Service Replication Strategy in Service Overlay Networks

LIU Yunkai

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

May 2004

Abstract

The service overlay network (SON) is an effective means to deploy end-to-end QoS guaranteed content delivery services on the current Internet. We extend the SON cost model and introduce a new *service delivery tree* model. The original cost model only allows a SON operator to control the bandwidth provisioning, which is a static resource management scheme based on previous traffic patterns. In our replication model we propose dynamical adaptation to traffic variation by adding replicated servers on the internal nodes of SON, in this way, the resources of the SON can be efficiently utilized. We propose a service replication strategy to maximize the total effective throughput as well as minimize the QoS violation penalty of the SON. We also present both centralized and distributed algorithms for the optimal placement of the replicated servers. Experiments are carried out to quantify the merit, effectiveness and the scalability of the proposed distributed algorithm service replication. In particular, the results obtained by our algorithm is very close to the optimal. The algorithms retain good performance even when we scale up the network size.

摘要

服務覆蓋網絡(Service Overlay Network)是一種在互聯網中能夠有效提供具備端到端服務質量保證的內容傳送服務方法。在本論文中，我們把服務覆蓋網絡的成本模型加以擴展並引入一個新的概念—服務傳送樹模型。舊有的成本模型只考慮了覆蓋網絡管理者控制網絡帶寬的供應問題，而這是一種基於網絡歷史流量的靜態管理方式。在我們的服務複製模型中我們在覆蓋網絡內部節點上增加複製服務器，以此達到動態地適應網絡流量的變化，這樣便能更充分合理地利用覆蓋網絡的資源。我們的複製模型策略能夠最大化覆蓋網絡的有效輸出以及最小化違反服務質量保證時的處罰。我們還提出了集中式和分布式算法來解決最優化的複製服務器的位置選擇。我們做了一些實驗來量化我們提出的服務複製算法的可行性，有效性和擴展性。結果證明我們的算法非常接近最優化的結果，並且當網絡中節點數成備擴大時我們的算法仍然保持很好的效率。

Acknowledgment

I would like to thank my supervisor [illegible] for her guidance and support throughout my study [illegible]. [illegible] me would be [illegible] for long future.

Acknowledgment

I would like to thank my supervisor Professor John C.S. Lui for his guidance and support throughout my study these years. His influences to me would be my priceless life-long fortune.

Contents

List of Figures

List of Tables

List of Publications

Part of this research work appeared in the following publications:

- Kevin Y.K. Liu, John C.S. Lui, and Zhi-Li Zhang. Service Replication Strategy of Service Overlay Networks. In *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Seoul, Korea. 2004

- Kevin Y.K. Liu, John C.S. Lui, and Zhi-Li Zhang. Distributed Algorithm for Service Replication in Service Overlay Network. In *Proceedings of the Third International IFIP TC-6 Networking Conference (Networking2004)*, Athens, Greece. 2004

Chapter 1

Introduction

The Internet is being used for many different user activities, including emails, software distribution, video/audio entertainment, e-commerce, and real-time games. Although some of these applications are designed to be adaptive to available network resources, they still expect good level of services from the network, for example, low latency and low packet loss, so as to deliver the desired performance at the application layer. However, the primary service provided by the Internet is the *best-effort* service model which does not perform any service differentiation, therefore, end-to-end quality-of-service (QoS) guarantees are difficult to maintain. Another reason for the difficulty in providing end-to-end QoS guarantees is that the Internet is organized as many different autonomous systems (**AS**s) wherein each AS manages its own traffic, performance guarantees and internal routing decisions. These autonomous systems also have various bilateral business relationships (e.g., peering and provider-customer) for traffic exchange so as to maintain the global connectivity of the Internet. For many network applications, the data traffic usually traverses across multiple autonomous systems, and it is difficult to establish a *"multi-lateral"* business relationship which spans many autonomous systems. Therefore, network services which need end-to-end QoS guarantees are still far from realization and the above mentioned problems hinder the deployment of

many QoS sensitive services on the Internet.

In [7], the authors advocate the notion of *service overlay network* (SON) as an effective means to address the problems of providing end-to-end services. A SON is an overlay network that spans many autonomous systems. In general, a SON purchases bandwidth with certain QoS guarantees from all **AS**s that the overlay network spans. This way, a logical end-to-end service delivery infrastructure can be built on top of the existing network infrastructure. On this logical service overlay network, one can provide different types of time sensitive services, such as video-on-demand, Internet radio and television, VoIP, ..., etc. A SON offers these services to different users who pay the SON for using these value-added services.

The ultimate goal of the service overlay network is to maximize its revenue and minimize the operating cost. In some previous works [7, 16], the authors formulate this problem as bandwidth provisioning model, wherein the revenue of SON comes from the fees paid by users and the costs consist of bandwidth provisioning cost and the QoS violation penalties. The goal is to optimally provision the bandwidths of SON such that the net profit is maximized. However, one important point to observe is that once the bandwidth provisioning is carried out, the overlay network is *committed* to a topology wherein each link in the overlay network has a *fixed* bandwidth capacity. This capacity of each link remains unchanged until the next bandwidth provisioning instant[1] .

In general, the time scale of bandwidth provisioning can be in terms of weeks or months. Since traffic/service demand is time varying and stochastic in nature, it is possible that there will be a sudden surge on traffic due to some unexpected event (e.g., a popular pay-per-view sport or musical event). This type of traffic surge may not be well-represented or characterized in the

[1] In[7], the authors also address the dynamic bandwidth provisioning problem, however it is technically difficult to implement[10].

original measured traffic distribution that was used for the bandwidth provisioning process. In this case, the allocated bandwidth for the SON may not be sufficient to provide the end-to-end QoS guarantees. This translates to lower profit for the SON operator since the operator needs to pay for the penalty for these QoS violations.

It is important to point out that many time-sensitive services provided by the SON are in the one-to-many format, for example, services such as video-on-demand and multi-players on-line games, wherein one "logical" server needs to support many users of the overlay network. As shown in [14], to delivery this type of service, a data delivery process is usually in the form of a tree topology. When the user demands increase, some links of the delivery tree could be overloaded or even congested. Instead of delivering a low quality of service over these congested links, (i.e. reduction in profit of SON), we propose to *dynamically replicate* services on the service gateways of SON so as to reduce the QoS penalty as well as increase the effective throughput of the SON. The problem of service replication along the delivery tree is to choose among a set of service gateways to place the additional server for service replication such that the total profit can be maximized.

In Chapter 2, we present the necessary background of service overlay networks. In Chapter 3, we provide our mathematical model for service replication as well as the distributed evaluation of effective throughput for a service delivery tree. We then present both centralized and distributed algorithm for service replication in Chapter 4. In Chapter 5, we illustrate the numerical experiment results and show the effectiveness of our algorithm. Finally we conclude our thesis in Chapter 7.

Chapter 2

Background

2.1 Notations

For the remaining of this section, we use the notations depicted in Table 2.1 to describe the bandwidth provisioning problem.

Parameter	Remarks
$\mathcal{G}$	A connected graph $\mathcal{G} = \{\mathcal{N}, \mathcal{L}\}$ representing a SON where $\mathcal{N}$ and $\mathcal{L}$ are the set of nodes and the set of links of the SON respectively.
$\mathcal{R}$	the set of all source-destination (SD) paths (or the traffic requirements) in $\mathcal{G}$.
ρ_r	a non-negative random variable denoting the bandwidth requirement for a SD path $r \in \mathcal{R}$.
ρ_l	a non-negative random variable representing the amount of traffic flow on link $l \in \mathcal{L}$. $\rho_l = \sum \rho_r, \forall r$ passes through link l.
$\bar{\rho}_l$	the average amount of traffic flow (in Mbps) on link $l \in \mathcal{L}$.
c_l	allocated capacity (in Mbps) on link $l \in \mathcal{L}$.
$\Phi_l(c_l)$	cost per unit of time for reserving c_l amount of bandwidth for link $l \in \mathcal{L}$.
e_r	revenue for carrying one unit of traffic flow along a SD pair $r \in \mathcal{R}$.
π_r	penalty of QoS violation for one unit traffic flow on SD pair $r \in \mathcal{R}$.

Table 2.1: Notations used for describing SON and the bandwidth provision problem.

2.2 Service Overlay Network Architecture

A SON $\mathcal{G}$ is a logical overlay network with a set of nodes $\mathcal{N}$ and a set of links $\mathcal{L}$. Each node in $\mathcal{N}$ is a *service gateway* which performs service-specific data forwarding and control functions. One can view a service gateway as a physical end host on the Internet, for example, a server which is controlled and managed by the SON operator. A link in $\mathcal{L}$ is a *logical* connection between two service gateways and the link is a network layer path provided by the underlying autonomous systems. The advantages of the SON architectural framework are: 1) one can purchase different bandwidth for different links in the SON and, 2) one can bypass congested peering points among **AS**s and thereby provide end-to-end QoS guarantees. Figure 2.1 illustrates the SON architecture.

When a user requests for a specific QoS guaranteed service, it will connect to the SON through its own network domain and its request will be forwarded to the proper service gateway. As shown in Figure 2.1, there are two source-destination pairs (a, b) and (a, c), and their traffic will go through different logical links of the SON.

The advantage of the SON architecture is that it decouples the application services from the network services [17] and thereby reduces the complexity of network control and management. Meanwhile, the SON can provide more diverse end-to-end QoS guaranteed services to satisfy the needs of its users.

2.3 The SON Cost Model

The formal mathematical framework for performing the bandwidth provisioning can be described as follows. We are given a network topology $\mathcal{G}$, the source-destination (SD) path requirements $\mathcal{R}$, the stochastic traffic demand

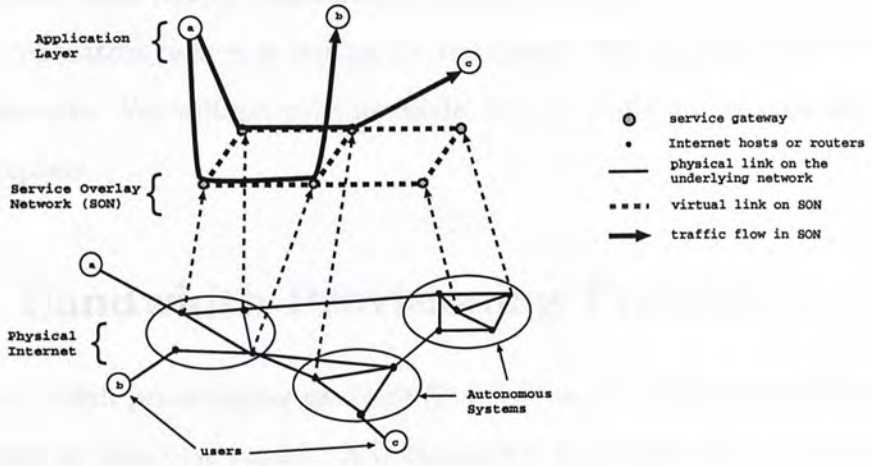


Figure 2.1: An example of a service overlay network

$\{\rho_r\}$ for each $r \in \mathcal{R}$, and the routing method. Assume that the traffic demand distribution on path r is known[1] , then the total net income for the SON, denoted by the random variable W, can be expressed as:

$$W(\{\rho_r\}) = \sum_{r\in\mathcal{R}} e_r\rho_r - \sum_{l\in\mathcal{L}} \Phi_l(c_l) - \sum_{r\in\mathcal{R}} \pi_r\rho_r B_r(\{\rho_r\}). \tag{2.1}$$

where $\sum_{r\in\mathcal{R}} e_r\rho_r$ is the total revenue received by a SON for carrying $\{\rho_r\}$ traffic along the SD path $r \in \mathcal{R}$; $\sum_{l\in\mathcal{L}} \Phi_l(c_l)$ is the total bandwidth cost that a SON must purchase from all its underlying autonomous systems; $\sum_{r\in\mathcal{R}} \pi_r\rho_r B_r(\{\rho_r\})$ is the total *penalty* that a SON suffered when the QoS guarantees for those traffic demands are violated. The variable B_r represents the probability that QoS guarantees for the SD pair r is violated.

Under the *independent link* assumption, i.e. the statistics of traffic flow on different links are independent from each other, we can further expand B_r as:

$$B_r(\{\rho_r\}) = 1 - \prod_{l\in r} (1 - B_l(\{\rho_l\})) . \tag{2.2}$$

[1]This traffic demand distribution can be obtained through long-term observation or measurement of past traffic history.

This net profit model summarizes the major concerns of the service overlay network operators, which is crucial for the design and implementation of any SON networks. We will extend this model for our problem formulation in the later chapters.

2.4 Bandwidth Provisioning Problem

The Bandwidth provisioning problem [7, 16, 8] is one of many problems that are related to this cost model. It is important in designing a service overlay network and closely related to our work. We thus briefly explain it here.

To guarantee the delivery of end-to-end services, the SON needs to purchase sufficient amount of bandwidth from different **AS**s on each link $l \in \mathcal{L}$ so that QoS guarantees can be maintained. The *"bandwidth provisioning problem"* for a SON is to determine the appropriate amount of bandwidth to purchase for each link in $\mathcal{L}$ from the underlying **AS**s, so that the QoS sensitive traffic demand for any source-destination pair in $\mathcal{R}$ can be satisfied and at the same time, the total net profit of the SON is maximized.

The problem of bandwidth provisioning can thus be formulated as to determine the appropriate amount of capacity $\{c_l\}$ for each link $l \in \mathcal{L}$ such that the expected total net profit $E(W)$ is maximized:

$$\max_{c_l} E(W). \tag{2.3}$$

The exact solution to this optimization problem is generally difficult to obtain because the objective function depends on both the particular forms of the traffic demand distribution $\{\rho_r\}$, as well as the service QoS violation probability B_r. In [7], the authors derived a lower bound of the $E(W)$ by introducing an additional variable: define a very small real number δ, for each SD pair r, define $\hat{\rho}_r > \bar{\rho}_r$ such that $\int_{\hat{\rho}_r}^{\infty} \rho_r d\rho_r \leq \delta$. i.e., $\text{Prob}\{\rho_r \geq \hat{\rho}_r\} \leq \delta/\hat{\rho}_r$.

This basically says that $\hat{\rho}_r$ is such that the probability the traffic demand along the SD path r exceeds $\hat{\rho}_r$ is very small, and thus negligible. Then, $E(W)$ is lower bounded by $V(\mathcal{L}, \mathcal{R})$, wherein:

$$\begin{aligned} V(\mathcal{L}, \mathcal{R}) &= \sum_{r \in \mathcal{R}} e_r \bar{\rho}_r - \sum_{l \in \mathcal{L}} \Phi_l(c_l) - \sum_{r \in \mathcal{R}} \pi_r \bar{\rho}_r B_r(\hat{\rho}_r) \\ &\quad - \sum_{r \in \mathcal{R}} \pi_r \delta_r (1 + \sum_{r' \neq r} \frac{\bar{\rho}_r}{\hat{\rho}_{r'}}). \end{aligned} \tag{2.4}$$

One can compute optimal bandwidth allocation c_l^* for all link $l \in \mathcal{L}$, such that $\frac{\partial V}{\partial c_l} = 0$. This way, one can find the optimal bandwidth to be provisioned of each link $l \in \mathcal{L}$ which maximizes $V(\mathcal{L}, \mathcal{R})$, the least average profit for a service overlay network.

2.5 Traffic Variation and QoS Violation Penalty

Note that the previously mentioned bandwidth provisioning method is only practical in an *off-line manner*. That is, once bandwidth is provisioned, it cannot be changed until the next bandwidth provisioning instant. However, due to difficulties in implementation and in adjusting the multi-lateral agreements, the period between two bandwidth provisioning instants may be long (e.g., weeks or even months). During this period, the traffic demand of a SON could fluctuate. This is especially true for a SON that spans a large geographical area wherein the time-of-day effect is significant, e.g., some part of the network is congested during rush hours, while other part of network is very lightly loaded because it is at a different time zone. Also, it is possible that there may be a surge in traffic demand due to some unexpected events, e.g., a popular pay-per-view sport or musical event that attracts many users. The variation of traffic flow will increase the QoS violation probability B_r. Therefore, it is crucial for the SON to have the adaptive capability to traffic

flow fluctuation. In this paper, we propose to *dynamically replicate services* within a SON so as to reduce the traffic demands on "overloaded" links and to maximize the net income of an SON operator.

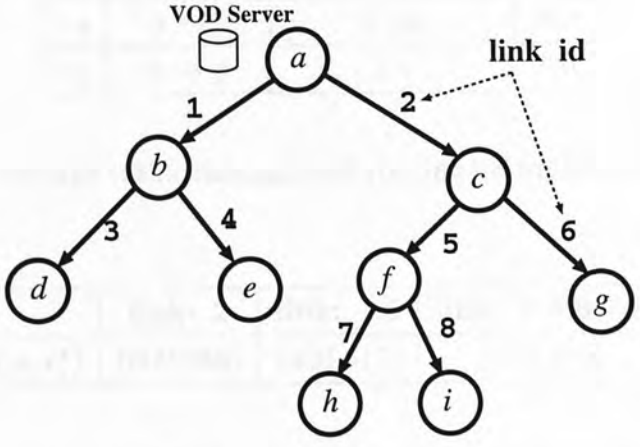


Figure 2.2: An illustration of the traffic fluctuation on a SON

To illustrate the reduction of profit due to traffic fluctuation, consider a SON with a tree topology as depicted in Figure 2.2. Node a is a service gateway with a video-on-demand service and it has five sets of clients[2] at nodes d, e, g, h and i. The average traffic demand of these five clients are 200 unit (Mbps) each. The source-destination path $\mathcal{R}$ and routing information are illustrated in Table 2.2.

Using the static bandwidth provisioning model in [7], the capacity to be provisioned for each link l is depicted in Table 2.3.

Now suppose that there is a popular pay-per-view movie which increases the traffic demands along the traversed links. This increase in traffic demand will cause some links to be overloaded and thus the SON will suffer from the QoS penalty and a reduction in the total profit. Figure 2.3 illustrates the

[2]A set of client may represent many users within the same network domain.

	Src-Dest	**Routing Path**	$\bar{\rho}_r$
r_1	$a \rightarrow d$	1-3	200
r_2	$a \rightarrow e$	1-4	200
r_3	$a \rightarrow h$	2-5-7	200
r_4	$a \rightarrow i$	2-5-8	200
r_5	$a \rightarrow g$	2-6	200

Table 2.2: The average traffic demand and routing information for all SD paths $r \in \mathcal{R}$.

	link: 2	**link: 1,5**	**link: 3,4,6,7,8**
$(\bar{\rho}_l; c_l^*)$	(600,888)	(400,617)	(200,310)

Table 2.3: The average traffic demand $(\bar{\rho}_l)$ and bandwidth provisioned (c_l^*) for each link $l \in \mathcal{L}$.

reduction in the expected net income when the traffic demands from all these five sets of clients are increased uniformly by some small percentages.

From the figure, one can observe that the total profit of a SON will decrease by as much as 40% when the traffic demand increases by only 20%. Since one cannot change the allocated link capacity until the next provisioning instant, instead of suffering from this traffic overload, we propose to dynamically replicate services within the SON. In the following two chapters, we describe the replication strategy to reduce the traffic loading.

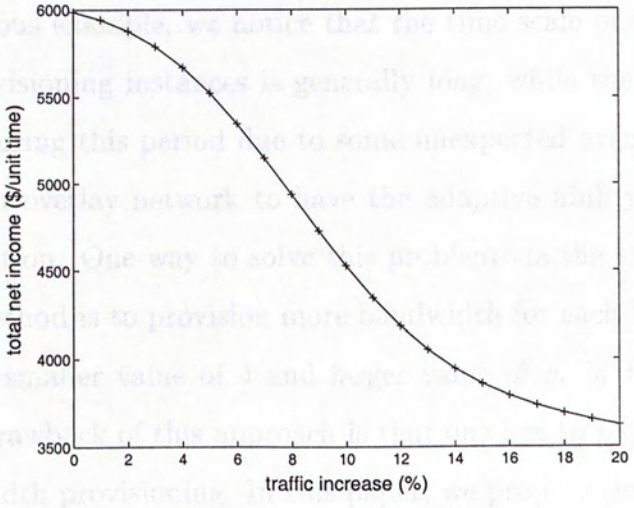


Figure 2.3: Reduction of total profit for a SON when traffic increases

Chapter 3

Service Replication Model

From the previous example, we notice that the time scale of two consecutive bandwidth provisioning instances is generally long, while the traffic demand could change during this period due to some unexpected events, therefore, it is crucial for an overlay network to have the adaptive ability to such traffic demand fluctuation. One way to solve this problem via the static bandwidth provisioning method is to provision more bandwidth for each link in the SON (e.g., having a smaller value of δ and larger value of $\hat{\rho}_r$ in Equation (2.4)). However, the drawback of this approach is that one has to pay a much higher cost for bandwidth provisioning. In this paper, we propose service replication approaches that make the SON more flexible and adaptive to traffic variation without purchasing extra bandwidth resource from the underlying autonomous systems.

Note that the *service gateway* inside a SON is a network host managed by the SON operator. The service gateway has sufficient storage and processing power to perform the basic packet forwarding function as well as some service-specific functions (e.g., video-on-demand service). The replication strategies make use of these service gateways and extend their functionalities. Therefore, each service gateway can be a potential server and deliver the content to users in the SON.

3.1 One-to-One Service Model

Let us start with a simple case of a single source, single destination service replication model. Given the source-destination paths in $\mathcal{R}$, the stochastic traffic demands $\{\rho_r\}$ for all $r \in \mathcal{R}$, one has to choose a set of demands in $\mathcal{R}$ to replicate. An SD path $r \in \mathcal{R}$ consists of a source node s_r, a destination node d_r, and its stochastic traffic demand ρ_r along the path r. It is important to point out that a destination node may consists of a large number of users, i.e., a set of users within the same network edge who wants to receive a video-on-demand service.

In the following context, we use γ to denote one *replication* event. We also introduce the following notations:

$loc(\gamma)$	the node which γ chooses to install the replicated service.
$target(\gamma)$	the SD path that γ chooses to replicate for.
$path(\gamma)$	the new path taken by γ to deliver the replicated service.
$\beta(\gamma)$	the fraction of traffic shift from $target(\gamma)$ onto $path(\gamma)$.

Consider Figure 3.1, suppose the replication event γ is for a SD path $r \in \mathcal{R}$ and we choose node $i \in \mathcal{N}$ to install the replicated service, then $target(\gamma) = r$ and $loc(\gamma) = i$. Let the average traffic demand on r be $\bar{\rho}_r$. After the replication process, the traffic demand on r will decrease because $loc(\gamma)$ is serving some of the clients in r. Therefore, the average traffic demand on r after the replication process is $\bar{\rho}_r(1 - \beta(\gamma))$. The replication process γ will create a new $path(\gamma)$ with source node in $loc(\gamma)$ and destination node in d_r for the replicated service. The traffic on this new path needs to deliver, on the average, $\beta(\gamma)\bar{\rho}_r$ amount of traffic to a set of users in r.

Let $\mathcal{D}$ denote the set of all replication events γ. Let $\mathcal{R}'$ denote the set of all

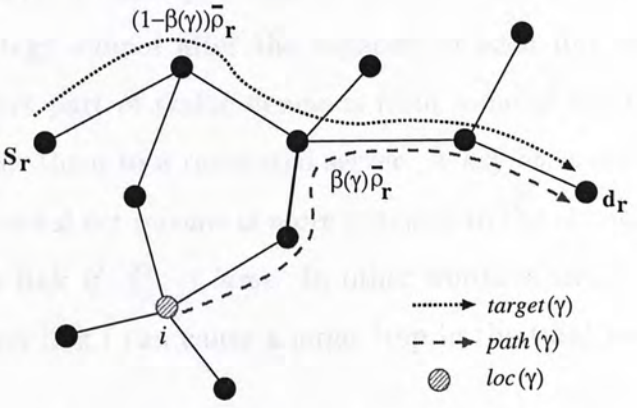


Figure 3.1: Illustration of replication event in a SON.

source-destination paths of the SON after the replication events $\mathcal{D}$. The single source, single destination replication is to find a set of replication events $\mathcal{D}$ which maximizes the increase in (the lower bound V of) the total net income $E(W)$ of SON by performing service replication, i.e., to maximize the following objective function:

$$\max_{\mathcal{D}} V(\mathcal{R}') - V(\mathcal{R})$$

subject to:

$$loc(\gamma) \in \mathcal{N}$$

$$target(\gamma) \in S_{loc(\gamma)}$$

$$0 \leq \beta(\gamma) \leq 1$$

Since the replication will not change the sum of all the traffic demands for the SD paths in $\mathcal{R}$ and the total bandwidth cost, we have,

$$\begin{aligned} V(\mathcal{R}') - V(\mathcal{R}) &= \sum_{r\in\mathcal{R}} \pi_r \bar{\rho}_r B_r(\hat{\rho}_r) - \sum_{r\in\mathcal{R}'} \pi_r \bar{\rho}_r B_r(\hat{\rho}_r) \\ &+ \sum_{r\in\mathcal{R}} \pi_r \delta_r \left(1 + \sum_{r'\neq r} \frac{\bar{\rho}_r}{\hat{\rho}_{r'}}\right) - \sum_{r\in\mathcal{R}'} \pi_r \delta_r \left(1 + \sum_{r'\neq r} \frac{\bar{\rho}_r}{\hat{\rho}_{r'}}\right) \end{aligned} \tag{3.1}$$

The motivation of our replication is as follows. Note that although the replication strategy *cannot* alter the capacity of each link in SON, it may change and divert part of traffic demands from some of the *highly congested* links and redirect them to a replicated server. A *key observation* is that for a given link l, the total net income is *more sensitive* to the change of total traffic demand on this link *if* $-\frac{\partial V}{\partial \rho_l}$ *is large.* In other words, a small decrease in the traffic demand on link l can cause a large drop in the total net income of the SON.

Therefore we focus on those links for which $-\frac{\partial V}{\partial \rho_l}$ is large and attempt to reduce the traffic demands on these links by service replication. In deciding which path r to select for service replication, we use the analogy from the optimal routing problem [1] and introduce the following notion.

Definition 3.1 Let a path r having $n \geq 1$ links $l_1, l_2, \ldots, l_n$. The "negative first derivative sum" (NFDS) of the path r is

$$\text{NFDS}(r) = -\sum_{i=1}^{n} \frac{\partial V}{\partial \rho_{l_i}}.$$

In deciding which path to replicate, we choose a path r that has the *most negative* NFDS value.

To determine which node (i.e., service gateway) to place the replication, we adopt the following strategy. For all the traffic going to a certain user, they must go through the link connecting that user to the SON (the "last-mile" link). So to place a replication whose target is path r, we only consider *those nodes along path* r. The rationale for this approach is that if one sets up replication on nodes not along the path in r, it will increase the traffic demands on other links (which may in turn increase the probability of violating the QoS requirements for those links). Therefore, our replication strategy only targets those nodes along path r.

3.2 Service Delivery Tree Model

Real-time content delivery is one of the major applications of SON, many QoS sensitive services can be deployed on the SON's infrastructure, such as VOD, Internet radio and television. However, most time it doesn't fit into the single-source single-destination model. As illustrated in [14], the optimal data delivery topology for these applications is a tree topology. Therefore, in the rest of the paper, we will focus on this tree model, and we name it as *service delivery tree* (SDT) model.

The root node of SDT is an application level service gateway, for example, a VOD server wherein the continuous media are stored. All the leaf nodes of a SDT are client nodes which are the access points for different users within the same network domain. The formulation of delivery tree can be different for different applications [14, 11, 4, 6], however, our model is generic for any tree topology.

To illustrate, consider an example in Figure 3.2 where node r is the root (or server) node; the darkened nodes are the client nodes of the SDT, thick lines represent links of the SDT.

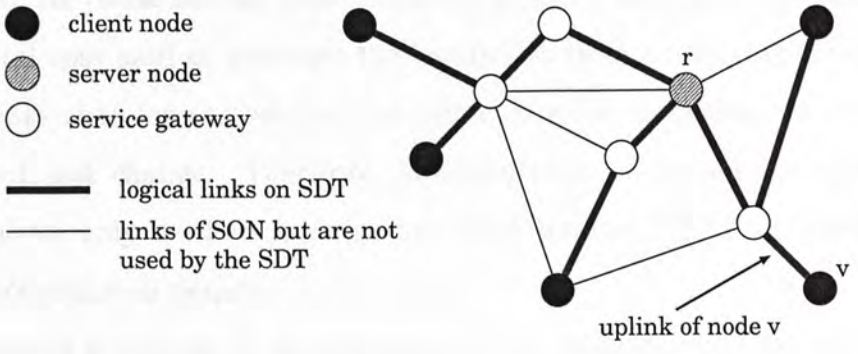


Figure 3.2: Illustration of a service delivery tree (SDT) on a SON

3.2.1 Problem Formulation

To formally describe the service replication problem, let us first define the following notations:

$\mathcal{T}$	the service delivery tree.
T_u	subtree of $\mathcal{T}$ rooted at node u.
S_u	all child nodes of u.
ρ_u	traffic demand from node u.
D_u	total traffic demand from all client nodes of subtree T_u.
c_u	bandwidth of the uplink of node u.
$q_u(D_u, c_u)$	probability of QoS guarantee on the uplink of node u, given that the bandwidth is c_u and traffic demand is D_u.
$F(T_u)$	total effective throughput of subtree T_u.

Table 3.1: Notations of Service Delivery Tree

In the original SON bandwidth provisioning model [7, 8], the QoS violation on link l is defined as $\rho_l B_l(\rho_l)$ where B_l is the QoS *"violation"* probability. In our service replication problem on SDT, we use an alternative metric. In the original model, the first two terms of the objective function in Eq. (2.1) are the total revenue (total income leveraged from all users) and the total bandwidth cost (total cost paid to purchase the bandwidth from underlying ASs) of the SON. Note that for the service replication process, the values of these two terms will not change. Therefore, in formulation of the service replication problem, we only need to focus on the third term of Eq. (2.1), namely, the total QoS violation penalty.

As stated in Section 2, the derivation of the expression of QoS violation is difficult due to the functional dependency on the joint traffic distribution and the violation probability B. Instead of directly evaluating the QoS violation penalty, we define a new function $F(T_u)$ to evaluate the effective throughput,

which in fact quantifies the level of QoS guarantee of any subtree T_u rooted at the node u.

First, we denote the generic link QoS guarantee probability function as following:

$$q_u(D_u, c_u) = 1 - B_l, \qquad \text{where } l \text{ is uplink of } u. \tag{3.2}$$

This probability function is independent of any particular form of QoS violation function B. Thus, similar to the QoS violation penalty in Eq. (2.1), $F(T_u)$ can be defined as:

$$F(T_u) = \sum_{v \in L_u} \rho_v \prod_{i \in path(u,v)} q_i(D_i, c_i), \tag{3.3}$$

where L_u denotes the set of leaf nodes of the subtree T_u and $path(u, v)$ denotes all the nodes along the path from u to v. $F(T_u)$ can also be expressed in a recursive form:

$$F(T_u) = \begin{cases} \rho_u & \text{if node } u \text{ is a leaf node,} \\ \sum_{v \in S_u} F(T_v) \cdot q_v(D_v, c_v) & \text{otherwise.} \end{cases} \tag{3.4}$$

Using the above recursive function, we can compute $F(T_r)$, i.e. the total effective throughput of the SDT with the root node r.

Figure 3.3 illustrates an example of computing $F(T_r)$. The number inside each node represents the F value of the subtree, while the number besides the link represents the probability of QoS guarantee (q function). Using the above formulation, the effective throughput of the root node r is equal to the effective throughput of its two children nodes, weighted by the probability of QoS guarantees, therefore, $F(T_r) = 500(0.6) + 460(0.5) = 530$. Suppose now we have an additional continuous media server resource and we can place this extra server at node v in Figure 3.4. The two children nodes of node v are then served by this additional media server, while the remaining client nodes along the SDT are still served by the original media server r. The direct

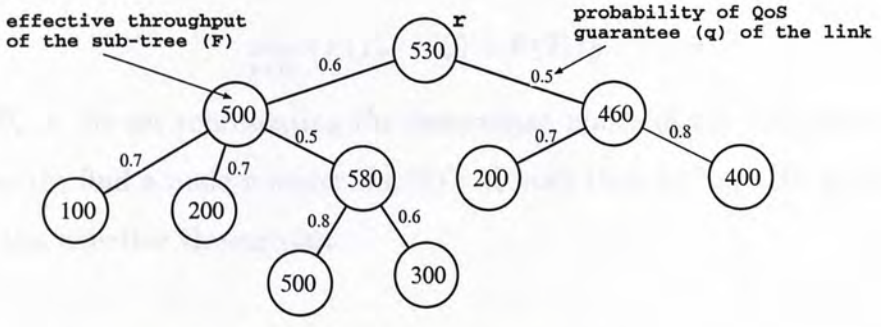


Figure 3.3: Illustrate of SDT. The value of each node is the effective throughput (F) of the subtree rooted at that node.

gain in effective throughput of placing an extra server at node v is equal to $580(1 - 0.5) = 290$. Since there will be no traffic going through the uplink of

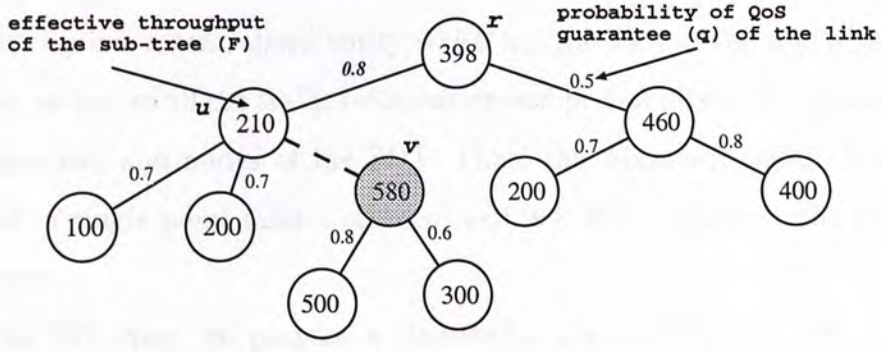


Figure 3.4: Illustration of SDT with one additional server placed at node v.

v, the total traffic on the uplink of node u will decrease, thus, improving the probability of QoS guarantee of that uplink, say from 0.6 to 0.8. As a result, the remaining tree $(T_r - T_v)$ also benefits from this additional placement of server in node v. As shown in the figure, the total effective throughput is equal to $F(T_r - T_v) + F(T_v) = 398 + 580 = 978$. In other words, there is around 85% gain in effective throughput by placing an additional server at node v.

Finally, given a SDT T_r, the *service replication problem* is formally defined

as [1] :

$$\max_{v \in \mathcal{D}_r} \{F(T_r - T_v) + F(T_v)\}. \tag{3.5}$$

where $\mathcal{D}_r$ is the set representing the descendant nodes of the root node r. In other words, find a node v under the SDT $\mathcal{T}_r$ such that we have the maximum gain in the effective throughput.

3.2.2 Distributed Evaluation of SDT

One way to find the optimal solution to the above problem in Eq. (3.5) is to perform an exhaustive evaluation at every nodes in the tree T_r and choose the node which maximizes the objective function in Eq. (3.5). However,since runtime of this approach is $O(n^2)$, it is computational prohibitive when the size of SDT is large. Another disadvantage of this exhaustive evaluation approach is that it requires a centralized entity which has the view of the whole network topology, as well as all the traffic information and probability of QoS guarantees of all the links and nodes of the SDT. Thus, this approach suffers from the potential of single point failure problem and it is not scalable as the network size grows.

In the following, we propose a *distributed* approach to solve the service replication problem. In our approach, each node only maintains *three* variables that summarize the characteristics of the subtree which rooted at that node. This way, the information can be *recursively evaluated* from the leaf nodes up to the root node. Since only a small amount of information is maintained at each node, the decision making can be carried out very efficiently in a top-down evaluation method.

We require that each node u of the SDT $\mathcal{T}$ maintains three variables,

[1] Here, for simplicity, we do not consider the cost of adding the replicated server. In Section 5, we illustrate the effect of replication cost in the final performance gain.

namely, D_u, q_u and F_u. The first variable D_u represents the total traffic demand of the subtree T_u, and it can be recursively evaluated using the following expression:

$$D_u = \begin{cases} \rho_u & \text{if } u \text{ is leaf node} \\ \sum_{v \in S_u} D_v & \text{otherwise} \end{cases} \tag{3.6}$$

The variable q_u is the probability of QoS guarantee on the uplink of node u to its parent node. This probability is computed at node u and it only depends on the traffic D_u and the allocated capacity c_u. It is important to point out that our approach can be applied to any general form of QoS guarantee probability function, as long as it is an increasing function of D_u. Lastly, the variable F_u is the total effective throughput of the subtree T_u. One can use the recursive expression in Eq. (3.4) to update these three variables and thereby obtain the effective throughput of the SDT.

Our evaluation scheme begins with all leaf nodes. Each leaf node, say u, will send the values of $\{D_u, q_u, F_u\}$ to its parent node v. The node v, upon receiving all the information from all its children nodes, will then update its own variables $\{D_v, q_v, F_v\}$ accordingly, and then send these values to its parent.

Consider an example which is illustrated in Figure 3.5 where each node maintains three local variables. After receiving the updated values from all its children, node 0 will then update its own values as follows:

$$\begin{aligned} D_0 &= D_1 + D_2 + D_3, \\ q_0 &= q_u(D_0, c_0), \\ F_0 &= q_1 F_1 + q_2 F_2 + q_3 F_3. \end{aligned}$$

All the other nodes are updated accordingly in a bottom-up manner. This process will continue until the root node r computes its effective throughput F_r. The above distributed approach is used to evaluate the effective throughput

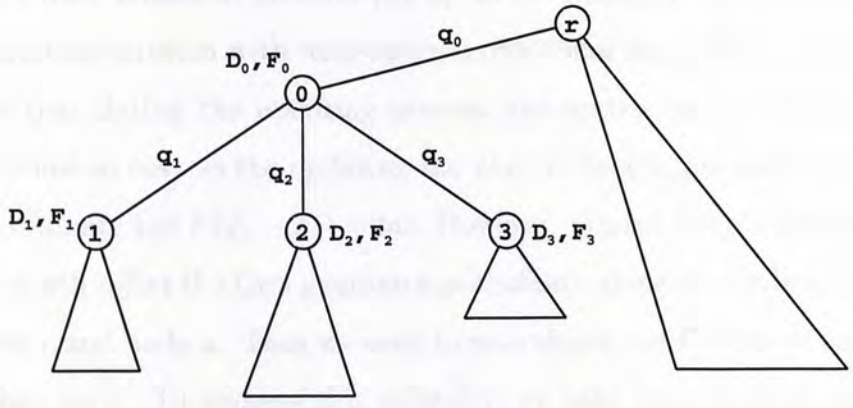


Figure 3.5: Illustration on the evaluation of SDT

of a SDT only. To find the proper node for service replication, we require each node, say node u, to maintain an extra variable G_u in order to find the optimal placement efficiently. The G_u is defined as the gain of total effective throughput if the additional server is placed at node u. The variable G_u can be expressed as:

$$G_u = F(T_r - T_u) + F(T_u) - F(T_r) \tag{3.7}$$

In other words, after placing the additional server at node u, the additional server will serve all the users of the subtree T_u only, while the original server r will serve all users from the remaining tree $T_r - T_u$. Therefore, the sum of the first two terms in Eq. (3.7) is the total effective throughput after service replication at node u and G_u represents the *gain* in the effective throughput if the replicated server is placed at node u.

3.2.3 Approximation

In general the problem of finding the optimal set of replicated server to maximize the total gain is NP-hard. It is not difficult to show that when we restrict the q-function to a constant function, this problem is equivalent to

the well-known p-median problem [13, 9]. In the following, we will propose an approximation solution with near-optimal result and much better efficiency.

Note that during the updating process, the update of the G function at node u is not so easy as the updating the D or F values. For each node u, we need to evaluate the $F(T_r - T_u)$ value. However, placing the additional server at node u will *affect* the QoS guarantee probability along all the links between root node r and node u. Thus we need to re-evaluate the F value of each node along that path. To address this difficulty, we take the following approach. We calculate the G function at node u simply by using the uplink probability only, i.e. $G_u = F(T_u)(1 - q_u)$ instead of calculating $F(T_r - T_u)$, because the $F(T_u)$ and q_u are directly available information at node u. In other words, the G_u of node u is the *minimum* guaranteed gain of the total effective throughput of SDT $\mathcal{T}$. Though we lose accuracy in the computation, we improve the efficiency of the algorithm, which is crucial for the real implementation.

Chapter 4

Service Replication Algorithms

In this section, we present the algorithm for selecting a node for service replication. To enhance the readers' understanding, we first present a centralized service replication algorithm, then we extend the concept to a distributed approach of service replication.

4.1 Centralized Service Replication Algorithm

The centralized algorithm has two phases, the *preprocessing phase* and the *searching phasing.*

4.1.1 Preprocessing Phase

The preprocessing phase can be carried in a recursive manner. Figure 4.1 illustrates the pseudocode of the recursive update of node u. Starting at the root node of SDT $\mathcal{T}$, this procedure is invoked recursively at each node. All the nodes of $\mathcal{T}$ are visited in a postorder sequence, and their D, F and G values are updated accordingly.

Lemma 4.1 Assuming the average degree of SDT is constant, the runtime complexity of preprocessing phase is $O(n)$.

```
UPDATE-NODE (u)
1   if S_u = ∅
        /* if u is leaf node */
2       D_u ← ρ_u
3       F_u ← ρ_u
4       G_u ← ρ_u · (1 − q_u(D_u, c_u))
5   else
        /* if u is internal node */
        /* update each child v of node u */
6       for v ∈ S_u do UPDATE-NODE (v)
7       D_u ← Σ_{v∈S_u} D_v
8       F_u ← Σ_{v∈S_u} F_v q_v(D_v, c_v)
9       G_u ← max{G_v}, (∀v ∈ S_u)
10      if parent(u) ≠ ∅
11          G_u ← max{G_u, F_u · (1 − q_u(D_u, c_u))}
```

Figure 4.1: Recursive update of node u of SDT $\mathcal{T}$

Proof: During the *preprocessing phase* (i.e. the UPDATE-NODE procedure), each node needs to gather all the information from all of its children nodes, therefore, the runtime for each node is proportional to the degree of that node, thus the total complexity is $O(n \cdot d)$, if d is the average degree. However, for the deployment on a service overlay network, because of the constraint of service gateway's restricted resources, the average degree of SDT will not be very large, and we can safely assume that it is a small constant. Therefore, the total complexity for the preprocessing phase is $O(n)$, where n is the number of nodes in the SDT. ∎

4.1.2 Searching Phase

When the preprocessing phase is completed, each node will obtain the updated information of $\{D_u, F_u, G_u\}$. Then one can search for the optimal server placement in a top-down manner starting at the root node of SDT $\mathcal{T}$. Figure 4.2 illustrates the procedures. The searching algorithm will output the node which maximizes the gain of effective throughput.

```
FIND-REP-NODE (T)
1  u ← root(T)
2  v ← max_v{G_v}, ∀v ∈ S_u
3  while G_v ≥ G_u and S_u ≠ ∅ do
4        u ← v
5        v ← max_v{G_v}, ∀v ∈ S_u
6  return u
```

Figure 4.2: Finding the node to place the replicated server

Lemma 4.2 The average runtime complexity of searching phase is $O(\log(n))$.

Proof: During the *searching phase* (i.e. the FIND-REP-NODE procedure), the searching starts from the root node and at the worst case, it will stop at the leaf node. At each node, it just chooses one child of the maximum G value. Thus the total complexity is proportional to the height of the tree. Although the worst case complexity is $O(n)$, for a practical deployment, the construction of a SDT is usually balanced, therefore the average complexity is $O(\log(n))$.

■

4.2 Distributed Service Replication Algorithm

Although the above centralized algorithm is simple to implement, it requires a centralized entity in the SON for execution. This requires extra resources and also has the potential of a single-point-failure problem. These problems will become significant when the size of the SON is large. We propose the following distributed algorithm, which can be concurrently executed on each node inside the SDT. Thus, no centralized management is required and the server replication can be carried in a more efficient manner.

The distributed algorithm achieves the same result as the centralized algorithm by sending messages among the nodes of SDT. Figure 4.3 illustrates the distributed service replication algorithm. It is divided into five parts. The first two parts (lines 1-15), correspond to the preprocessing phase, while the rest three parts (lines 16-24) correspond to the searching phase.

The DISTRIBUTED-NODE-UPDATE() procedure can be implemented as an event-driven program running at each node. The information exchange between nodes can be implemented as a simple protocol with the following set of of messages: $\langle request\_update \rangle$, $\langle reply\_update \rangle$, $\langle exec\_search \rangle$, $\langle request\_G \rangle$, $\langle reply\_G \rangle$.

The root node will initiate the distributed algorithm by sending the $\langle request\_update \rangle$ to all its child nodes. Upon receiving this message, these nodes will send the same message to their children (line 6), and this message will be propagated till the leaf nodes. The leaf nodes will then send the $\langle reply\_update \rangle$ to their parents with the updated values of D, F, G (line 5). Each node, upon receiving $\langle reply\_update \rangle$ message will then update its own D, F, G values (line 8-10). When it receives the updates from all its children, the processing phase on that node is finished, and it will send the $\langle reply\_update \rangle$ message to its parent (line 14). When the root node finally

receives all the updates from its children and updates its own D, F, G, the whole preprocessing phase is terminated.

The root node will then start the searching phase (line 15). It will ask the G values of all its children by sending the $\langle request\_G \rangle$ message. Upon receiving the reply, it will pick the child node with the G value not less than the G value of itself, and then send the $\langle exec\_search \rangle$ message (line 22). This process will stop when there is one node in which the G value of all its children are less than itself (line 23). At this moment, the searching phase is terminated and that node will be picked to place the replication.

For the distributed service replication algorithm, the preprocessing phase can be executed in a *parallel* fashion, in which case the total running time of the *preprocessing phase* can be improved to $O(\log(n))$ (proportional to the height of the tree). Therefore, the total running time of our algorithm is also improved to $O(\log(n))$. It is much faster than the exhaustive searching method ($O(n^2)$), when the total number of node (n) is large.

4.3 Improved Distributed Algorithm

The centralized and distributed algorithms discussed above are easy to implement on top of SON. However, one may provide a better solution (e.g., in terms of finding a closer-to-optimal gain in the effective throughput) if each node is allowed to stored more information. In the following, we provide an improved version of the distributed algorithm which can find a better solution at the cost of extra computational resources.

In the previous algorithms, to determine the minimum possible gain in the total effective throughput (G_u) of placing a replicated server at node u, we consider the uplink QoS guarantee probability (q_u) only. However, in this improved distributed algorithm, we use the total QoS guarantee probability

along the path from the root node r to the node u, and we denote this total probability to be Q_u for each node u. We can define Q_u recursively as:

$$Q_u = \begin{cases} 1 & u \text{ is root node} \\ q_u \cdot Q_{parent(u)} & \text{otherwise} \end{cases} \tag{4.1}$$

Therefore, we redefine the G_u to be:

$$G_u = F(T_u)(1 - Q_u) \tag{4.2}$$

To deploy this new algorithm, each node needs to maintain an extra variable Q_u, and the following procedure NODE-IMPROVE which serves as an add-on module to the basic distributed algorithm, can be invoked, if necessary, after the *preprocessing phase* and before the *searching phase*. To use this add-on module, we only need to modify the (line 15) of DISTRIBUTED-NODE-UPDATE to:

15 **else send** $\langle improve\_Q : 1 \rangle$ **to self**

Then the root node, before starting the searching phase, will first initiate the updating of the Q_u as well as the G_u value of each node.

Figure 4.4 illustrate the add-on module where line (1-7) updates the Q_u of each node, and line (8-15) updates the G_u of each node.

```
DISTRIBUTED-NODE-UPDATE (u)
1   upon receiving ⟨request_update⟩
2   if S_u = ∅
3       D_u ← ρ_u, F_u ← ρ_u
4       G_u ← ρ_u(1 − q_u(D_u, c_u))
5       send ⟨reply_update : D_u, F_u, G_u⟩ to parent(u)
6   else send ⟨request_update⟩ to all v ∈ S_u

7   upon receiving ⟨reply_update : D_v, F_v, G_v⟩ from child v
8   D_u ← D_u + D_v
9   F_u ← F_u + F_v q_v(D_v, c_v)
10  G_u ← max{G_u, G_v}
11  if received ⟨reply_update⟩ message from all children
12      if parent(u) ≠ ∅
13          G_u ← max{G_u, F_u(1 − q_u(D_u, c_u))}
14          send ⟨reply_update : D_u, F_u, G_u⟩ to parent(u)
15      else send⟨exec_search⟩ to self

16  upon receiving ⟨exec_search⟩
17  if S_u = ∅ output u
18  else send ⟨request_G⟩ to all v ∈ S_u

19  upon receiving ⟨request_G⟩
20  send ⟨reply_G : G_u⟩ to parent(u)

21  upon receiving ⟨reply_G : G_v⟩ from child v
22  if G_v ≥ G_u send ⟨exec_search⟩ to node v
23  else if received ⟨reply_G⟩ messages from all children
24      output u
```

Figure 4.3: Distributed algorithm running at each node u

```
NODE-IMPROVE (u)
1   upon receiving ⟨improve_Q : Q_p⟩
2   Q_u ← q_u(D_u, c_u) · Q_p
3   if S_u = ∅
4       G_u ← F_u(1 − Q_u)
5       send ⟨improve_G : G_u⟩ to parent(u)
6   else
7       send ⟨improve_Q : Q_u⟩ to all v ∈ S_u
8   upon receiving ⟨improve_G : G_v⟩ from child v
9   G_u ← max{G_u, G_v}
10  if received ⟨improve_G⟩ message from all children
11      if parent(u) ≠ ∅
12          G_u ← max{G_u, F_u · (1 − Q_u)}
13          send ⟨improve_G : G_u⟩ to parent(u)
14      else
15          send ⟨exec_search⟩ to self
```

Figure 4.4: Add-on module for improved distributed algorithm

Chapter 5

Performance Evaluations

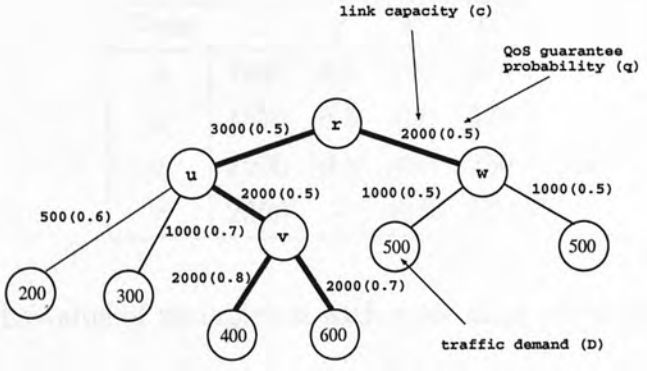
In this section, we perform three experiments so as to evaluate the performance and effectiveness of the service replication algorithm. The first experiment provides some understanding of our replication algorithm and shows the benefit of performing service replication. The second experiment evaluates the quality of the results obtained by our algorithm as compares to random selection and exhaustive selection of the replication. The third experiment illustrates the scalability of the service replication algorithm when we increase the size of the SDT.

5.1 Experiment 1: Algorithm Illustration

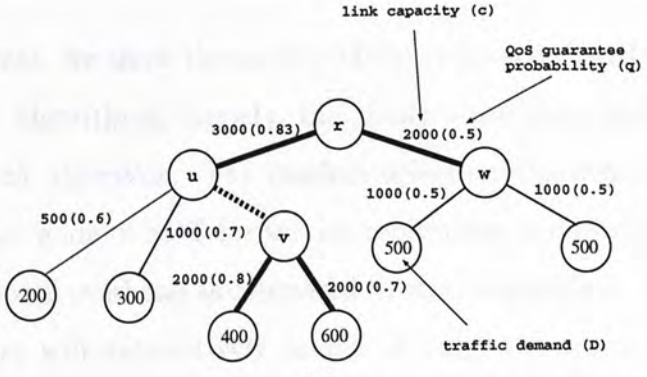
In this experiment, we show the benefit of the replication algorithm. Figure 5.1(a) illustrates a simple yet illustrative topology of the SDT, as well as the traffic demand of each client node and the capacity of each link. In this experiment, the QoS guarantee probability function q is expressed as the following linear equation[1] :

$$q_u(D_u, c_u) = 1 - \frac{D_u}{c_u} \tag{5.1}$$

[1]The probability function is only used as an illustration, the algorithm can accommodate general forms of probability function.



(a)



(b)

Figure 5.1: (a) The SDT before replication (b) The SDT after replication

In Table 5.1, we illustrate the variables of each internal node after running the distributed service replication algorithm. It finds that the node v has the maximum G value and the additional server will be placed at node v.

After placing the additional service node at node v, the original SDT is split into two subtrees, as shown in Figure 5.1(b). Since there is no longer any traffic going through link (u, v), the F value of node u and r will be changed to 330 and 525 respectively. Therefore, the total effective throughput after replication is $F_r + F_v = 525 + 740 = 1265$ and the gain of placing a replication at node v is 1265-600=665, which is a 110% gain as compare to the original SDT with no replication.

Node	D	q	F	G
v	1000	0.5	740	370
u	1500	0.5	700	370
w	1000	0.5	500	250
r	2500	–	600	370

Table 5.1: The value of variables at each node after preprocessing phase

5.2 Experiment 2: Performance Comparison

In this experiment, we show the quality of the proposed distributed algorithm with other two algorithms, namely, the *random selection algorithm* and the *exhaustive search algorithm.* The random selection algorithm will randomly select an internal node of SDT for service replication and obviously, it has the least computational overhead as compared to other algorithms. The exhaustive search algorithm will exhaustively search through the whole SDT tree and will find the optimal node for replication. This algorithm has the largest computational complexity and it cannot be scaled as we increase the size of the SON.

In this experiment, we use a random tree generator to generate 100 random SDTs with 500 nodes each, and the average number of children of each internal node is three (e.g., average degree is 3). Each client node also has a random traffic demand which is uniformly distributed with a range from 1 to 1000 unit. The link capacity is provisioned[7] in the way such that the loading on each link (D/c) is a constant. We then vary this constant, and compare the gain of placing a replicated server on the SDT.

Table 5.2, 5.3 and 5.4 illustrate the result of our experiments. We compare the results using three different q functions. The second column is the gain of placing service replication at a randomly chosen node. The next two columns

are the gains of placing a server at the node obtained by our basic distributed algorithm and the improved distributed algorithm. The rightmost column is the gain of placing the replication at the optimal node using the exhaustive search (which is obtained by exhaustively evaluating all the nodes). From

D/c	**Random selection**	**Basic Distributed Alg**	**Improved Distributed Alg**	**Exhaustive selection**
0.4	2.21%	19.45%	21.29%	22.33%
0.6	7.90%	35.71%	40.56%	42.61%
0.8	46.34%	84.51%	87.80%	91.76%

Table 5.2: Comparison of our algorithm with random placement and optimal placement, when $q_u = 1 - D_u/c_u$.

D/c	**Random selection**	**Basic Distributed Alg**	**Improved Distributed Alg**	**Exhaustive selection**
0.4	0.61%	6.43%	7.10%	7.79%
0.6	1.99%	16.93%	18.93%	20.97%
0.8	11.27%	40.33%	47.17%	53.62%

Table 5.3: Comparison of our algorithm with random placement and optimal placement, when $q_u = 1 - (D_u/c_u)^2$.

these three tables, we can conclude that our improved distributed algorithm, which has a much lower computational complexity than the exhaustive search, has a performance result which is very closed to the optimal. Another observation can be made from these tables is that when the average loading on each link (D/c) is high, it is more beneficial to perform the service replication.

D/c	**Random selection**	**Basic Distributed Alg**	**Improved Distributed Alg**	**Exhaustive selection**
0.4	0.19%	0.94%	1.05%	1.37%
0.6	0.60%	5.11%	5.69%	6.73%
0.8	3.10%	20.09%	22.90%	27.95%

Table 5.4: Comparison of our algorithm with random placement and optimal placement, when $q_u = 1 - (D_u/c_u)^4$.

5.3 Experiment 3: Scalability Analysis

In the following 3 experiments, we illustrate the performance of our algorithm when the size of the SDT increases. For each size of the SDT, we generate 100 instances of random SDTs with link capacity set to the value such that the loading D/c is 0.8.

5.3.1 Experiment 3A

In this experiment, we illustrate the percentage of average performance gain obtained by our algorithm for SDTs from 100 to 2000 nodes.

As shown in Figure 5.2, although there is a little fluctuation when the SDT size is small, the average gain of our replication algorithm still remains at a certain percentage when the network size grows.

We also test on SDTs of different average degrees. When the average degree increases, the average gain decrease. The reason is that when the average degree of tree is small, the height of tree is larger, i.e. the average path length from client node to the root node is longer. This means the QoS guarantee are much harder to preserve for the client nodes, therefore doing replication at SDT of small average degree will have more benefit. Meanwhile, because of the constraint of resources at each service gateway of SON, the average degree

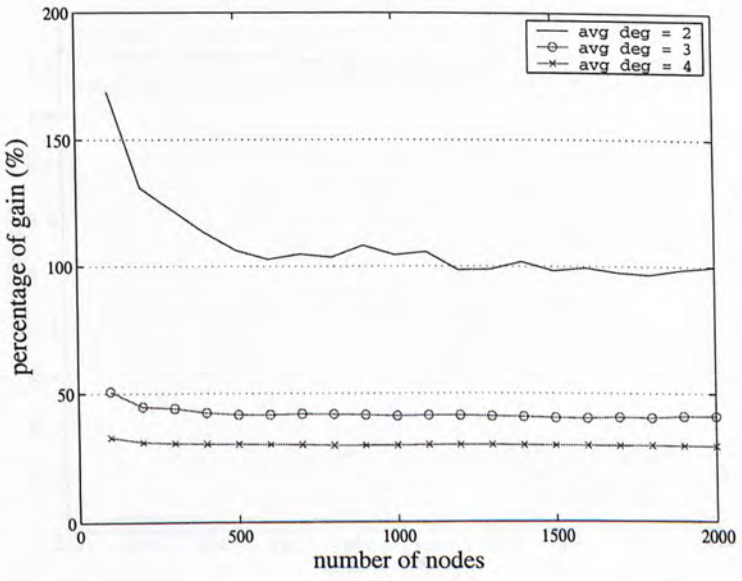


Figure 5.2: Illustration on the performance gain of our service replication algorithm when the size of the tree grows from 100 to 2000 nodes.

of SDT in real situation will not be a big number, so our service replication is suitable for SDT.

5.3.2 Experiment 3B

In this experiment, we illustrate the percentage of average normalized gain (with respect to the optimal gain) obtained by our algorithm for SDTs from 100 to 1000 nodes.

We repeat this experiment for three different kind of QoS guarantee function (q-function). As shown in Figure 5.3, for the more loading-sensitive q-function, our algorithms obtain lower performance gain, which is predictive because the entire SON is more easily to suffer QoS penalty when congestion happens.

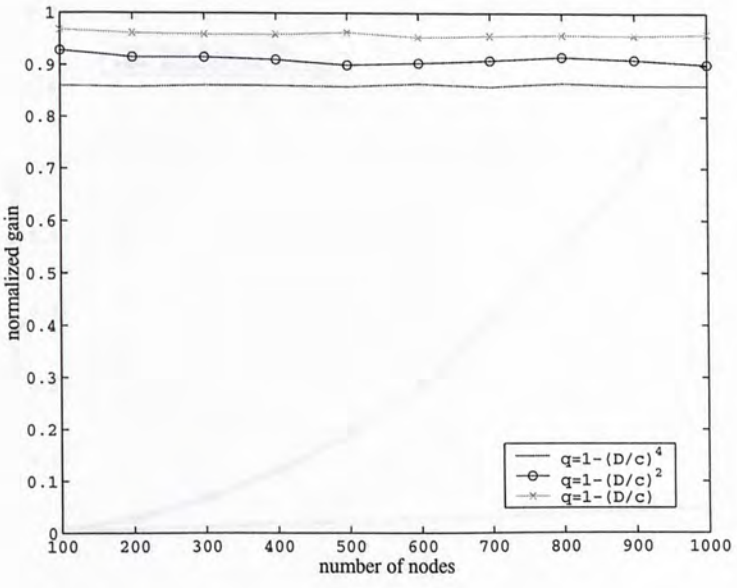


Figure 5.3: Normalized gain obtained by our algorithm retains a certain percentage when the network size increases.

However, the gain obtained by our algorithm for each QoS guarantee function is independent from the network size. Even the network size increases 10 times, it still remains at almost the same level.

5.3.3 Experiment 3C

In this experiment, we shows the average CPU time used by our algorithm as compared to the exhaustive search method for the network size increases from 100 to 1000 nodes. The time measurement is averaged over 100 instances for each network size.

In Figure 5.4, we illustrate the real CPU time for our algorithm as compared to the time used by exhaustive search method. It is a further proof the complexity of our algorithm is linear time and is scalable for bigger network topology.

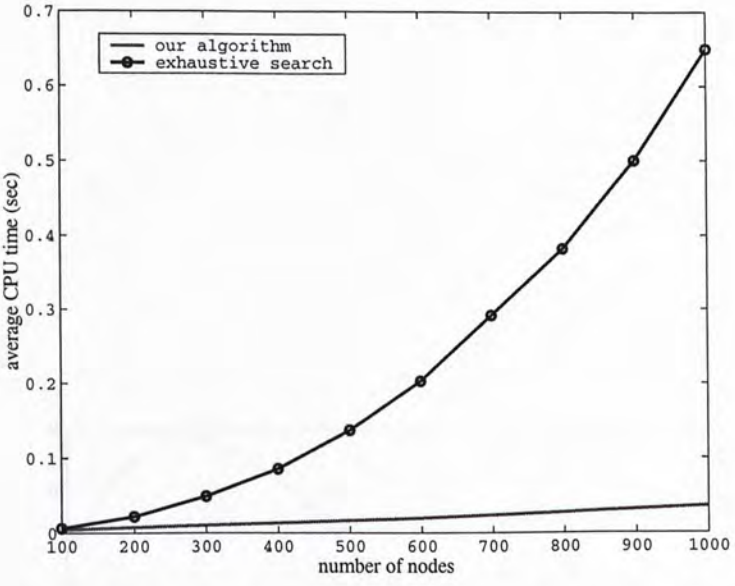


Figure 5.4: Average CPU time for finding one optimal replicated server.

5.4 Experiment 4: Multiple replications

In this experiment, we try to do multiple replications on the SDT by slightly modifying our algorithm to maintain a priority queue of potential nodes for the placement of replicated server. We then compute the gain of total effective throughput of putting various numbers of replicated servers. Figure 5.5, shows the average gain of 100 random generated SDTs of 1000 nodes each.

In the previous experiments, we didn't consider the cost of doing replication, however in reality, there will be some cost of placing one cost, and it will affect the result when we want to do multiple replications. Thus we set the cost of adding one more replicated server to be 0.1% of the total traffic demand D of the SDT. As shown in the figure, because of this replication cost, when the number of replications increases, the percentage gain will diminish at some point and eventually we can not gain more by adding extra replications.

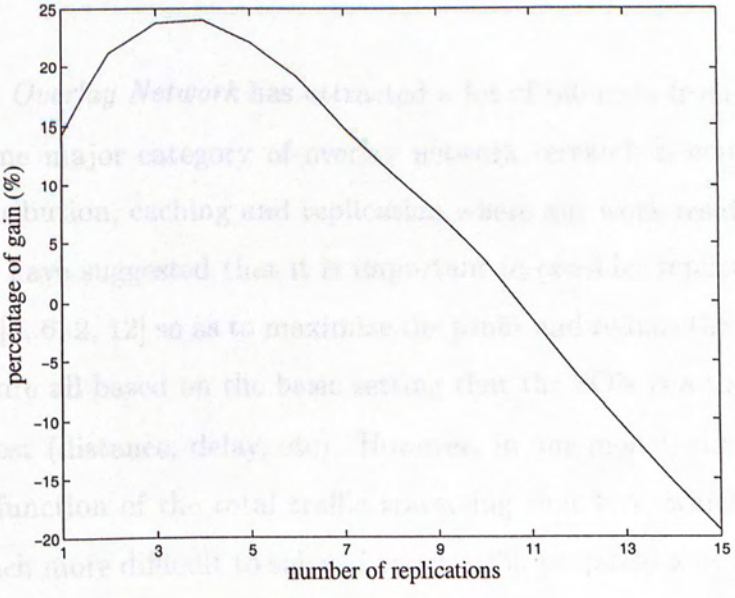


Figure 5.5: Illustration of the average gain of multiple replications on SDT of 500 nodes

Chapter 6

Related Work

The *Service Overlay Network* has attracted a lot of interests from researchers recently. One major category of overlay network research is concerning the content distribution, caching and replication where our work resides in. Various authors have suggested that it is important to consider replicated service in the SON [5, 6, 2, 12] so as to maximize the profit and reduce the operational cost. They are all based on the basic setting that the SON is a topology with fixed link cost (distance, delay, etc). However, in our model, the link cost is a dynamic function of the total traffic traversing that link, which makes the problem much more difficult to solve. Low *et al* [3], proposed a novel approach for the relevant server placement problem. They model the candidate nodes as source and replicated server nodes as code and thus formulate a source coding problem that minimize the coding distortion. Though their solution fits any general graph, their high-density nodes assumption may not always be valid in SON. Krishnan *et al* [15], proposed a dynamic programming algorithm to solve the cache placement problem. They also achieve near-optimal result, however, their problem is based on the network delay cost model, which is also different to our effective throughput metrics.

Chapter 7

Conclusion

Previous works have studied the bandwidth provisioning problems and optimal distribution tree formulation on SON. However, since the bandwidth is fixed after provisioning and the topology is static, the SON is inflexible to traffic demand variation, and when there is sudden increase of traffic demand, the QoS guarantee probability will decrease on links, therefore the total effective throughput will decrease and that means the reduction in total profit of SON.

We proposed to address this problem by service replication in the service delivery tree model. We have presented both centralized and distributed algorithms to find the placement of a replicated server, which maximize the total effective throughput of SDT. The distributed algorithm requires very little resource at each node, and can be implemented as a simple protocol among all the service gateways of SON. The complexity of the algorithm is much lower than the brute-force exhaustive search method, but still achieve a near-optimal result. Furthermore, it has a good scalability and can be deployed in large scale SON networks.

Bibliography

[1] D. Bertsekas and R. Gallager. *Data Networks*, chapter 5.5, pages 451–455. Prentice Hall, 2nd edition, 1992.

[2] J. Byers, J. Considine, and M. Mitzenmacher. Informed Content Delivery Across Adaptive Overlay Networks. In *ACM SIGCOMM*, Aug. 2002.

[3] C. W. Cameron, S. H. Low, and D. X. Wei. High-density model for server allocation and placement. In *IEEE Infocom*, 2002.

[4] Y. Chu, S. G. Gao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM 2001*, Apr. 2001.

[5] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *ACM SIGCOMM*, 2002.

[6] Y. Cui, Y. Xue, and K. Nahrstedt. Optimal resource allocation in overlay multicast. In *"IEEE 11th International Conference on Network Protocols (ICNP'03)"*, Nov. 2003.

[7] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Service Overlay Networks: SLAs, QoS and Bandwidth Provisioning. In *IEEE 10th International Conference on Network Protocols (ICNP'02)*, Paris, France, Nov. 2002.

[8] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Service Overlay Networks: SLAs, QoS and bandwidth provisioning. Technical report, Computer Science Department, University of Minnesota, Feb. 2002.

[9] M. R. Garey and D. S. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[10] J. Jannotti. *Network Layer Support for Overlay Networks.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Aug. 2002.

[11] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, Oct. 2000.

[12] J. Kangasharju, J. Roberts, and K. W. Ross. Object Replication Strategies in Content Distribution Network. *Computer Communications*, 25, 2002.

[13] O. Kariv and S. L. Hakimi. An Algorithmic Approach to Network Location Problems - Part II:p-medians. *SIAM J. Appl. Math.*, 37:539–560, 1979.

[14] M. S. Kim, S. S. Lam, and D.-Y. Lee. Optimal Distribution Tree for Internet Streaming Media. In *23rd IEEE ICDCS*, May 2003.

[15] P. Krinshnan, D. Raz, and Y. Shavitt. The Cache Location Problem. *IEEE/ACM Transactions on Networking (TON)*, 8:568–582, Oct. 2000.

[16] D. Mitra and Q. Wang. Stochastic traffic engineering, with applications to network revenue management. In *IEEE Infocom 2003*, San Francisco, USA, 2003.

[17] Z.-L. Zhang, Z. Duan, Y. T. Hou, and L. Gao. Decoupling QoS Control from Core Routers: A Novel Bandwidtg Broker Architecture for Scalable Support of Guaranteed Services. In *ACM SIGCOMM*, Aug. 2000.