學

電信工程學系

碩 士 論 文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Communication Engineering

December 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十二月

以局部收集的網路狀態為依據之服務品質繞徑法

研究生：顏廷任　　　　　　　　　　指導教授：廖維國 博士

國立交通大學電信工程學系

摘要

我們研究探討一種服務品質繞徑(QoS routing)的問題，其中網路狀態的資訊只是附加包含於原有的資源保留協定(RSVP)中的RESV信息，而沒有使用另外的繞徑協定。如此的作法是令人滿意的，因為它完全排除了在服務品質繞徑協定裡由於時常更新網路狀態而造成的運算負擔。基本上，我們所提出的繞徑法，也就是所謂的”局部性依據網路狀態繞徑法”(localized state-dependent routing)，促進改善了”局部性比例分配繞徑法”(localized proportional routing)以及藉由離散時間馬可夫決策過程所模擬出來的”依據網路狀態可分離式繞徑法”(state-dependent separable routing)。我們的方法是這樣的：假如網路狀態的資訊仍然是即時的，那我們就採用這個資訊；相反的，我們則採用平均頻寬資訊。隨著增加”依據網路狀態可分離式繞徑法”使用上的比重，我們發現在我們所設計的模擬情境中，由我們所提出的繞徑法的效能表現越來越好。除此之外，從我們模擬的結果可以看出，我們提出的繞徑法表現的比”局部性依據網路狀態繞徑法”還要好。而且當網路流量沒有很繁重的時候，我們所提出的繞徑法的效能甚至接近最理想的效能。

Localized State-Dependent Control in QoS Routing

Student: Ting-Jen Yen Advisor: Dr. Wei-Kuo Liao

Department of Communication Engineering
National Chiao Tung University

Abstract

We study the QoS routing problem where the network state information is only piggybacked in the RESV messages in RSVP (Resource Reservation Protocol) without using any other routing protocol. Doing so is desirable because it completely eliminates the communication overheads entailed by frequent state updates by QoS routing protocol. Basically, our proposed routing called localized state-dependent routing boosts the localized proportional routing and the state-dependent separable routing formulated by Discrete-Time Markov Decision Process in such a way that the link state information will be explored if it is fresh enough; otherwise it remains to use the average bandwidth information. By increasing the configurable weight on the state-dependent separable routing, we find the performance of our proposed routing eventually getting better in our selected simulation scenario. Besides, as the simulation results shown, our proposed routing outperforms the localized proportional routing and even is nearly optimal when the load is not heavy.

誌謝

首先，感謝我的父母，從小到大提供我一個良好的求學環境，並且提供我生活上的幫助，讓我可以無後顧之憂的唸書做學問。另外感謝我的指導教授廖維國老師，在這兩年多來，在課業及學術方面給我很好的教導，讓我的學習到很多做學問的知識和方法，受益良多。同時也感謝兩位口試委員，張仲儒教授以及魏學良教授，對我口試時的建議和建言，對於我的論文有很大幫助。此外，也感謝我的實驗室同學、學長姊、學弟妹，平常對我的照顧和幫忙，也讓我生活添加了很多樂趣。最後要感謝我的女朋友媛茜，從我大學到碩士，一路上都一直默默的幫助我和支持我，是我最大的精神糧食。

Contents

List of Tables

List of Figures

Chapter 1
Introduction

For a large-scale network, controlling the load to each link is mandatory to meet the requirements of each connection. Without appropriate load control on links, the load may exceed the link capacity and end user will experience quality degradation, such as delay jitter, unexpected dropping packets, etc.

To this end, the IntServ (Integrated Service) working group defined the RSVP (Resource Reservation Protocol) [7] to enable the reservation-based load control. Consider the case of one sender and one receiver trying to get a reservation for traffic flowing between them. By sending a PATH message from the sender to the receiver that contains the flow's traffic characteristics (i.e. the sender's TSpec), the receiver can establish a resource reservation at each router on that path. Each router looks this PATH message as it goes past, and it figures out the reverse path. Having received a PATH message, the receiver sends back a RESV message for resource reservation along the reverse path. This message contains the sender's TSpec and an RSpec describing the requirements of this receiver. Each router on the path looks at the reservation request and tries to allocate the necessary resources to satisfy it. If the reservation can be made, the RESV request is passed on to the next router. If not, an error message is returned to the receiver who made the request. If all goes well, the correct reservation is installed at every router between the sender and the receiver.

The issue of how to direct the PATH message to reduce the chance of declined flow request is addressed in the *QoS routing*. Among many alternatives, the source-routing-based QoS routing has been extensively studied. Essentially, the source selects a "best path" for flow according to the QoS requirement of the flow and the knowledge of the resource availability at network nodes. The source-routing-based QoS routing can be categorized by the way that they gather information about the network state and select a path by this information. For instance, in the state-dependent routing approach [2] each node can construct a global view of the network QoS state information through periodic information exchange among nodes in a network and selects the best path for a flow based on this global view of the network state. Examples of state-dependent routing approach are those QoS routing schemes [9, 10] based on QoS extensions to the OSPF (Open Shortest Path First) routing protocol.

However, for a large-scale network, the prohibitive communication overheads entailed by such frequent state updates precludes the possibility of always maintaining an accurate view of the current network QoS state for each node. To handle such scalability problem, in [4] the authors proposed a method called *localized proportional routing*, where feasible path are selected based on the average available bandwidth information and flows are adaptively proportioned among these feasible path based on locally collected information from other nodes. Indeed, by collecting the information which is piggybacked in the RESV messages, there is no need for another protocol to collect the network state information. As shown in their simulation results, by adaptively proportioning flows the localized proportional routing approach performs well than the state-dependent routing approach when the update interval is long enough, typically more than one minute.

In viewing that the state-dependent routing with short update interval still outperforms the localized proportional routing, we proposed the routing scheme called *localized state-dependent routing*. Briefly put, our proposed routing combines the localized proportional routing scheme with the state-dependent separable routing formulated by Discrete-Time Markov Decision Process in a way that the state information will be explored if it is fresh enough; otherwise it remains to use the average bandwidth information. By sending RESV message which additionally contains the state information back to source, source obtains the local network information feedback from the routers, and therefore we do not increase the communication overhead. From the simulation results of our selected system model, our proposed routing outperforms the localized proportional routing.

The remainder of this thesis is organized as follows: in chapter 2, we discuss the necessary background knowledge, including Discrete-Time Markov Decision Process [1], Resource Reservation Protocol (RSVP) [7], State-Dependent Routing [2], and Adaptive Proportional Routing [3, 4]. In chapter 3, we describe the proposed “localized state-dependent routing” to minimize the overall blocking probability by its self-refrained alternative routing with the localized view of the network state. The simulator design and the simulation results are shown in chapter 4. Finally, we make a conclusion in chapter 5.

Chapter 2
Background

In this chapter, we will introduce the basic concept of Discrete-Time Markov Process [1] and RSVP (Resource Reservation Protocol) [7]. And then we introduce two kinds of routing schemes, state-dependent routing [2] and adaptive proportional routing [3, 4].

2.1 Discrete-Time Markov Decision Process

Suppose that in the system there are ***N*** states numbered from 1 to ***N***. If the system now occupies state *i*, the probability of a transition to state *j* during the next constant time interval is a function only of *i* and *j* and not any history of the system before its arrival in *i*. In other word, we may specify a set of conditional probability p_{ij} that a system which now occupies state *i* will occupy state *j* after its next transition. Since the system must be in some state after its next transition, $\sum_{j=1}^{N} p_{ij} = 1$, where the probability that the system will remain in *i*, p_{ii}, has been included. Since p_{ij} is probability, $0 \leqq p_{ij} \leqq 1$. The set of transition probabilities for the process may be described by a transition probability matrix **P** with elements p_{ij}.

Suppose that an ***N***-state Markov process earns r_{ij} if it makes a transition from state *i* to state *j*. We call r_{ij} the "reward" associated with the transition from *i* to *j*. The set of rewards for the process may be described by a reward matrix **R** with elements r_{ij}. The Markov process generates a sequence of rewards as it makes transitions from state to state. The reward is thus a random variable with a probability distribution governed by the probabilistic relations of the Markov process.

Now we define $v_i(n)$ as the expected total rewards in the next *n* transitions if the system is in state *i* now. Some reflection on this definition allows us to write the recurrent relation,

$$v_i(n) = \sum_{j=1}^{N} p_{ij}[r_{ij} + v_j(n-1)] \qquad i = 1, 2, \Lambda\ , N \qquad n = 1, 2, 3, \Lambda \tag{2.1}$$

If the system makes a transition from *i* to *j*, it will earn the reward r_{ij} plus the amount it expects to earn if it starts in state *j* with one move fewer remaining. As shown in Eq. (2.1), these rewards from *i* to *j* must be weighted by the probability of such a transition, p_{ij}, to obtain the expected total rewards.

Notice that Eq. (2.1) may be also written in the form

$$v_i(n) = q_i + \sum_{j=1}^{N} p_{ij} v_j(n-1) \qquad i = 1, 2, \Lambda\ , N \qquad n = 1, 2, 3, \Lambda \tag{2.2}$$

, where the quantity q_i is defined by

$$q_i = \sum_{j=1}^{N} p_{ij} r_{ij} \qquad i = 1, 2, \Lambda\ , N \tag{2.3}$$

The quantity q_i may be defined as the reward to be expected in the next transition out of state *i*. It will be called the "expected immediate reward" for state *i*. Rewriting Eq. (2.1) as Eq. (2.2) shows us that it is not necessary to specify both a **P** matrix and an **R** matrix in order to determine the expected earnings of the system. All that is needed is a **P** matrix and a **q** column vector with N components q_i. In vector form, Eq. (2.2) may be written as

$$\mathbf{v}(n) = \mathbf{q} + \mathbf{P}\mathbf{v}(n-1) \qquad n = 1, 2, 3, \Lambda \tag{2.4}$$

, where $\mathbf{v}(n)$ is a column vector with ***N*** components $\mathrm{v}_i(n)$, called the total-value vector.

If there is only one recurrent chain in the system so that it is completely ergodic. Consider a completely ergodic ***N***-state Markov process described by a transition-probability matrix **P** and a reward matrix **R**. Suppose that the process is allowed to make transitions for a very, very long time and that we are interested in the earnings of the process. The total expected earnings (rewards or costs) depend upon the total number of transitions that the system undergoes, so that this quantity grows without limit as the number of transitions increases. A more useful quantity is the average earnings of the process per unit time. This quantity is meaningful if the process is allowed to make many transitions; it was called the "gain" of the process.

Since the system is completely ergodic, the limiting state probabilities π_i are independent of the starting state, and the gain *g* of the system is

$$g = \sum_{i=1}^{N} \pi_i q_i \tag{2.5}$$

, where q_i is the expected immediate return state *i* defined by Eq. (2.3).

Consider the three-dimensional array of Fig. 1, which presents in graphical form the states and alternatives.

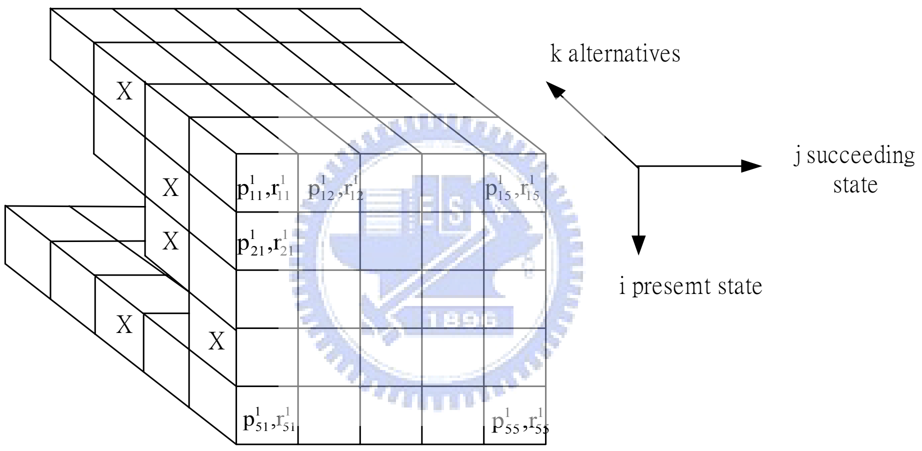


Fig. 1 A possible five-state problem

The array as drawn illustrates a five-state problem that has four alternatives in the first state, three in the second, two in the third, one in the fourth, and five in the fifth. Entered on the face of the array are the parameters for the first alternative in each state, the second alternative in each state, and so forth. An **X** indicates that we have chosen a particular alternative in a state with a probability and reward distribution that will govern the behavior of the system at any time that it enters that state. Thus the alternative selected is called the “decision” for that state. The set of decisions for all states is called a “policy”. Selection of a policy thus determines the Markov process with rewards that will describe the operation of the system.

An optimal policy is defined as a policy that maximizes the gain, or average return per transition. We can find the optimal policy in a small number of iterations by the policy-iteration method. It is composed of two parts, the value-determination operation and the policy-improvement routine which are diagrammed as shown in Fig.2

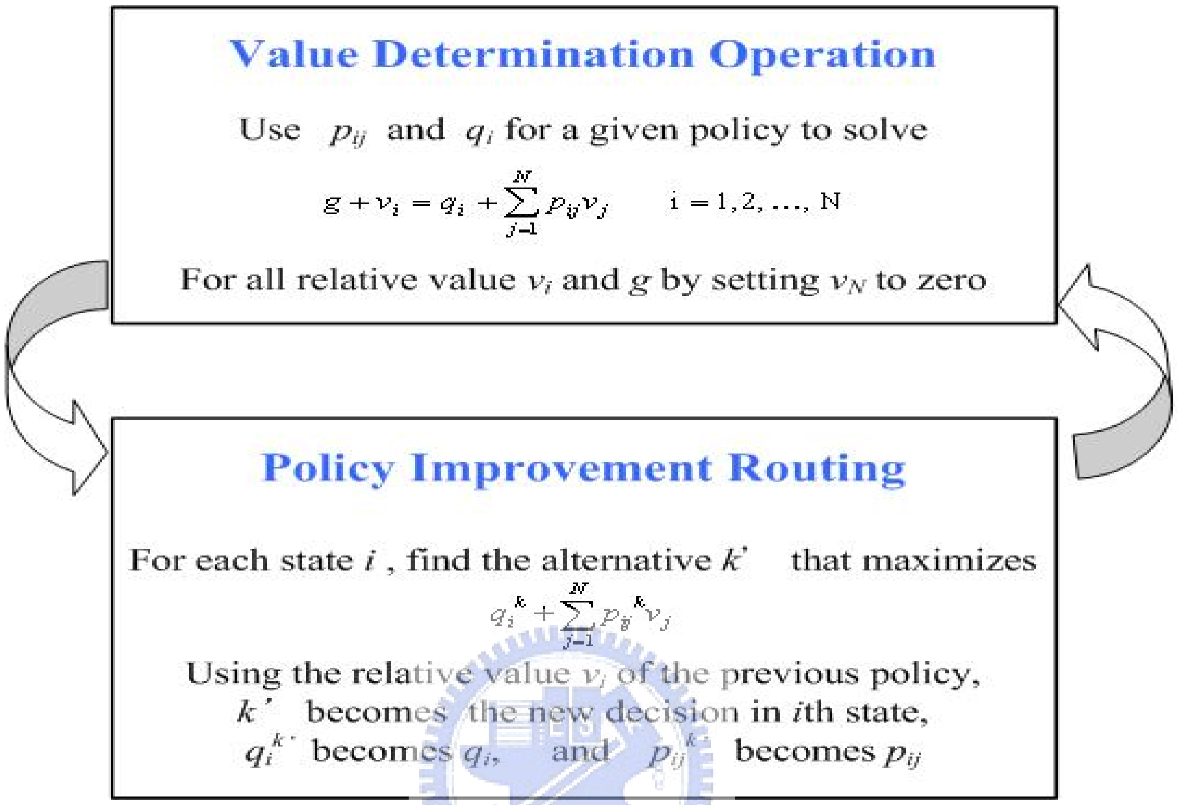

Value Determination Operation
Use p_{ij} and q_i for a given policy to solve
$g + v_i = q_i + \sum_{j=1}^{N} p_{ij} v_j \quad i = 1, 2, \ldots, N$
For all relative value v_i and g by setting v_N to zero
Policy Improvement Routing
For each state i, find the alternative k' that maximizes
$q_i^k + \sum_{j=1}^{N} p_{ij}^k v_j$
Using the relative value v_i of the previous policy,
k' becomes the new decision in ith state,
$q_i^{k'}$ becomes q_i, and $p_{ij}^{k'}$ becomes p_{ij}

Fig. 2 Policy Iteration Cycle

In [1] it proves that the new policy will have a higher gain than the old policy. First, however, we shall show how the value-determination operation and the policy-improvement routine are combined in an iteration cycle whose objective is to find a policy that has highest gain among all possible policies. The upper box, the value-determination operation, yields the g and v_i corresponding to a given choice of p_{ij} and q_i. The lower box yields p_{ij} and q_i that increase the gain for a given set of v_i. In other words, the value-determination operation yields values as a function of policy, whereas the policy-improvement routine yields the policy as a function of the values.

We may enter the iteration cycle in either box. If the value-determination operation is chosen as the entrance point, an initial policy must be selected. If the cycle is to start in the policy-improvement routine, then a starting set of values is necessary. The selection of an initial policy that maximizes expected immediate reward is quite satisfactory in the majority of cases.

At this point it would be wise to say a few words about how to stop the iteration cycle once it has done its job. The rule is quite simple: The final robust policy has been reached (g is maximized) when the policies on two successive iterations are identical. In order to prevent the policy-improvement routine from quibbling over equally good alternatives in a particular state, it is only necessary to require that the old decision d_i be left unchanged if the test quantity for that d_i is as large as that of any other alternative in the new policy determination.

In summary, the policy-iteration method just described has the following three properties:

a. The solution of the sequential decision process is reduced to solving sets of linear simultaneous equations and subsequent comparisons.
b. Each succeeding policy found in the iteration cycle has a higher gain than the previous one.
c. The iteration cycle will terminate on the policy that has largest gain attainable within the realm of the problem; it will usually find this policy in a small number of iterations.

These three properties are proved in detail in [1].

2.2 RSVP (Resource Reservation Protocol)

The term "Integrated Service" (often called IntServ fo short) refers to a body of work that was produced by the IETF around 1995-97. The IntServ working group developed specifications of a number of service classes (e.g. guaranteed service and controlled load service) designed to meet the need of some the application types. It also defined how RSVP (Resource Reservation Protocol) could be used to make reservation using these service classes.

With a best-effort service we can just tell the network where we want our packets to go and leave it at that, but a real-time service involves telling the network something more about the type of service we require. In addition to describing what we want, we need to tell the network something about what we are going to inject into it, since a low-bandwidth application is going to require fewer network resources than a high-bandwidth application. The set of information that we provide to the network is referred to as a flowspec. There are two separable parts to the flowspec: the part that describes the flow's traffic characteristics (called the TSpec) and the part that describes the service requested from the network (called the RSpec).

When we ask the network to provide us with a particular service, the network needs to decide if it can in fact provide that service. The process of deciding when to say no is called admission control. When some new flow wants to receive a particular level of service, admission control looks at the TSpec and RSpec of the flow and tries to decide if the desired service can be provided to that amount of traffic, given the currently available resources, without causing any previously admitted flow to receive worse service than it had requested. More knowledge about admission control could be found in [7].

One of the key assumptions underlying RSVP is that it should not detract from the robustness that we fid in today's connectionless network. RSVP tries to maintain this robustness by using the idea of soft state in the routers. In contrast to the hard state found in connection-oriented networks, soft state doesn't need to be explicitly deleted when it is no longer needed. Instead, it times out after some fairly short period if it is not periodically refreshed. Another important characteristic of RSVP is that it aims to support multicast flows just as effectively as unicast flows. For multicast applications, rather than having the senders keep track of a potentially large number of receivers, it makes more sense to let the receivers keep tack of their own needs.

The soft state and receiver-oriented nature of RSVP give it a number of nice properties. One nice property is that it is very straightforward to increase or decrease the level of resource allocation provided to a receiver. Since each receiver periodically sends refresh messages to keep the soft state in place, it is easy to send a new reservation that asks for a new level of resources. In the event of a host crash, resources allocated by that host to a flow will naturally time out and be released. Now we look a little more closely at the mechanics of making a reservation.

Initially, consider the case of one sender and one receiver trying to get a reservation for traffic flowing between them. There are two things that need to happen before a receiver can make a reservation. First, the receiver needs to know what traffic the sender is likely to send so that it can make an appropriate reservation. That is, it needs to know the sender's TSpec. Second, it needs to know what path the packets will follow from sender to receiver, so that it can establish a resource reservation at each router on that path. Both of these requirements can be met by sending a PATH message from the sender to the receiver that contains the TSpec.

The other thing that happens is that each router looks this PATH message as it goes past, and it figures out the reverse path that will be used to send reservation from the receiver back to the sender in an effort to get the reservation to each router on the path.

Having received a PATH message, the receiver sends a reservation back “up” the multicast tree in a RESV message. This message contains the sender’s TSpec and an RSpec describing the requirements of this receiver. Each router on the path looks at the reservation request and tries to allocate the necessary resources to satisfy it. If the reservation can be made, the RESV request is passed on to the next router. If not, an error message is return to the receiver who made the request. If all goes well, the correct reservation is installed at every router between the sender and the receiver.

Now we can see what happens when a router or link fails. Routing protocols will adapt to the failure and create a new path from sender to receiver. PATH messages are sent about every 30 seconds, and may be sent sooner if a router detects a change in its forwarding table, so the first one after the new route stabilizes will reach the receiver over the new path. The receiver’s next RESV message will follow the new path and hopefully establish a new reservation on the new path. Meanwhile, the routers that are no longer on the path will stop getting RESV message and these reservations will time out and be released. Thus RSVP deals quite well with changes in topology, as long as routing changes are not excessively frequent.

As for the case of multi-senders and multi-receivers, it is discussed in detail in [7]. In Fig. 3, it graphically displays how senders make reservations on a multicast tree.

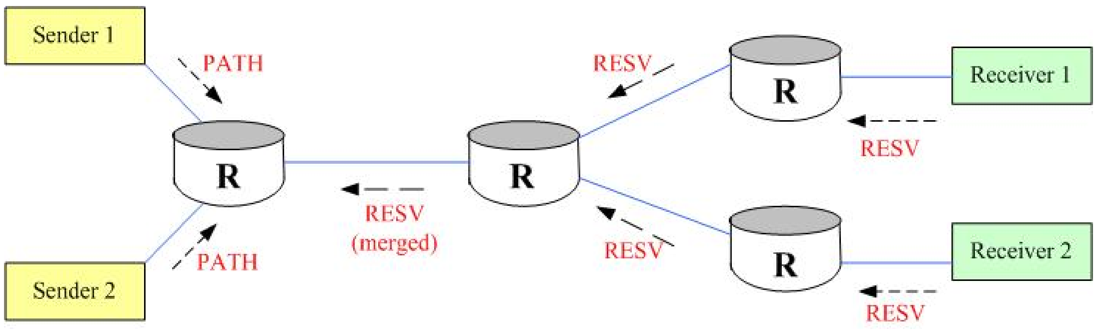


Fig. 3 Make reservation on a multicast tree

2.3 State-Dependent Routing: Separable Routing

Separable routing is the first of some routing schemes for circuit switched telephone traffic invented at Bellcore. These routing schemes are state-dependent, in the sense that, for each call attempt, a routing decision is made on the basis of the state of the network (defined in terms of the numbers of busy and idle trunks in the various links at the moment of the call attempt).

In the state-dependent network, the nodes are connected by links. An n-hop route is a route which traverses n links, or two end nodes and (n-l) via nodes. Therefore the goal is to find some rules which make an optimal or almost optimal routing decision, for each call attempt, as a function of the origin-destination of the call attempt and of the state of the network at the moment of the call attempt.

There are two kinds of state space of the system: “route-based state space” and “link-based state space”. We first discuss the route-based state space. For each node pair (S; D) we have a list of legal routes from S to D; the state of the system at time *t* is given by the array $\{\mathrm{N(S,D,R)}(t)\}$, where the element $\mathrm{N(S,D,R)}(t)$ gives the number of busy calls in over route R at time *t* for each node pair (S, D). As for the link-based state space, we number the link description *k* = 1, 2.... K, and that the state of the network at any time *t* is given by the array $\{X_k(t)\}$, $1 < k <$ K, where $X_k(t)$ is the number of busy calls in link *k* at time *t*.

It must be noted that if calls have independent, exponentially distributed holding times with the same mean (independent of the node pair of the call), and if each call that is routed as an n-hop call at once split into n independently terminating one-hop calls with exponential holding times of the same mean, then the link-based state description would be mathematically complete. The assumption is, of course, incorrect. The oversimplification is warranted because of the considerable decrease in complexity, and because it is a fairly minimal source of error.

We assume that there are N nodes (switches) in the network. The links will be numbered *k* = 1, 2..., $K = 1/2N \times (N-1)$. Link k has S_k trunks (unit of bandwidth), S_k > 0. We also assumed that S_k is given and does not vary over time. $X_k(t)$ denotes the number of busy trunks in link *k* at time *t*. We use the link-based state description, which means that the state of the network at time *t* is defined by the vector $X(t) = (X_1(t), X_2(t), \ldots, X_K(t))$.

In separable routing, we have costs $\Delta(k, X_k)$ $(0 \le X_k \le S_k)$ for every k $(1 \le k \le K)$, with the property that

$$0 < \Delta(k,0) < \Delta(k,1) < \mathrm{K} < \Delta(k, S_{k-1}) < \Delta(k, S_k) = 1 \tag{2.6}$$

, where $\Delta(k, X_k)$ is an estimate of the expected cost, in terms of additional calls blocked in the future, of now adding one call to link k if that link currently already has X_k busy trunks. Whenever a call attempt is made from node n_1 to node n_2, for each route $\mathrm{R} \in \mathrm{R}(n_1, n_2)$, we compute the state-dependent cost of routing the call over route R. This cost is computed by the formula

$$\cos t(R) = \sum_{k \in R} \Delta(k, X_k) \tag{2.7}$$

, where the summation is over all link k in route R. Eq. (2.7) assumes that X_k, the number of busy trunks in link k, is known for all $k \in \mathrm{R}$.

Next, we find the minimum cost route R^* :

$$\text{find} \quad \mathrm{R}^* \in \mathrm{R}(n_1, n_2) \quad \text{with} \quad \text{cost}(\mathrm{R}^*) = \min_{\mathrm{R} \in \mathrm{R}(n_1, n_2)} \text{cost}(\mathrm{R}) \tag{2.8}$$

The decision rule now is that

if cost $(\mathrm{R}^*) < 1$: route the call over R^*, (2.9a)

if cost $(\mathrm{R}^*) > 1$: block the call. (2.9b)

The intuition behind this rule is that blocking the call leads to the loss of exactly one call (the blocked call), while routing the call over route R leads to an expected number cost(R) of future blocked calls.

In [2], $N_A(t)$ is the total number of call attempts offered to the network during the time interval [0, t], and $N_B(t)$ is the total number of calls blocked during the same time interval. Furthermore, it is assumed that the statistical patterns in the call attempt process are constant over time (i.e. we assume that the call attempt process is a stationary process), so that the average arrival rate λ_A exists, where

$$\lambda_A = \lim_{t\to\infty} \frac{N_A(t)}{t} \tag{2.10}$$

, and we of course assume that

$$0 < \lambda_A < \infty \tag{2.11}$$

We also consider only state-dependent routing policies for which

$$g = \lim_{t\to\infty} \frac{N_B(t)}{t} \tag{2.12}$$

exist; *g* is called the overall blocking rate in the network (the average number of calls blocked per unit of time). Our goal is to find a routing scheme which minimizes *g*.

Eq. (2.10) and Eq. (2.12) imply the following exists,

$$P_B = \frac{g}{\lambda_A} = \lim_{t\to\infty} \frac{N_B(t)}{N_A(t)} \tag{2.13}$$

where P_B is called the average blocking probability.

Let **P** be a state-dependent routing policy. With some additional assumptions described in [2], we have a result which is stronger than (2.12)

$$E[N_B(t) \mid \mathbf{X}(0) = \mathbf{x},\ \mathbf{P}] = t \cdot g(\mathbf{P}) + v^{(\mathbf{P})}(\mathbf{x}) + o(1) \qquad (t \to \infty) \tag{2.14}$$

, where $E[\cdot]$ denotes the expectation operator and where x is any K-dimensional vector in the state space of the stochastic process $\mathbf{X}(t)$. In Eq. (2.14), the conditioning is on the initial state $\mathbf{X}(0)$ and on the fact that policy **P** is consistently used to make routing decisions, $v^{(\mathbf{P})}(x)$ is (up to an undetermined additive constant) the well-known relative value (in Markov decision processes) or cost of starting in state $\mathbf{X}(0) = x$. Eq. (2.14) also determines this constant. The "small *o*" symbol *o*(1) means that while for t → ∞ both the RHS and LHS in Eq. (2.14) go to infinity, the difference goes to zero. Since $v^{(\mathbf{P})}(x)$ represents an expected number of blocked (lost) calls, we call it the cost of starting in state x under policy **P**. In Markov decision theory, determining $v^{(\mathbf{P})}(x)$ from **P** is called the value determination step. It is convenient to define $v^{(\mathbf{P})}(x) = +\infty$ for all K-dimensional vector x not in the state space of the process $\mathbf{X}(t)$. Suppose that, for any policy **P**, the value determination step can be done. Then, given any policy $\mathbf{P_0}$, it is possible to find a new policy $\mathbf{P_1}$ which is at least as good as $\mathbf{P_0}$. $\mathbf{P_1}$ is found by the policy improvement step defined in Markov decision theory, see [1].

It is well known, see [3], that for this Erlang-B model with calls arriving according to a Poisson process with intensity λ and call holding times which are exponentially distributed with expected value 1, we say that this system is in state k ($0 \le k \le s$) when exactly k trunks are busy. In this model, the stationary probability that the system is in state k is equal to

$$p_k(s,\lambda) = \left(\frac{\lambda^k}{k!}\right) \Big/ \sum_{j=0}^{s} \frac{\lambda^j}{j!} \tag{2.15}$$

, and that the blocking probability equals

$$p_s(s,\lambda) = \left(\frac{\lambda^s}{s!}\right) \Big/ \sum_{j=0}^{s} \frac{\lambda^j}{j!} = B(s,\lambda) \tag{2.16}$$

, where $B(s,\lambda)$ is the well-known Erlang-B function which is defined by Eq. (2.16). The blocking rate $\boldsymbol{g}$ equals

$$g = g(s, \lambda) = \lambda \cdot B(s, \lambda) \tag{2.17}$$

In a telephone network where only direct routing is allowed, all links become independent systems. If each link k ($1 \le k \le K$) is modeled as an Erlang-B system with s_k trunks and arrival rate λ_k, then the cost function $v(\cdot)$ satisfies

$$v(x_1, \ldots, x_k) = \sum_{k=1}^{K} v_k(x_k) \tag{2.18}$$

$$v_k(j+1) - v_k(j) = \frac{B(s_k, \lambda_k)}{B(j, \lambda_k)} \qquad \text{for} \quad 0 \le j \le s_k \tag{2.19}$$

$$\sum_{j=0}^{s_k} p_j(s_k, \lambda_k) v_k(j) = 0 \tag{2.20}$$

where $[v_k(j+1) - v_k(j)]$ is the expected cost of the increase in the number of future blocked calls, of now adding one call to link k if the system is in state $x = (x_1, x_2, \ldots, x_K)$. It finds that the cost depends only on x_k and is independent of the states of the other links. We thus can take delta-costs

$$\Delta(k, x_k) = \frac{B(s_k, \lambda_k)}{B(x_k, \lambda_k)} \tag{2.21}$$

and use these delta-costs to derive a separable routing scheme.

2.4 Adaptive Proportional Routing: A Localized QoS Routing Approach

QoS (Quality-Of-Service) routing is concerned with the problem of how to select a path for a flow such that the flow's QoS requirements such as bandwidth or delay are likely to be met. Most of the QoS routing schemes proposed so far require periodic exchange of QoS state information among routers, imposing both communication overhead on the network and processing overhead on core routers. Furthermore, stale QoS state information causes the performance of these QoS routing schemes to degrade drastically. In order to circumvent these problems, we focus on localized QoS routing schemes where the edge routers make routing decisions using only local information and thus reducing the overhead at core routers.

We assume that source routing is used, and that network topology information is available to all source nodes (e.g., via the OSPF protocol), and one or multiple explicit-routed paths are setup a priori for each source and destination pair using, e.g., MPLS [5].

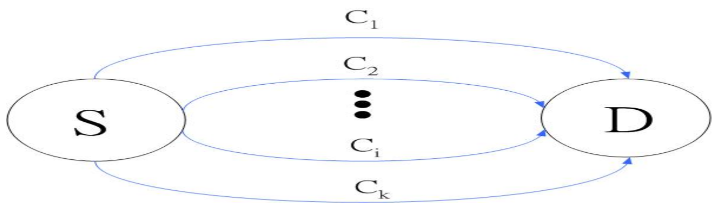


Fig. 4 Disjoint paths between a source-destination pair

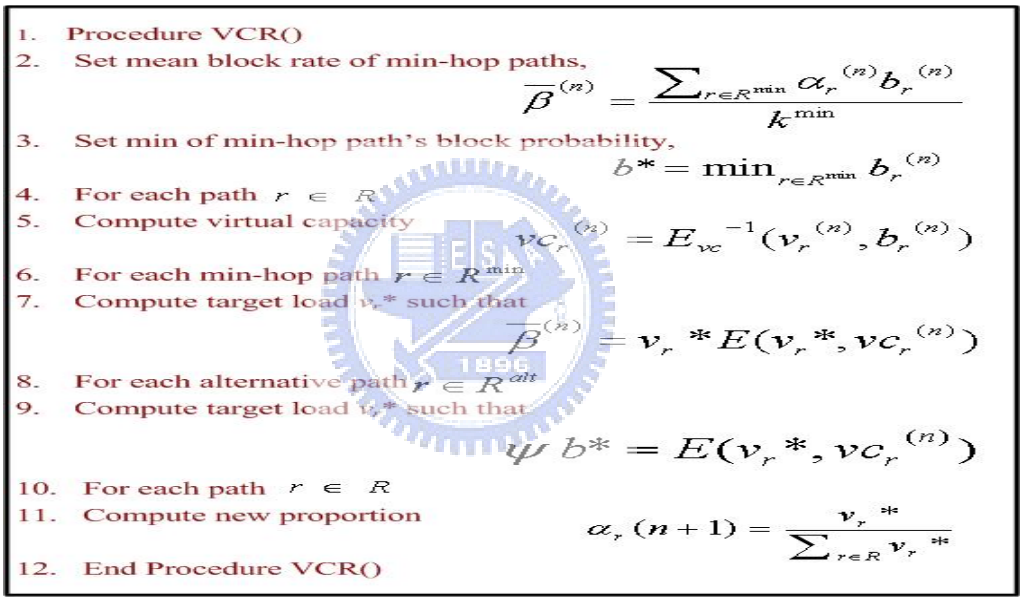
Consider a simple topology shown in Fig.4, where a source and a destination are connected by k disjoint paths r_1, r_2, K, r_K. Each path r_i has a (bottleneck) capacity of c_i units of bandwidth and is assumed to be known to the source S. Suppose that calls arrive at the source S at an average rate λ, and the average call holding time is 1/μ. We assume that call arrivals are Poisson and call holding times are exponentially distributed. For simplicity, we also assume that each call consumes one unit of bandwidth. In other words, path r_i can accommodate c_i calls at any time. Suppose that, on the average, the proportion of calls routed along path r_i is α_i, where $i = 1,2,K, K$ and $\sum_{i=1}^{K} \alpha_i = 1$. Then the blocking probability b_i at path r_i is:

$$b_i = E(v_i, c_i) = \frac{\dfrac{v_i^{c_i}}{c!}}{\sum_{n=0}^{c_i} \dfrac{v_i^{n}}{n!}}$$

, where $v_i = \alpha_i(\lambda / \mu)$ is referred to as the average load on path r_i. The total load on the system is denoted by $v = \sum_{i=1}^{k} v_i = \lambda / \mu$.

There are two alternative strategies for flow proportioning: equalization of blocking probabilities (ebp) and equalization of blocking rates (ebr). Here we skip the first strategy and account for the ebr strategy. The objective of the ebr strategy is to find a set of proportions $\{\hat{\alpha}_1, \hat{\alpha}_2, K, \hat{\alpha}_K\}$ such that flow blocking rates of all the paths are equalized, i.e., $\hat{\alpha}_1 \hat{b}_1 = \hat{\alpha}_2 \hat{b}_2 = \Lambda = \hat{\alpha}_K \hat{b}_K$, where $\hat{b}_i$ is the flow blocking probability of path r_i.

By incorporating this self-refrained alternative routing method into the virtual capacity model defined in [3], it devise a theoretical adaptive proportional routing scheme, which is referred to as the Virtual Capacity based Routing (vcr) scheme. In this vcr scheme, it uses the ebr strategy to proportion calls along the min-hop paths, and proportion calls along the alternative paths. The scheme is shown in Fig. 5.



```
1.  Procedure VCR()
2.    Set mean block rate of min-hop paths,   $\overline{\beta}^{(n)} = \frac{\sum_{r \in R^{\min}} \alpha_r^{(n)} b_r^{(n)}}{k^{\min}}$
3.    Set min of min-hop path's block probability,   $b^* = \min_{r \in R^{\min}} b_r^{(n)}$
4.    For each path $r \in R$
5.    Compute virtual capacity   $vc_r^{(n)} = E_{vc}^{-1}(v_r^{(n)}, b_r^{(n)})$
6.    For each min-hop path $r \in R^{\min}$
7.    Compute target load $v_r^*$ such that   $\overline{\beta}^{(n)} = v_r^* E(v_r^*, vc_r^{(n)})$
8.    For each alternative path $r \in R^{alt}$
9.    Compute target load $v_r^*$ such that   $\psi b^* = E(v_r^*, vc_r^{(n)})$
10.   For each path $r \in R$
11.   Compute new proportion   $\alpha_r(n+1) = \frac{v_r^*}{\sum_{r \in R} v_r^*}$
12.   End Procedure VCR()
```

Fig. 5 VCR Procedure

Suppose the total load for a source–destination pair is *v*. At a given step, n ≥ 0 let $v_r^{(n)} = \alpha_r^{(n)} \times v$ be the amount of the load currently routed along a path $r \in \mathrm{R}$, and let $b_r^{(n)}$ be its observed blocking probability on that path. Then the virtual capacity of path r is given by $vc_r^{(n)} = E_{vc}^{-1}(v_r^{(n)}, b_r^{(n)})$. For each min-hop path, the mean blocking rate of all the min-hop paths $\overline{\beta}^{(n)}$ is used to compute a new target load. Similarly, for each alternative path, a new target load is computed using the target blocking probability $\psi \times b^*$ where ψ is a constant parameter. Given these new target loads for all the paths, the new proportion of flows, $\alpha_r^{(n+1)}$, for each path *r*, is obtained in lines10–11, resulting in a new load $v_r^{(n+1)} = \alpha_r^{(n+1)} \times v$ on path *r*.

Chapter 3
Localized State-Dependent Routing

In this chapter, we first describe the shortcomings of the state-dependent separable routing scheme [2] and adaptive proportional routing scheme [3, 4]. We then propose three possible boosting methods in anticipate that the shortcomings of both routing schemes can be remedied.

3.1 Shortcomings of State-Dependent Routing and Proportional Routing

In chapter 2, we understand the concepts of state-dependent separable routing scheme and adaptive proportional routing scheme. In the state-dependent separable routing formulated by the Markov decision process, a routing decision is made on the basis of the state of the network. It gathers global network state information and selects a best path with minimum delta-cost for an incoming flow based on network state information. The routing will work well when each source node has a reasonably accurate global view of the network state. Since network resource availability changes with each flow arrival and departure, it is impractical to maintain an accurate view of the network state, due to prohibitive communication and processing overheads entailed by frequent state information exchanges. However, as our simulation in Chapter 4 shown, inaccurate view of network state increases the blocking rate dramatically.

In adaptive proportional routing scheme feasible paths are selected based on infrequently exchanged average available bandwidth information and flows are adaptively proportioned among these feasible paths based on locally collected information from other nodes. However, our simulation also shows that the adaptive proportional routing approach works significantly worse than the state-dependent separable routing approach if the network state information obtained in the source is delayed not more than one minute.

3.2 Overview of Proposed Boosting Methods

The above discussions give us the motivation for boosting the state-dependent separable routing by Markov Decision Process (MDP Routing for shorted) on the basis of Proportional Routing approach to compensate each other. In this section, we propose three methods to boost these two routing schemes.

First, we can statically boost these two routing schemes by assigning fixed weight. For example, we can assign weight $\delta (0 \le \delta \le 1)$ for Proportional Routing and weight (1- δ) for MDP Routing. If we prefer the Proportional Routing scheme, we can increase the weight δ. It is noteworthy that the selection of optimal weight δ is network-dependent and is expected to be a difficult task. We propose such a boosting simply for the comparison purpose.

Second, we dynamically adjust the weight of these two routing schemes. Given the observed blocking probability of these two routing schemes for a fixed time interval, we can calculate the new weight of these schemes for next time interval to favor the one with low blocking probability. Using the new weight, we can randomly choose one of these two schemes to select one best path for various network situations.

And from the observation in Chapter 2, we find that the Proportional Routing approach works well than the MDP Routing when its update interval is more than one minute. So the third method is that we can dynamically adjust the weight based on the delay time of local state information piggybacked in RESV message. If the state information is delayed too long, the state information must be stale and therefore we prefer to use the average bandwidth information in Proportional Routing to select a best path. Otherwise, we use the state information in MDP routing because it is fresh enough. Before we introduce these methods in detail, we make some assumptions for our model

3.3 Assumptions

We assume that source routing is used, and that network topology information is available to all source nodes (via the OSPF protocol), and one or multiple explicit-routed paths are setup a priori for each source and destination pair. In Fig.6, source S has K feasible paths $\{ r_i \mid i = 1,2,\ldots,K \}$. Each path r_i can accommodate c_i unit of bandwidth. For source S, we assume that call arrivals are Poisson with intensity λ_s and call holding times are independent, exponentially distributed with mean μ (independent of the node pair of the call). For simplicity, we also assume that each call consumes one unit of bandwidth. Suppose that, on the average, the proportion of calls routed along path r_i is α_i, where $i = 1,2,\ldots,K$ and $\sum_{i=1}^{K} \alpha_i = 1$.

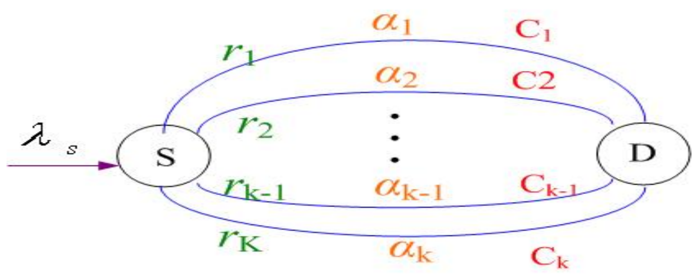


Fig. 6 Source routing

And in the following sections, we will introduce particularly these three methods to boost the Proportional Routing and MDP Routing in various network situations.

3.4 Boosting with Fixed Weights

Every "*Update Proportion Interval*" (*upi*), source S calculates the new proportion of paths $\{fp[\cdot]\}$ by the VCR algorithm defined in Fig. 5 according to the old proportions of paths $\{fp[\cdot]\}$, the observed blocking probability of paths $\{bp[\cdot]\}$, and the load of source S $\{v_s\}$ during this update proportion interval. And every "*Update Lambda Interval*" (*uli*), source S calculates λ_i, the average arrival rate of path *i*, in order to compute the delta-cost $\{dc[\cdot]\}$ defined in Eq. (2.21).

When a new call arrives, we first check whether there is available bandwidth or not. If there is no available bandwidth, we have to reject this call. Otherwise, we select one best path based on the state information. We have two kinds of approaches to choose one path, i.e. Proportional Routing and MDP Routing. We can assign fixed weight δ $(0 \le \delta \le 1)$ for Proportional Routing and weight (1- δ) for MDP Routing statically. According to the following pseudo code, we can find out the best path and use RSVP to setup this call along the selected path.

```
// pseudo code for path-selecting
If (available bandwidth)
    tmp =   (rand()%1000) /1000.0;                  // random value : tmp
    if( 0<=tmp && tmp<$\delta$)
        using " Proportional Routing " to select path
    else                        // if ($\delta$<=tmp && tmp<1 )
```

```
            using " MDP Routing " to select path
    else
        Reject this call
```

3.5 Boosting with Dynamic Weights Adjusted by Blocking Probability

Besides *Update Proportion Interval* and *Update Lambda Interval*, we calculate the blocking probability for calls using Proportional Routing and MDP Routing for every "*Update Weight Interval*" (*uwi*). Then we calculate new weights using the following equations.

In "*n*-th" *Update Weight Interval*:

For Proportional Routing:

$$e^{bp\_PR^{(n)}} = e^{bpPR^{(0)}} \times e^{bpPR^{(1)}} \times \Lambda \times e^{bpPR^{(n-1)}} \times e^{bpPR^{(n)}} = e^{bp\_PR^{(n-1)}} \times e^{bpPR^{(n)}}$$

where $e^{bpPR^{(k)}}$ is the blocking probability in $k-th$ *Update Weight Interval*

For MDP Routing:

$$e^{bp\_MR^{(n)}} = e^{bpMR^{(0)}} \times e^{bpMR^{(1)}} \times \Lambda \times e^{bpMR^{(n-1)}} \times e^{bpMR^{(n)}} = e^{bp\_MR^{(n-1)}} \times e^{bpMR^{(n)}}$$

where $e^{bpMR^{(k)}}$ is the blocking probability in $k-th$ *Update Weight Interval*

New weight for using Proportional Routing:

$$\delta^{(\mathrm{n})} = \frac{e^{bp\_PR^{(n)}}}{e^{bp\_PR^{(n)}} + e^{bp\_MR^{(n)}}} \qquad (3.1a)$$

New weight for using MDP Routing:

$$1-\delta^{(\mathrm{n})} = \frac{e^{bp\_MR^{(n)}}}{e^{bp\_PR^{(n)}} + e^{bp\_MR^{(n)}}} \qquad (3.1b)$$

Different from the previous method which using fixed weights δ, we calculate the new weight $\delta^{(n)}$ for Proportional Routing and $1-\delta^{(n)}$ for MDP Routing in the n-th *Update Weight Interval*. According to the following pseudo code, we can find out the best path and use RSVP to setup this call along the selected path.

```
// pseudo code for path-selecting
If (available bandwidth)
    tmp =   (rand()%1000) /1000.0;                    // random value : tmp
    if( 0<=tmp && tmp< $\delta^{(n)}$)
        using " Proportional Routing " to select path
    else                              // if ($\delta^{(n)}$<=tmp && tmp<1 )
        using " MDP Routing " to select path
else
    Reject this call
```

3.6 Boosting with Dynamic Weights Adjusted by the Delay Time of State Information

After source S receives the RESV message of RSVP, source S records the moment of message feedback from path i, i.e. $t_0[i]$. The delay time, $dt[i]$, of a new call attempt at time t is calculated as follows:

For path i, $\quad dt[i] = t - t_0[i] \qquad i = 1,2,\ldots,K$

Different from the previous two methods which choose one of these two routing approaches to select one best path, we non-linearly combine MDP Routing and Proportional Routing by the exponentially decayed weights. We introduce this method in detail as follow:

For path i, $i = 1,2,\ldots,K$, we calculate the path-combine-proportion $pcp[i]$, using the following equation :

$$pcp[i] = (1 - e^{-\gamma \cdot dt[i]}) \cdot fp[i] + e^{-\gamma \cdot dt[i]} \cdot (1 - dc[i]) \tag{3.2}$$

where γ is a configurable parameter, and

$dt[i]$ is the delay time of this call for path i,

$dc[i]$ is the delta-cost of this call for path i using Eq .(2.21) in MDP routing

$fp[i]$ is the flow-proportion of path i using VCR algorithm in proportional routing,

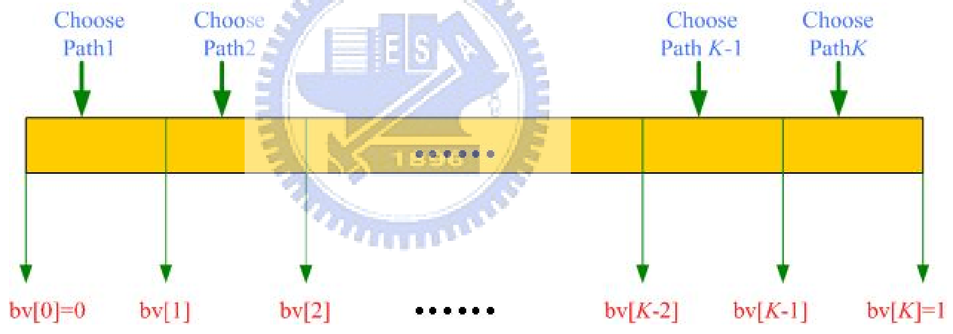
And then we normalize the path-combine-proportion $pcp[i]$ for path i, $i = 1,2,\ldots,K$

$$pcp[i] = pcp[i] \Big/ \sum_{j=1,2,\ldots,K} pcp[j] \tag{3.3}$$

Finally we select one path proportionally based on $pcp[i]$ for path i , $i = 1,2,\ldots,K$

Given a random value rv , $0 \le rv < 1$, $m = 1,2,\ldots,K$

If $bv[m-1] \le rv \le bv[m]$, we choose path m as the best path.



where

$$\begin{aligned} bv[0] &= 0 \\ bv[1] &= 0 + pcp[1] \;; \\ bv[2] &= bv[1] + pcp[2] \;; \\ &\vdots \\ bv[K] &= bv[K\text{-}1] + pcp[K] = 1 \end{aligned}$$

When the average load of source is heavy, the state-information is fresh because the update interval (delay-time information $dt[i]$) of the source is short. When the state information is fresh, we have better to choose the MDP Routing scheme to find the best path. On the other hand, if the state information is delayed and outdated, we need to use the Proportional Routing scheme. Therefore, we have to multiply $(1 - dc[i])$ by $e^{-\gamma \times dt[i]}$ and multiply $fp[i]$ by

$1 - e^{-\gamma \times dt[i]}$ in order to cooperate these two schemes adaptively and choose one path proportionally. So we use the different weight with exponentially decayed for MDP routing and proportional routing. And by sending RESV message which additionally contains the state information back to source, source obtains the local network information feedback from the routers, and therefore we do not increase the communication overhead.

In the summary, we illustrate our methods by the following block diagram.

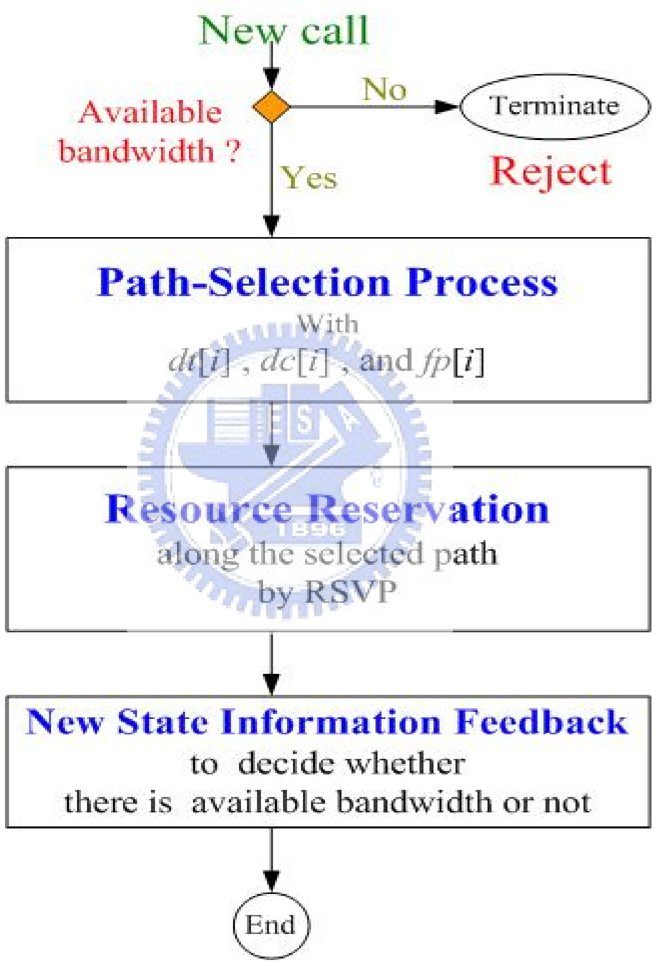


Fig. 7 Boosting process

Chapter 4
Simulation Results

In the following we illustrate a simple network topology how our scheme works better than adaptive proportional routing when the load of source is varying with time.

Consider the fish topology shown in Fig. 8.

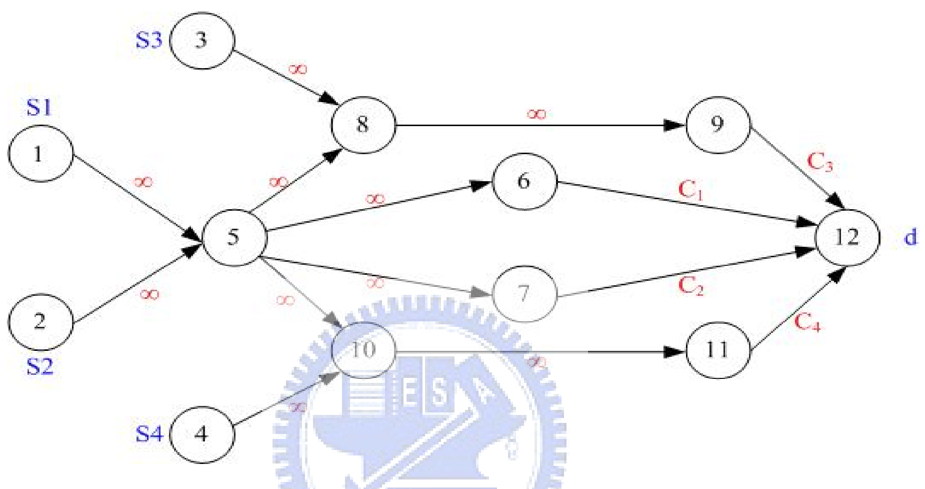


Fig. 8 Fish topology

The nodes 1, 2, 3, and 4 are the source nodes and node 12 is the destination node. Each of node 1 and node 2 has two min-hop paths (1→5→6→12, 1→5→7→12 and 2→5→6→12, 2→5→7→12) and two alternative paths (1→5→8→9→12, 1→5→10→11→12 and 2→5→8→9→12, 2→5→10→11→12) to the destination node 12. Other two sources, nodes 3 and node 4, have just one min-hop path (3→8→9→12 and 4→10→11→12) to the destination node 12.The alternative path of source node1 and node 2 share the bottleneck link 9 → 12 and 11→ 12 with the min-hop paths of source node 3 and node4. We assume that the capacities c1, c2, c3, and c4 of the bottleneck links are all set to 80 units of bandwidth and others are infinite bandwidth.

The follow are our simulator design for this fish topology. At first, the Object Model Diagram in UML is illustrated in Fig. 9 below. There are four Source objects constructed by system in our simulation model. Source S_1 and source S_2 have four feasible paths and source S_3 and source S_4 have only one feasible path.

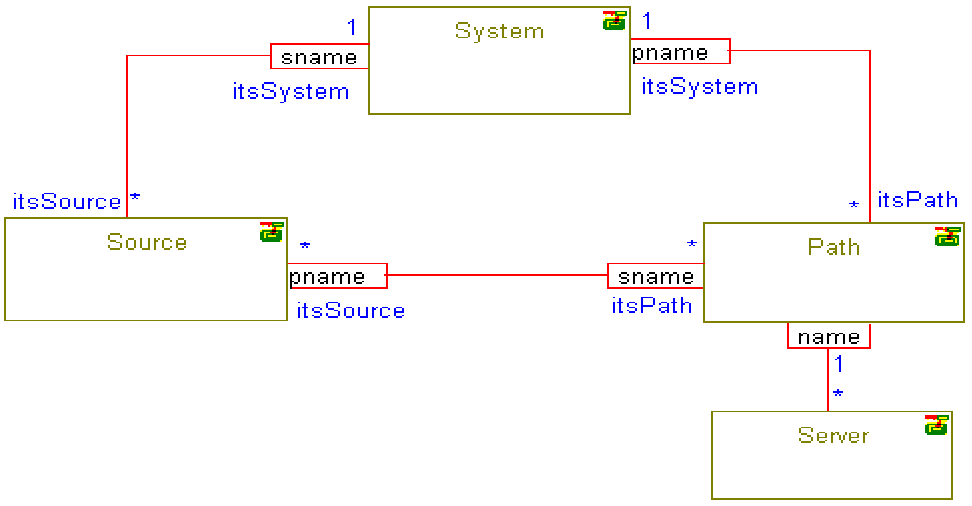


Fig. 9 Object Model Diagram in UML

4.1 Comparison of Proportional Routing and MDP Routing

Before displaying the simulation performance of our proposed routing, we first compare these two routing approaches "State-Dependent Separable Routing formulated by MDP "(called MDP routing for shorted) and "Adaptive Proportional Routing" (called Proportional routing for shorted). We observe the blocking probability of source S_1 by increasing the update interval of S_1 when the average load of all sources is set to 40.

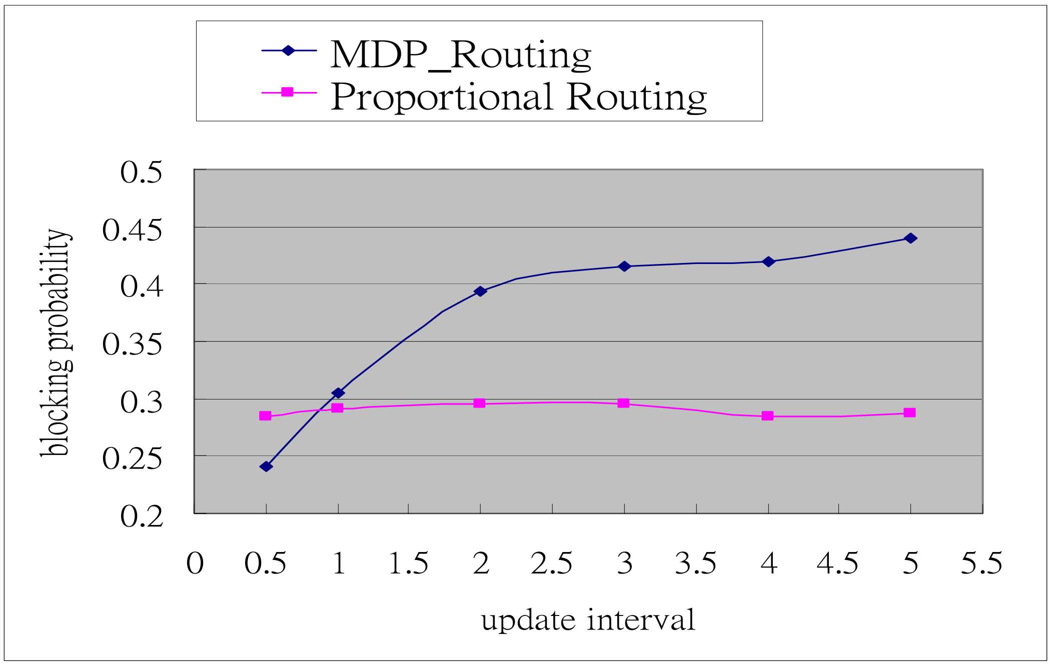


Fig. 10 Comparison of Proportional Routing and MDP Routing

In Fig. 10, we find that the performance of MDP Routing degrades rapidly as the update interval increases. As for the scheme of Proportional Routing, the blocking probability of S_1 is keeping at about 0.3 and is even better than the scheme of MDP Routing when its update interval is longer than one minute. It is because that the proportional routing adopts the average available bandwidth information instead of stale state information to make path selection.

From the observation in Fig. 10, we proposed three boosting methods, which are mentioned at Chapter 3 in detail. Before we illustrate how these three routing methods adaptively adjust the load proportion of feasible paths as the load increases, we firstly introduce some parameters in our simulator.

4.2 Simulator Parameters Setting

We assume that the average offered load of S_1, S_2, S_3, and S_4 are 40, 40, 5, and 5 respectively in the beginning. And we set the "*Update Proportional Interval*" as 10 minutes, "*Update Lambda Interval*" as 1 minute and "*Update Weight Interval*" as 1 minute. For source S_1, we set the initial flow proportion of path 1, 2, 3, and 4 to 0.4999, 0.4999, 0.0001, and 0.0001 respectively. So does source S_2. And we set the constant parameter γ in Eq.(3.2) to 0.01. Consider the scenario: when the number of calls generated by source S_1 is equal to 1000, we increase the average load of S_3 and S_4 to 20. And when the number of calls generated by source S_1 is equal to 3000, we increase the average load of S_3 and S_4 to 40. By increasing the average load of S_3 and S_4, we study how source S_1 and S_2 adjust their flow proportion on the feasible paths in order to decrease the overall blocking probability.

The following table is the setting of parameters in our simulator.

Parameter	Initial value
Path1_Flow-proportion of S1 and S2	0.499
Path2_Flow-proportion of S1 and S2	0.499
Path3_Flow-proportion of S1 and S2	0.001
Path4_Flow-proportion of S1 and S2	0.001
Average Load of S1 and S2	40 calls/min

Average Load of S3 and S4	5 calls/min
Link Capacity (c1 , c2 , c3, c4)	80 calls
psi (ψ)	0.95
Update Proportional Interval	10 minutes
Update Lambda Interval	1 min
Update Weight Interval	1 min
Total Arrival of Source1	5000 calls
Gamma(γ)	0.01

Table .1 Setting of parameters in our simulator

In the following sections, we will introduce our simulator for these three methods and their simulation results.

4.3 Simulator Design and Results for BFW

In this method, we assign fix weight δ to Proportional Routing and fix weight $1-\delta$ to MDP Routing. We define this method as "Boosting with Fixed Weight" (BFW for shorted).

4.3.1 System Operations

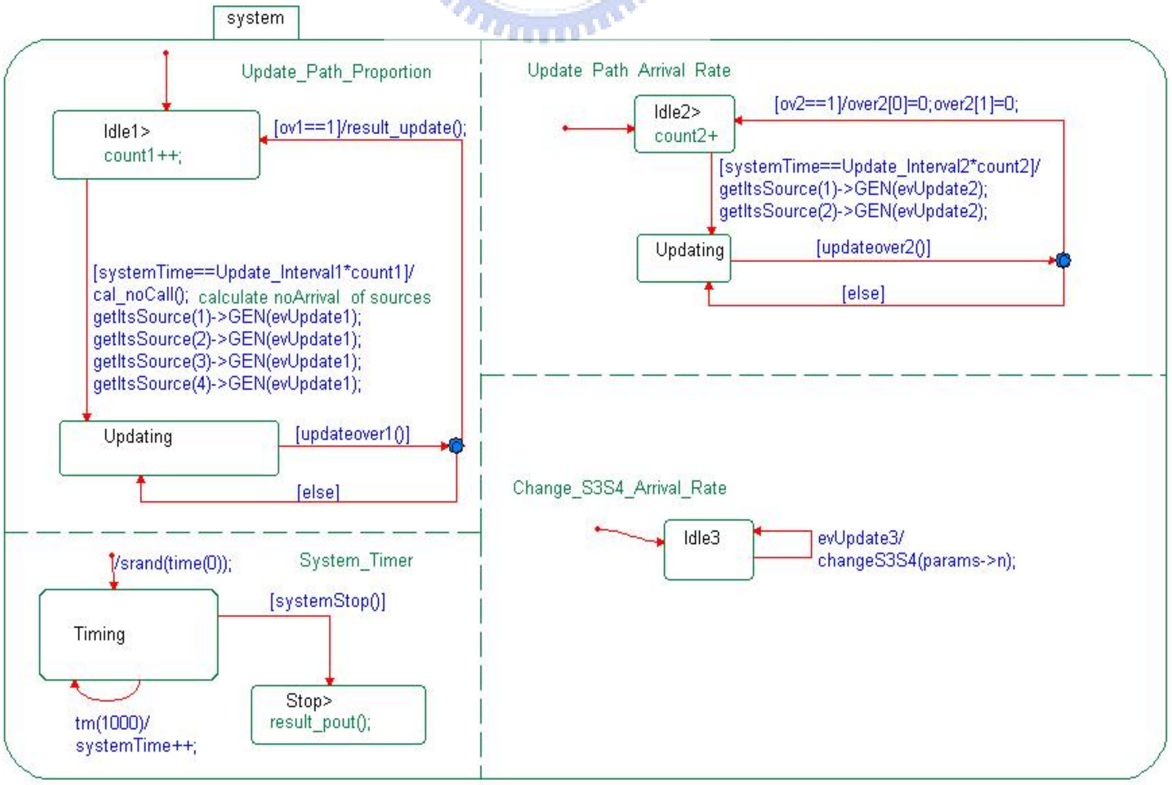
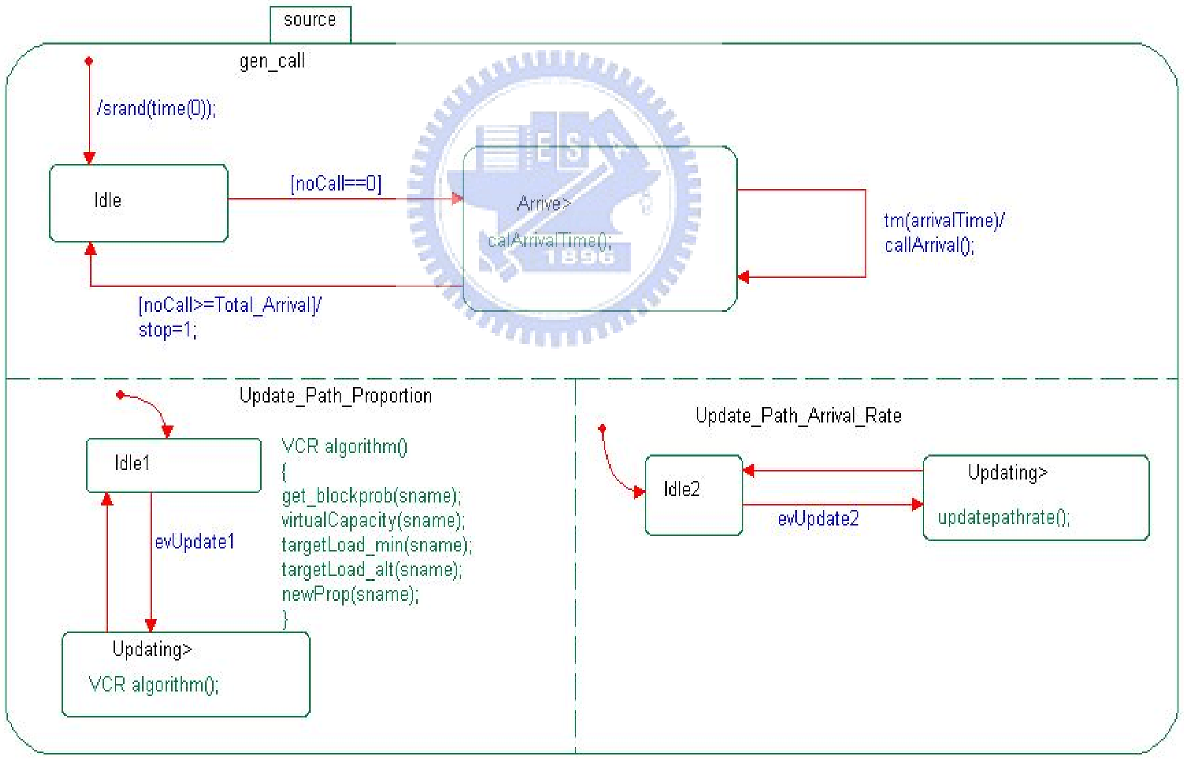

system
Update_Path_Proportion
Idle1>
count1++;
[ov1==1]/result_update();
[systemTime==Update_Interval1*count1]/
cal_noCall(); calculate noArrival of sources
getItsSource(1)->GEN(evUpdate1);
getItsSource(2)->GEN(evUpdate1);
getItsSource(3)->GEN(evUpdate1);
getItsSource(4)->GEN(evUpdate1);
Updating
[updateover1()]
[else]
Update_Path_Arrival_Rate
Idle2>
count2+
[ov2==1]/over2[0]=0;over2[1]=0;
[systemTime==Update_Interval2*count2]/
getItsSource(1)->GEN(evUpdate2);
getItsSource(2)->GEN(evUpdate2);
Updating
[updateover2()]
[else]
/srand(time(0));
System_Timer
Timing
[systemStop()]
Stop>
result_pout();
tm(1000)/
systemTime++;
Change_S3S4_Arrival_Rate
Idle3
evUpdate3/
changeS3S4(params->n);

Fig. 11 System diagram of BFW in UML

In Fig.11, there are four sub-states. In System_Timer sub-state, timer is running and the parameter systemTime plus one per second. And in Update_Path_Proportion sub-state, system calculates the total arrival of sources during the "*Update Proportion Interval*", and triggers the event *evUpdate1* of sources to calculate the new path flow proportion ($fp[\cdot]$) by VCR algorithm. And in Update_Path_Arrival_Rate sub-state, system triggers the event *evUpdate2* of sources to calculate the path arrival rate (λ_i) during the "*Update Lambda Interval*". Finally, system can change the arrival rate of source S_3 and S_4 in the Change_S_3S_4_Arrival_Rate sub-state.

4.3.2 Source Operations

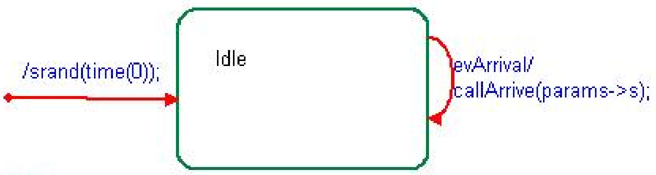


```
VCR algorithm()
{
get_blockprob(sname);
virtualCapacity(sname);
targetLoad_min(sname);
targetLoad_alt(sname);
newProp(sname);
}
```

Fig. 12 Source diagram of BFW in UML

In Fig. 12, there are three sub-states. In the gen_call sub-state, source generates calls by Poisson arrival process. Each call performs callArrival() to find a path by Path-Selection-Process based on the value of δ, path flow-proportion ($fp[\cdot]$), and path delta-cost ($dc[\cdot]$). And then source uses RSVP to setup this call along the selected path. When the number of call generated by source equals to Total_Arrival, source stops generating new call. In Update_Path_Proportion sub-state, the event *evUpdate1* is triggered from system and source updates the path flow-proportion ($fp[\cdot]$) by VCR algorithm. At last, in Update_Path_Arrival_Rate sub-state, the event *evUpdate2* is triggered from system and source updates the path arrival rate (λ_i) in order to calculate the path-delta-cost, ($dc[\cdot]$) in Eq. (2.21).

4.3.3 Path Operations



```
callArrive(int SID)
if(SID==1 || SID==2)
{
        if(count<Max_Capacity)
        {
           count++;
           noAccept++;
           getItsSource(SID)->ps[pname-1]= count;
           addItsServer(noAccept,new Server(noAccept));
           getItsServer(noAccept)->startBehavior();
        }
        else
        {
           getItsSource(SID)->ps[pname-1]= Max_Capacity;
           getItsSource(SID)->noBlock[pname-1]++;
        }
}

if(SID==3 || SID==4)
{
        if(count<Max_Capacity)
        {
           count++;
           noAccept++;
           addItsServer(noAccept,new Server(noAccept));
           getItsServer(noAccept)->startBehavior();
        }
        else
        {
           getItsSource(SID)->noBlock[pname-1]++;
        }
}
```

Fig. 13 Path diagram of BFW in UML

In Fig. 13, it demonstrates that the selected path accepts this call which consumes one unit of bandwidth (count plus one) and triggers one server object. At the same time, the selected path has to inform source, which generates this call, the latest state information by RESV message.

4.3.4 Server Operations

In Fig. 14, it describes that call holding time is exponentially distributed with mean μ. After the serviceTime, this call stops serving and the server object will be terminated.

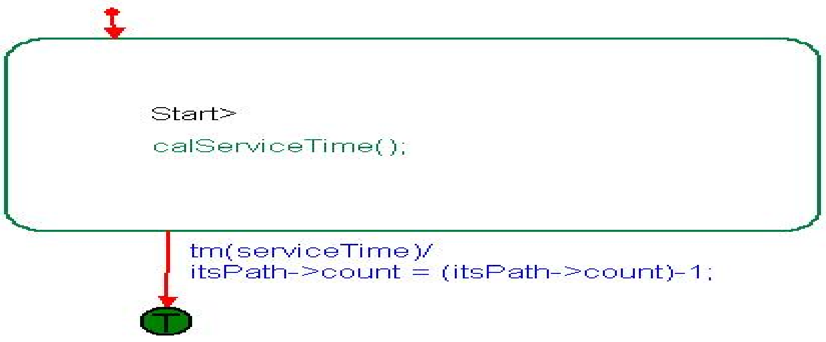


Fig. 14 Server diagram of BFW in UML

In summary, our simulator model could be illustrated in the sequence diagram below.

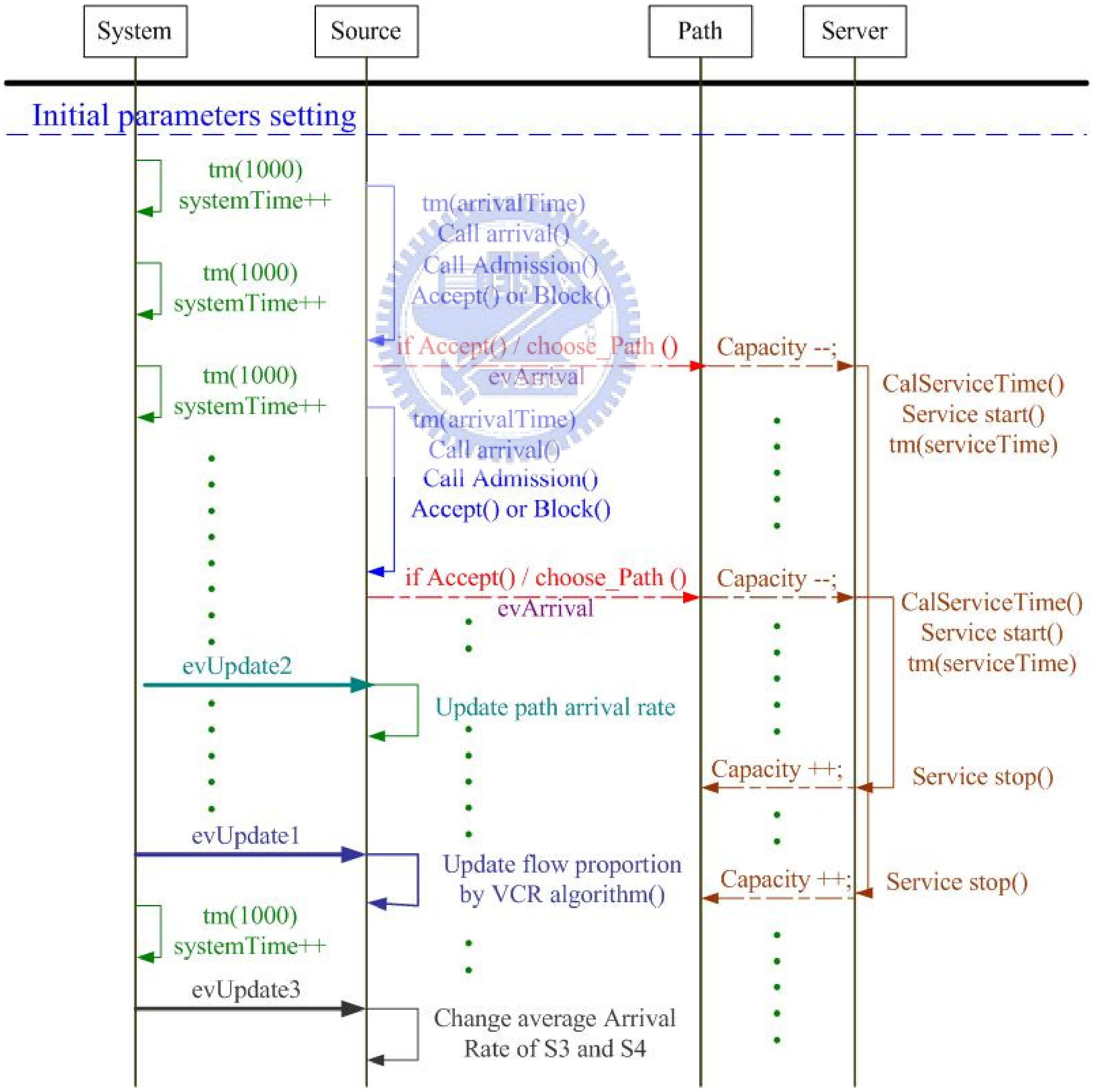


Fig. 15 Sequence diagram for BFW in UML

4.3.5 Simulation Result

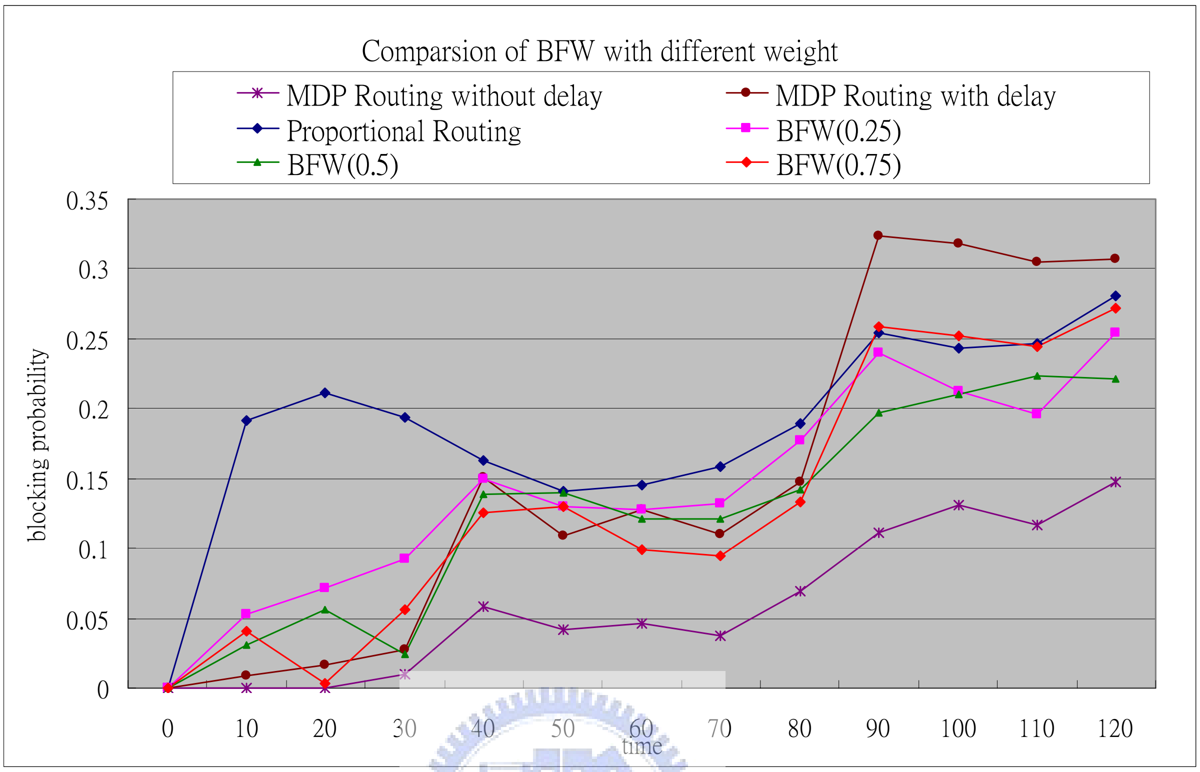


Fig. 16 Comparison of BFW with different weight (δ)

In Fig.16, the scheme of MDP Routing without delay is nearly optimal because sources obtain instantaneous network state information. We can find that when the average load of S_3 and S_4 are low, the performance of Proportional Routing is worst. As the average load of S_3 and S_4 are increasing, the performance of MDP Routing with delayed information gets worse. When the average loads of S_3 and S_4 increase to 40, the performance of MDP Routing with delayed information gets worse than Proportional Routing. Consider our proposed method BFW with different value of δ which is the weight of using Proportional Routing to select one path. No matter what the value δ is, the performance of BFW is better than Proportional Routing and is better than MDP Routing with delayed information when the load is heavy.

4.4 Simulator Design and Results for BDW-BP

In this method, we dynamically boost the weight adjusted by the observed blocking probability during a fixed time interval. We define this method as “Boosting with Dynamic Weights Adjusted by Blocking Probability” (BDW-BP for shorted). The server operation in UML is the same as previous method.

4.4.1 System Operations

In order to update the new weights of the new *Update Weight Interval*, we have to calculate the blocking probability of calls using Proportional Routing and calls using MDP Routing in the Update_Blocking_Probability sub-state in Fig. 17.

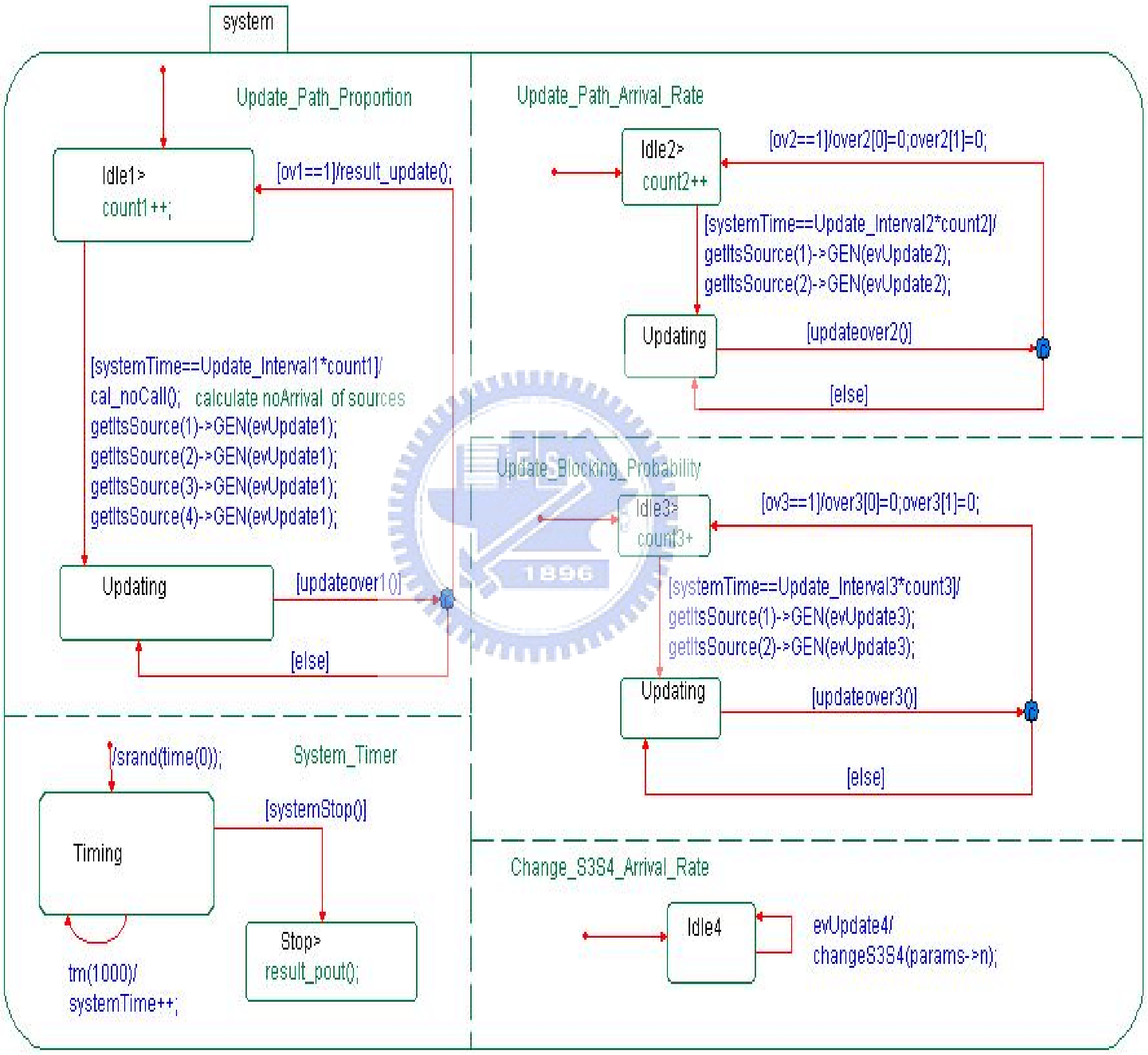

system
Update_Path_Proportion
Idle1>
count1++;
[ov1==1]/result_update();
[systemTime==Update_Interval1*count1]/
cal_noCall(); calculate noArrival of sources
getItsSource(1)->GEN(evUpdate1);
getItsSource(2)->GEN(evUpdate1);
getItsSource(3)->GEN(evUpdate1);
getItsSource(4)->GEN(evUpdate1);
Updating
[updateover1()]
[else]
Update_Path_Arrival_Rate
Idle2>
count2++
[ov2==1]/over2[0]=0;over2[1]=0;
[systemTime==Update_Interval2*count2]/
getItsSource(1)->GEN(evUpdate2);
getItsSource(2)->GEN(evUpdate2);
Updating
[updateover2()]
[else]
Update_Blocking_Probability
Idle3>
count3+
[ov3==1]/over3[0]=0;over3[1]=0;
[systemTime==Update_Interval3*count3]/
getItsSource(1)->GEN(evUpdate3);
getItsSource(2)->GEN(evUpdate3);
Updating
[updateover3()]
[else]
/srand(time(0));
System_Timer
Timing
[systemStop()]
Stop>
result_pout();
tm(1000)/
systemTime++;
Change_S3S4_Arrival_Rate
Idle4
evUpdate4/
changeS3S4(params->n);

Fig. 17 System diagram of BDW-BP in UML

4.4.2 Source Operations

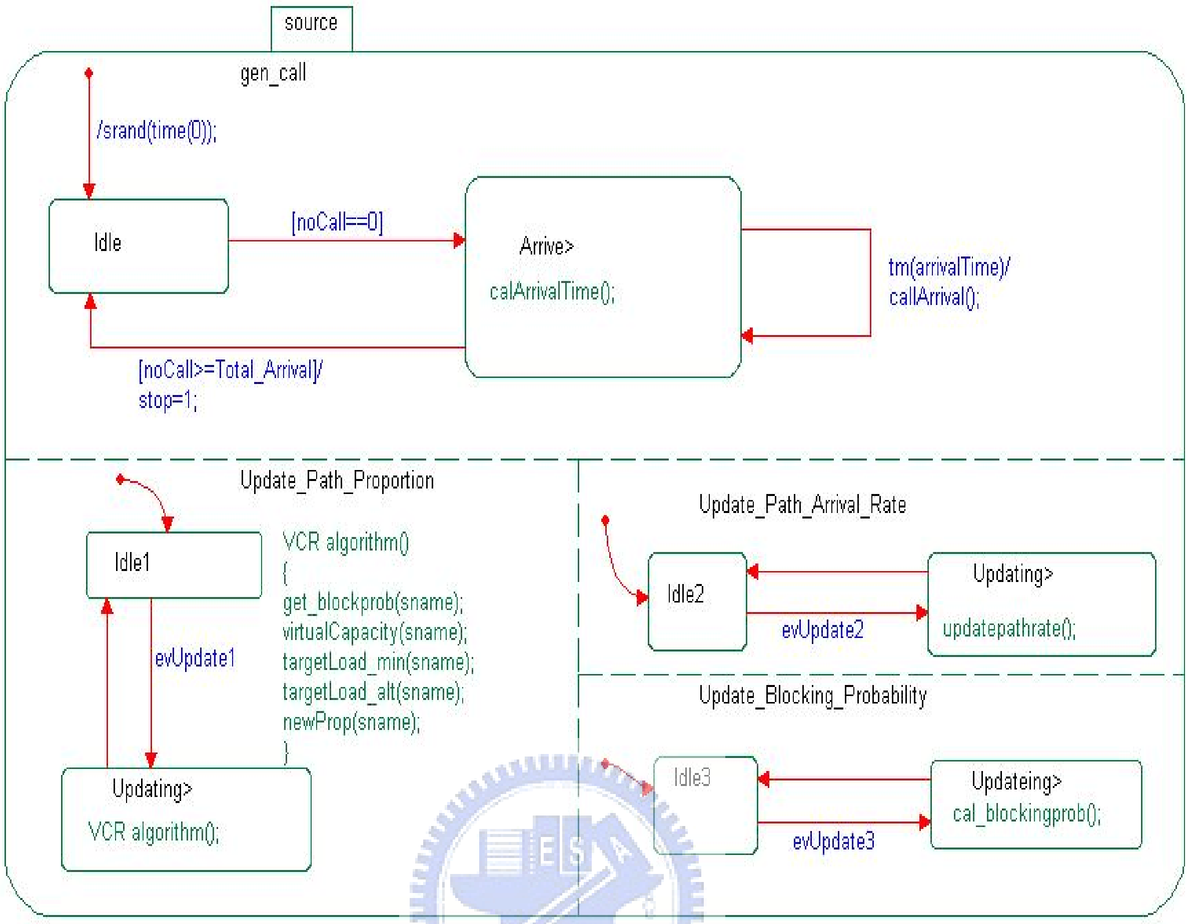


Fig. 18 Source Diagram of BDW-BP in UML

Different from the source operations in BFW, when the event *evUpdate3* is triggered, source has to calculate the blocking probability of calls using Proportional Routing and calls using MDP Routing in order to find out the new weights using Eq. (3.1a) and Eq. (3.1b).

In " *n*-th " *Update Weight Interval*:

New weight for using Proportional Routing:

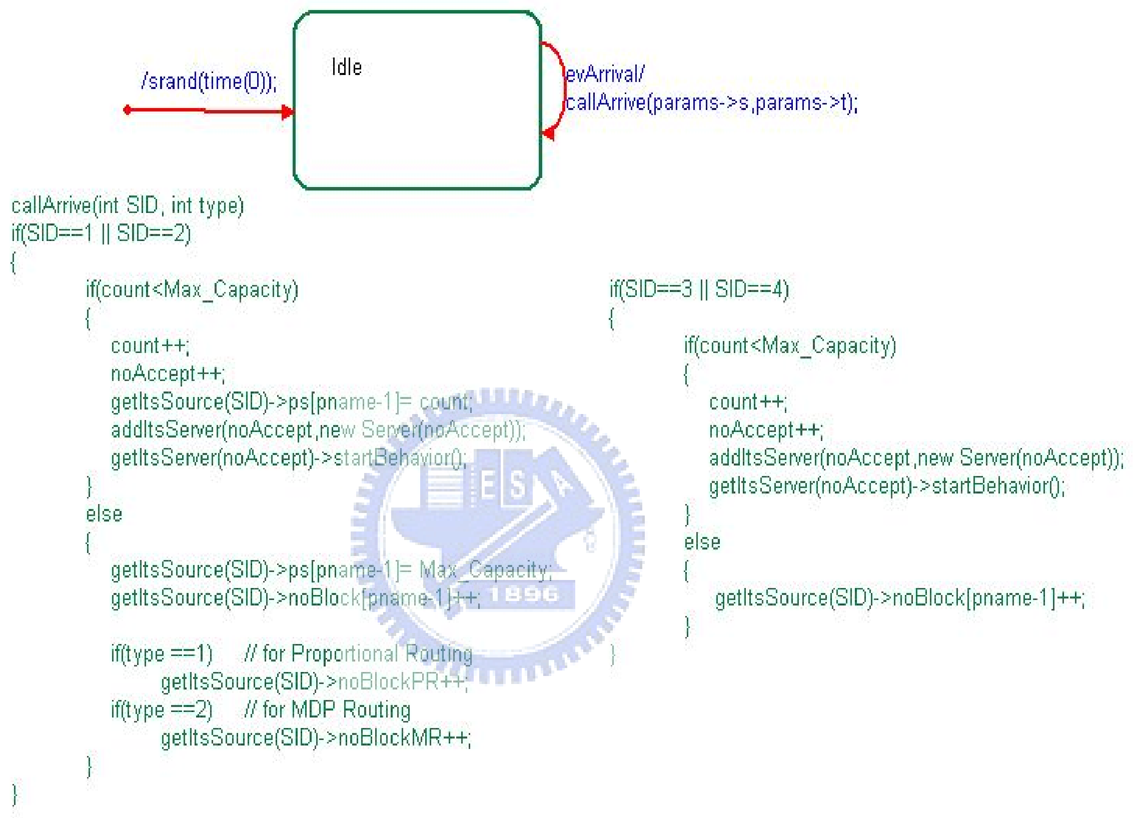
$$\delta^{(n)} = \frac{e^{bp\_PR^{(n)}}}{e^{bp\_PR^{(n)}} + e^{bp\_MR^{(n)}}} \tag{3.1a}$$

New weight for using MDP Routing:

$$1-\delta^{(n)} = \frac{e^{bp\_MR^{(n)}}}{e^{bp\_PR^{(n)}} + e^{bp\_MR^{(n)}}} \tag{3.1b}$$

And then source chooses one path proportionally based on the new weight $\delta^{(n)}$, path flow-proportion ($fp[\cdot]$), and path delta-cost ($dc[\cdot]$).

4.4.3 Path Operations



```
callArrive(int SID, int type)
if(SID==1 || SID==2)
{
        if(count<Max_Capacity)
        {
           count++;
           noAccept++;
           getItsSource(SID)->ps[pname-1]= count;
           addItsServer(noAccept,new Server(noAccept));
           getItsServer(noAccept)->startBehavior();
        }
        else
        {
           getItsSource(SID)->ps[pname-1]= Max_Capacity;
           getItsSource(SID)->noBlock[pname-1]++;

           if(type ==1)     // for Proportional Routing
                 getItsSource(SID)->noBlockPR++;
           if(type ==2)     // for MDP Routing
                 getItsSource(SID)->noBlockMR++;
        }
}
```

```
if(SID==3 || SID==4)
{
        if(count<Max_Capacity)
        {
           count++;
           noAccept++;
           addItsServer(noAccept,new Server(noAccept));
           getItsServer(noAccept)->startBehavior();
        }
        else
        {
           getItsSource(SID)->noBlock[pname-1]++;
        }
}
```

Fig. 19 Path Diagram of BDW-BP in UML

Different from path operations in BFW, path has to check the blocked call which is using Proportional Routing or using MDP Routing, and then informs source that generated this blocked call.

In summary, our simulator model could be illustrated in the sequence diagram below.

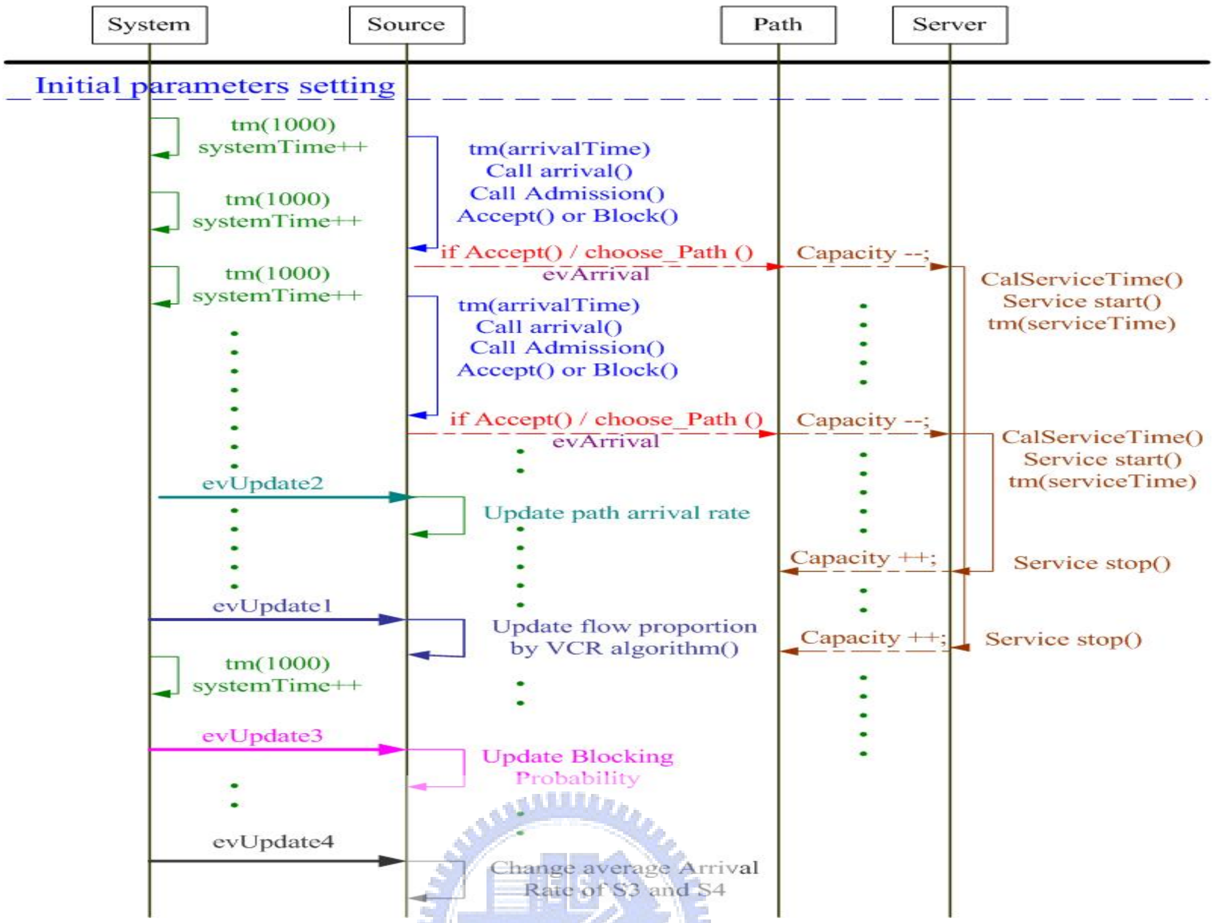


Fig. 20 Sequence diagram for BDW-BP in UML

4.4.4 Simulation Result

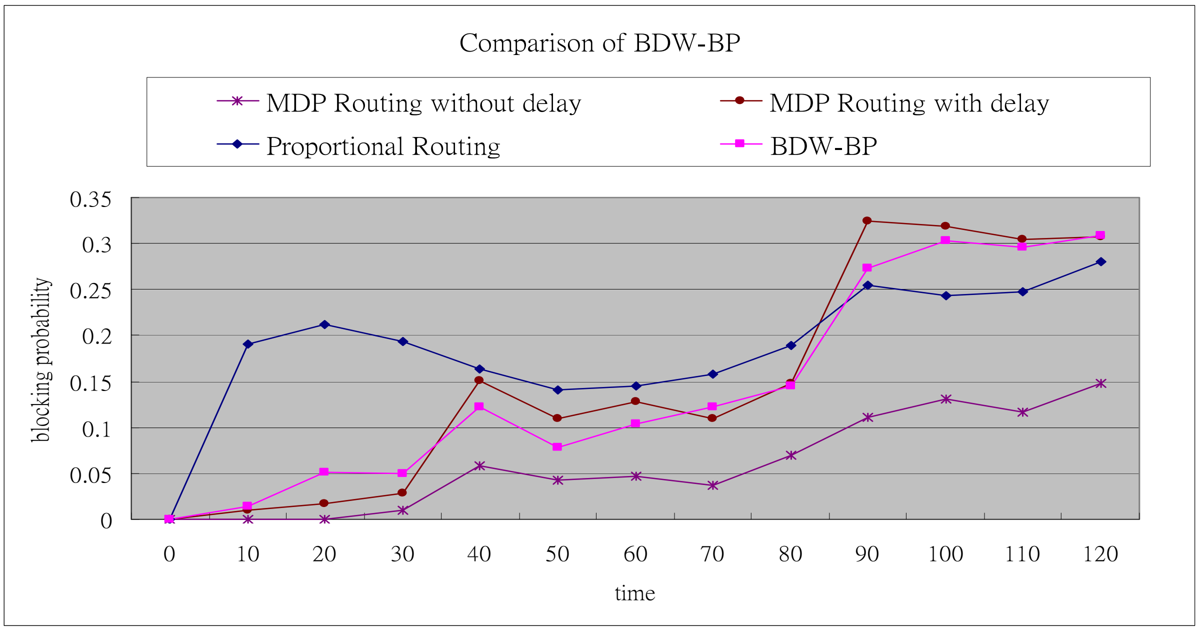


Fig. 21 Comparison of BDW-BP

The performances of “MDP Routing without delay”, “MDP Routing with delay” and “Proportional Routing” are the same as the results in Fig. 16. Consider the performance of BDW-BP, we find that the performance is comparable with the MDP Routing with delayed information, but is worse than the performance of Proportional Routing when the load is heavy.

4.5 Simulator Design and Results for BDW-DT

In this method, we dynamically boost the weight adjusted by the delay time of state information. We define this method as “Boosting with Dynamic Weight Adjusted by the Delay Time of State Information” (BDW-DT for shorted). The system operations and server operations in UML are the same as BFW.

4.5.1 Source Operations

The source diagram in UML is the same as the source operations in BFW. For each new call arrival, source has to calculate the delay-time ($dt[\cdot]$) for all feasible paths and proportionally selects one path using Eq. (3.2).

For path $i, i = 1,2,\ldots,K$, we calculate the path-combine-proportion $pcp[i]$

$$pcp[i] = (1 - e^{-\gamma \cdot dt[i]}) \cdot fp[i] + e^{-\gamma \cdot dt[i]} \cdot (1 - dc[i]) \tag{3.2}$$

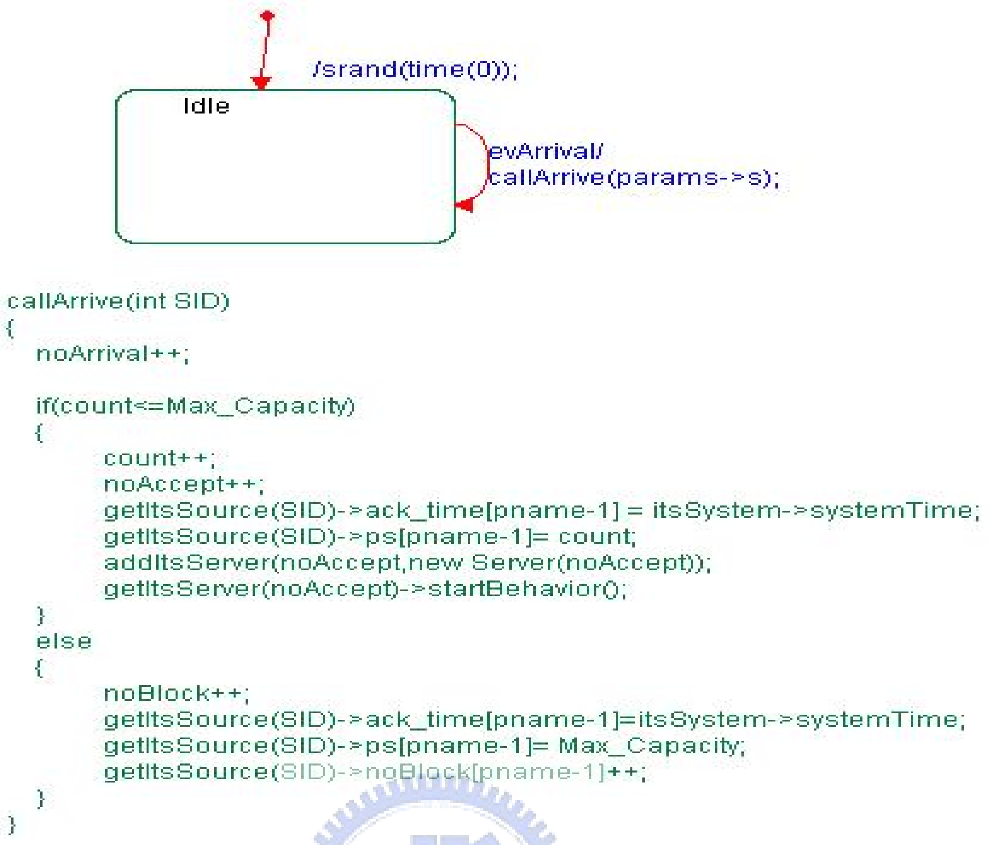
where γ is a configurable parameter, and

$dt[i]$ is the delay-time of this call for path i,

$dc[i]$ is the delta-cost of this call for path i using Eq .(2.21) in MDP routing

$fp[i]$ is the flow-proportion of path i using VCR algorithm in proportional routing,

4.5.2 Path Operations



```
callArrive(int SID)
{
   noArrival++;

   if(count<=Max_Capacity)
   {
          count++;
          noAccept++;
          getItsSource(SID)->ack_time[pname-1] = itsSystem->systemTime;
          getItsSource(SID)->ps[pname-1]= count;
          addItsServer(noAccept,new Server(noAccept));
          getItsServer(noAccept)->startBehavior();
   }
   else
   {
          noBlock++;
          getItsSource(SID)->ack_time[pname-1]=itsSystem->systemTime;
          getItsSource(SID)->ps[pname-1]= Max_Capacity;
          getItsSource(SID)->noBlock[pname-1]++;
   }
}
```

Fig. 22 Path diagram of BDW-DT in UML

No matter the call is accepted or blocked, path not only feedbacks the state information, but also asks source to record the *ack_time*[] for this path in order to calculate the delay-time of state information.

4.5.3 Simulation Result

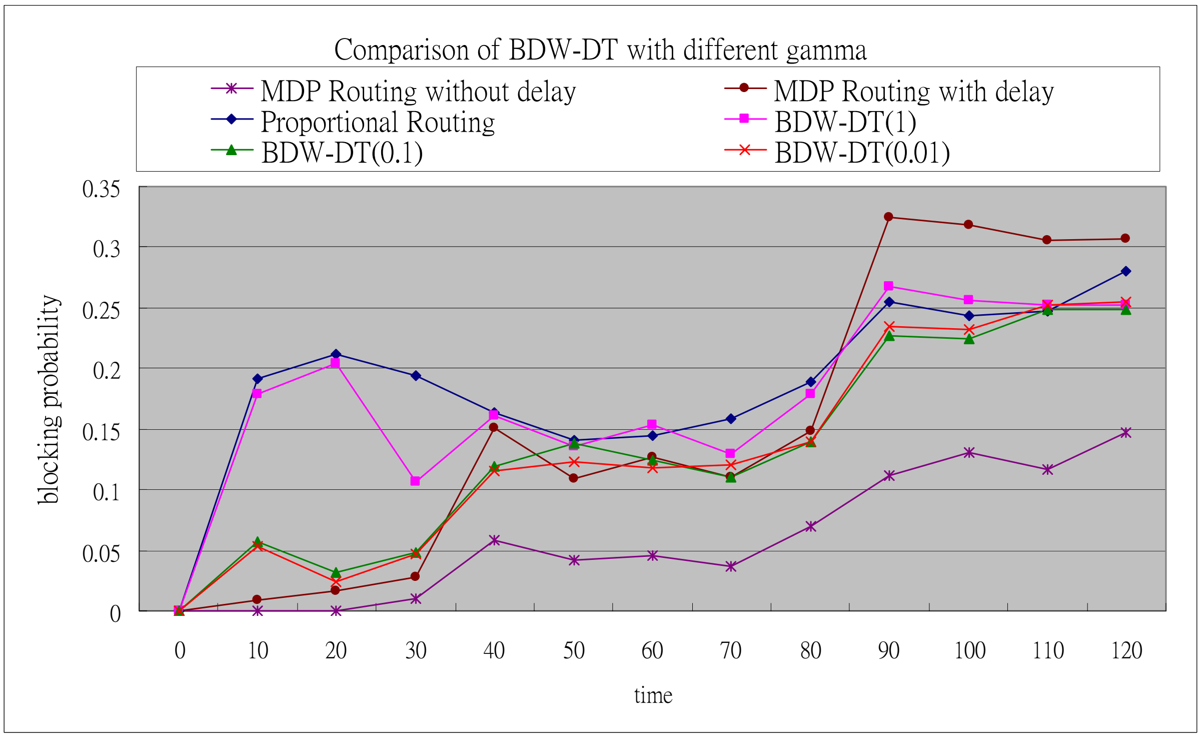


Fig. 23 Comparison of BDW-DT with different gamma (γ)

In Fig. 23, the performances of "MDP Routing without delay", "MDP Routing with delay" and "Proportional Routing" are the same as the results in Fig. 16. By changing the value of γ in Eq. (3.2), we find that the performances of "BDW-DT (0.1)" and "BDW-DT (0.01)" are better than "BDW-DT (1)", and are even better than "MDP Routing with delay" when the load is heavy. Consider Eq.(3.2).

$$pcp[i] = (1 - e^{-\gamma \cdot dt[i]}) \cdot fp[i] + e^{-\gamma \cdot dt[i]} \cdot (1 - dc[i]) \qquad (3.2)$$

When the load is heavy, the link state information is fresh enough. So we increase the weight $e^{-\gamma \cdot dt[i]}$ on "$1 - dc[i]$" and decrease the weight $(1 - e^{-\gamma \cdot dt[i]})$ on $fp[i]$. On the other hands, the link state information may be delayed, i.e. $dt[i]$ is long. So the bigger $e^{-\gamma \cdot dt[i]}$ compensates the uncertainty of "$1 - dc[i]$". No matter what the value of $dt[i]$ it is, we adaptively proportion flows in any situation using Eq. (3.2).

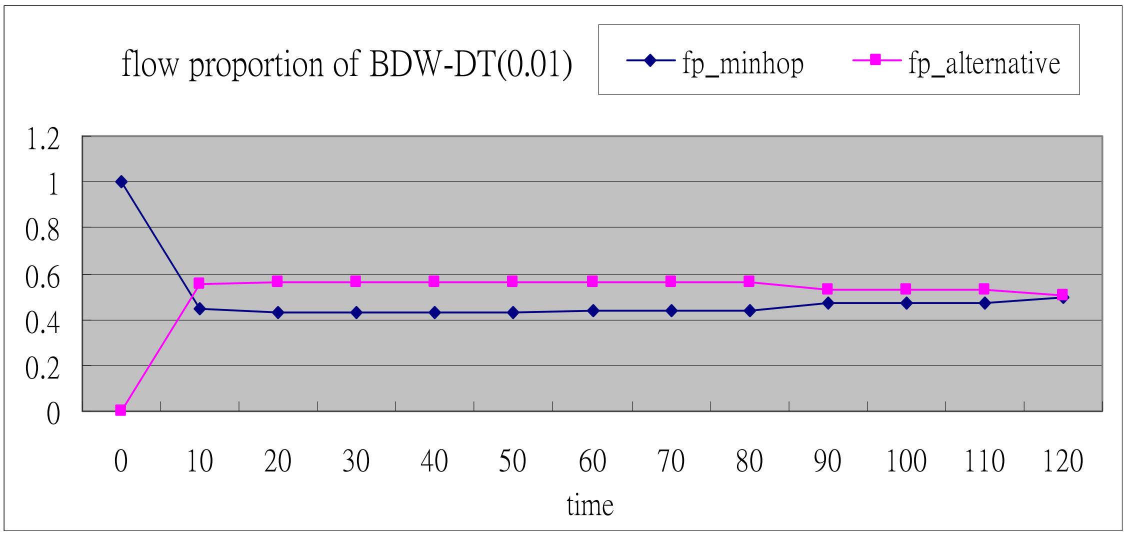


Fig. 24 Flow proportion of BDW-DT (gamma=0.01)

In Fig. 24, it illustrates that the flow proportion of min-hop paths and alternative paths. In the beginning, the flow proportion of min-hop paths is 0.998 and the flow proportion of alternative path is 0.002. With the increasing load of S_3 and S_4, the flow proportional of alternative paths increases because the blocking probability of min-hop paths is higher than alternative paths. And when the blocking probability of alternative path is increasing with the increasing load of S_3 and S_4, the system starts decreasing the proportion of alternative in order to equalize the overall blocking rate.

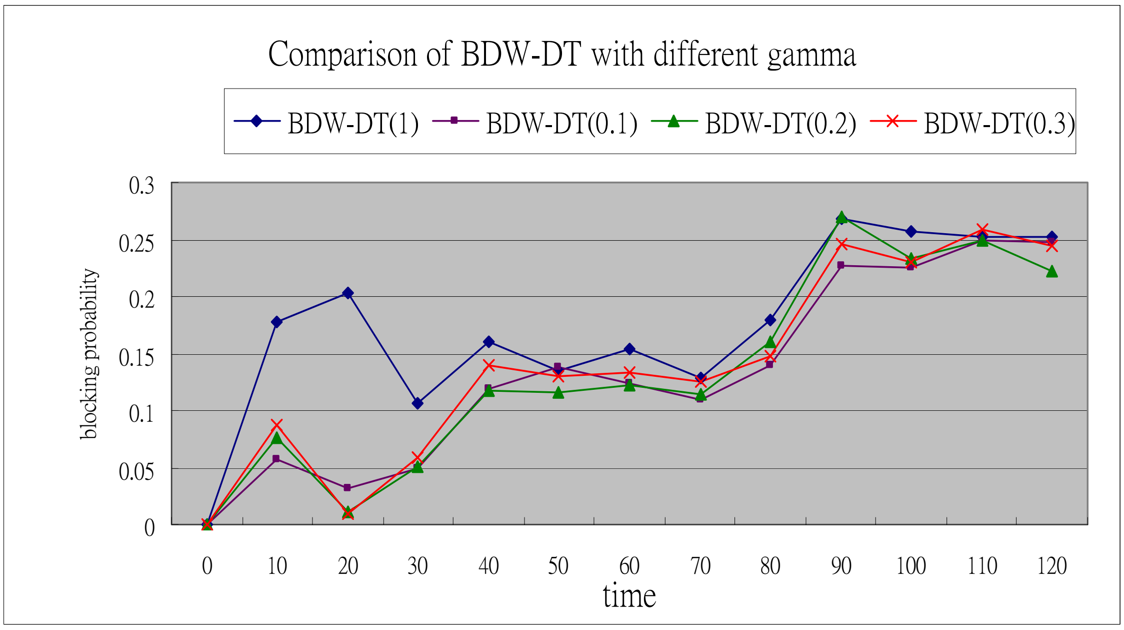


Fig. 25 Comparison of BDW-DT with different gamma (γ)

In Fig .25, we compare four scenarios where gamma (γ) in Eq. (3.2) is equal to 1, 0.1, 0.2, and 0.3, respectively. From the simulation results, we find that the performance is not necessarily getting better if we decrease γ. We consider that the performance is not only related to γ but also to the load of the network.

Finally, we compare our proposed three methods with MDP Routing and Proportional Routing in Fig. 26. We find that these three methods indeed boost MDP Routing and Proportional Routing in some network situation.

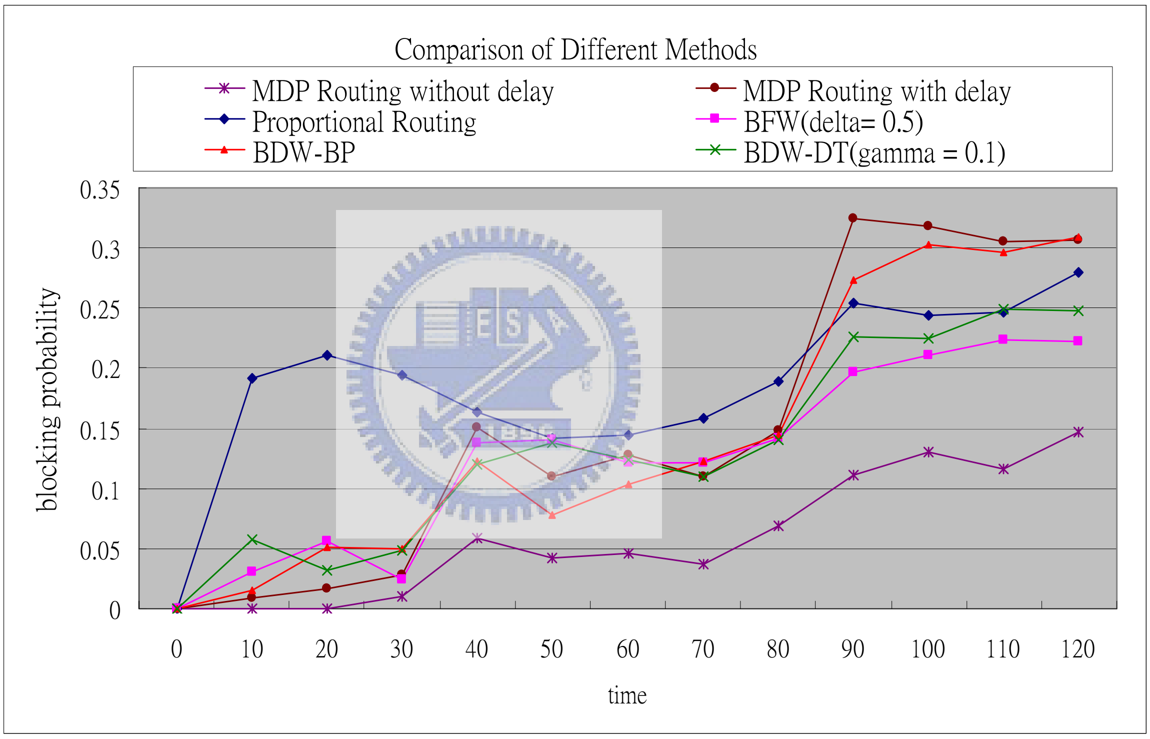


Fig. 26 Comparison of all methods

Chapter 5
Conclusion

In this thesis, we first compare the state-dependent separable routing and adaptive proportional routing with different update interval and find that the latter works significantly worse than the former if the network state information obtained in the source is delayed not more than one minute. So we propose three possible boosting methods to compensate the shortcoming of proportional routing and state-dependent routing. As shown in Fig. 26, when the load is light, the performance of BDW-BP is better than other boosting methods. But as the load increases, we find that the performance of BFW is getting better and even better than MDP Routing with delayed state information when the load is heavy. Therefore, these three boosting methods are load-dependent and none of them can outperform others in any network situations.

Our future work will be conducting the simulation on the more complicated network model where the call arrival rates for most of the source-destination pairs are less frequent. We believe in such a model the network state tends to be more obsolete whereby configuring the weight on the state-dependent separable routing will become a more important issue.

References

[1] R. A. Howard, Dynamic Programming and Markov Processes

[2] T. J. Oti, and K. R. Krishnan, "Separable Routing: A Scheme for State-Dependent Routing of Circuit Switched Telephone Traffic", Beilcore. 445 South Street. P.O. Box 1910. Morristown. NJ 07960-1910. USA

[3] S. Nelakuditi, Zhi-Li Zhang, R. P. Tsang, and David. H.C. Du, "Adaptive Proportional Routing: A Localized QoS Routing Approach", IEEE/ACM Transactions on Networking, VOL.10, NO.6, DECEMBER 2002

[4] S. Nelakuditi and Zhi-Li Zhang, "A Localized Adaptive Proportioning Approach to QoS Routing", IEEE Communications Magazine June 2002

[5] E. Rosen, A. Viswanathan, and R. Callon, "Multi-Protocol Label Switching Architecture", Internet Draft draft-ietf-mpls-arch-06.txt, August1999, work in progress.

[6] R. B. Cooper, Introduction to Queuing Theory, Edition 2, North-Holland 1981.

[7] L. L. Peterson and B. S. Davie, Computer Networks, A System Approach, Edition 3.

[8] S. Chen and K. Nahrstedt, "An overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions" IEEE Network Magazine, vol. 12, pp.14-79, Nov.-Dec. 1998.

[9] G. Apostolo, "Quality of Service Based Routing: A Performance Perspective", ACM SIGCOMM, 1998.

[10] M. Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees", IEEE ICNP, 1997.