Distributed Algorithm for Service Replication in Service Overlay Network

Kevin Y.K. Liu[1], John C.S. Lui[1], and Zhi-Li Zhang[2] *

[1] Computer Science & Engineering, The Chinese University of Hong Kong
{ykliu,cslui}@cse.cuhk.edu.hk
[2] Department of Computer Science, University of Minnesota zhzhang@cs.umn.edu

Abstract. The service overlay network (SON) is an effective mean to deploy end-to-end QoS guaranteed content delivery services on the current Internet. We model the content delivery service on a SON as a service delivery tree (SDT). Previous studies have addressed the optimal distribution tree formulation issues. In this paper, we focus on the problem of maximizing the total operation profit of the SON. In [1], authors introduced a cost model for optimal bandwidth provisioning in SON. In this paper, we extended this concept and propose an alternative approach to maximize the total effective throughput of SDT as well as to minimize the QoS violation penalty of the SON by service replication. We present both centralized and distributed algorithms for the placement of replicated servers on the SDT. Experiments are carried to quantify the merit, effectiveness and the scalability of the proposed service replication algorithm. In particular, the performance gain is very close to the exhaustive search. The algorithm performs well when we scale up the service overlay networks. Finally, we show that one only needs to perform a small number of replications to attain the optimal gain.

Key words: overlay networks, QoS, replication algorithm

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

* John C.S Lui was supported in part by the RGC Research Grant 4420/01E. Zhi-Li Zhang was supported in part by the National Science Foundation (NSF)under the Grant ITR-0085824. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views the NSF.

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 Internet global connectivity. 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 [1], authors advocate the notion of *service overlay network* (SON) as an effective mean to address 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 network, one can provide different types of time sensitive services, such as video-on-demand, Internet radio, VoIP, etc. SON offers these services to 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 [1, 2], authors formulate this problem as bandwidth provisioning model, wherein the revenue of SON comes from the fee paid by users and the costs consist of bandwidth provisioning cost and the QoS violation penalties. 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.

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 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.

Note 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 [3], to deliver this type of service, a data delivery process is usually in 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.

2 Background on Service Overlay Network

In this section, we provide the necessary background on service overlay networks and its bandwidth provisioning problem.

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. A link in $\mathcal{L}$ is a *logical* connection between two service gateways and the link is an IP level 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. 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.

The advantage of the SON architecture is that it decouples the application services from the network services 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.

The *"bandwidth provisioning problem"*[1, 2, 4] 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 formal mathematical framework for performing the bandwidth provisioning can be described as follows. Given a network topology $\mathcal{G}$, the source-destination (SD) path requirements in $\mathcal{R}$, the stochastic traffic demand $\{\rho_r\}$ for each $r \in \mathcal{R}$, and the routing method, one can provide a *lower bound* of the expected net profit (or income) for the service overlay network. Let r denote a path in the source-destination path set $\mathcal{R}$. Assume that the traffic demand distribution on path r is known[1] and traffic of all paths in $\mathcal{R}$ are described by the stochastic traffic demand matrix $\{\rho_r\}$, the total net income for the SON, denoted by the random variable W, can be expressed as:

$$W(\{\rho_r\}) = \sum_{r\in\mathcal{R}} e_r\rho_r - \sum_{l\in\mathcal{L}} \Phi_l(c_l) - \sum_{r\in\mathcal{R}} \pi_r\rho_r B_r(\{\rho_r\}). \tag{1}$$

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. The problem of bandwidth provisioning can thus be formulated as the optimization of the average total net profit $E(W)$, or:

$$\max_{c_l} E(W). \tag{2}$$

In other words, determining the appropriate amount of capacity $\{c_l\}$ for each link $l \in \mathcal{L}$.

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

Note that the above 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. 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.

3 Mathematical Formulation for Service Replication

Real-time content delivery is one of the major applications of SON, many QoS sensitive services can be deployed on the SON's infrastructure. As illustrated in [3], the optimal data delivery topology for these applications is a tree topology. In this paper, we call the topology as a *service delivery tree* (SDT). The root node of SDT is an application level service gateway. All the leaf nodes are called client nodes. They are access points for users within the same network domain. The formation of delivery tree can be different for different applications [3, 5–7], however, our model is generic for any tree formation.

To formally define the service replication problem, we use the following notations:

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

In the original SON bandwidth provisioning model [1, 4], 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. (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. (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:

$$q_u(D_u, c_u) = 1 - B_l, \qquad \text{where } l \text{ is uplink of } u. \tag{3}$$

This probability function is independent of any particular form of QoS violation function B. Similar to the QoS violation penalty in Eq.(1), $F(T_u)$ can then be defined as:

$$F(T_u) = \sum_{v \in L_u} \rho_v \prod_{i \in path(u,v)} q_i(D_i, c_i), \tag{4}$$

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:

$$F(T_u) = \begin{cases} \rho_u & \text{if node } u \text{ is a leaf node,} \\ \sum_{v \in S_u} F(T_v) \cdot q_v(D_v, c_v) & \text{otherwise.} \end{cases} \tag{5}$$

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. [2]

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

$$\max_{v \in \mathcal{D}_r} \{F(T_r - T_v) + F(T_v)\}. \tag{6}$$

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 $\mathcal{T}_r$ to maximize the gain in the effective throughput.

3.1 Distributed Approach to Evaluate the Effective Throughput of SDT

One way to find the optimal solution to the above problem in Eq. (6) 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. (6). 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 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, 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:

$$D_u = \begin{cases} \rho_u & \text{if } u \text{ is leaf node} \\ \sum_{v \in S_u} D_v & \text{otherwise} \end{cases} \tag{7}$$

[2] We have omitted an example here due to page limit. It is available at [8].

The variable q_u is the probability of QoS guarantee on the uplink of node u to its parent node. It is computed at node u. It is defined as an generic QoS guarantee function of the traffic D_u and the capacity c_u. It is important to point out that our approach can be applied to any specific form of this 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. (5) 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 them to its parent. 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.

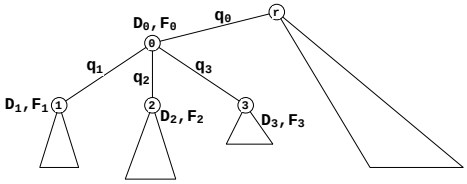

Fig. 1. Illustration on the evaluation of SDT

Consider an example illustrated in Fig.1. After receiving the updated values from all its children, node 0 will then update its own values as follows: $D_0 = D_1 + D_2 + D3$, $q_0 = q_u(D_0, c_0)$ and $F_0 = q_1F_1 + q_2F_2 + q_3F3$.

The above distributed approach is used to evaluate the effective throughput of a SDT only. To find the proper node for service replication efficiently, we require each node, say node u, to maintain an extra variable G_u This G_u represents the gain of total effective throughput by placing the additional server at node u. It is defined as:

$$G_u = F(T_r - T_u) + F(T_u) - F(T_r) \tag{8}$$

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. (8) 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.

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 calculate $F(T_r - T_u)$, because the $F(T_u)$ and q_u are directly available information at node u. In other words, G_u is the *minimum* guaranteed gain of the total effective throughput of SDT $\mathcal{T}$.

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, namely, *preprocessing* and *searching*. The preprocessing phase can be carried in a recursive manner. Fig.2 illustrates the pseudo-code of the recursive update of node u.

```
UPDATE-NODE (u)
1  if S_u = ∅                                   /* if node 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                                         /* node u is not leaf node */
6       for v ∈ S_u do UPDATE-NODE (v)           /* for each child v of node u */
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) ≠ ∅                         /* if node u is not root node */
11           G_u ← max{G_u, F_u · (1 − q_u(D_u, c_u))}
```

Fig. 2. Recursive update of node u of SDT $\mathcal{T}$

Lemma 1. *Assuming the average degree of SDT is constant, the runtime complexity of preprocessing phase is* $O(n)$. [3]

At the end of preprocessing phase, each node will obtain the updated values 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 $\mathcal{T}$. Fig.3 illustrates the procedures. The searching algorithm will output the node that maximizes the gain of effective throughput.

```
FIND-REP-NODE (T)
1 u ← root(T)
2 v ← max_v{G_v}, ∀v ∈ S_u          /* get the child v with maximum gain */
3 while G_v ≥ G_u and S_u ≠ ∅ do    /* while G_v ≥ G_u and u is not leaf node */
4       u ← v                      /* proceed to the next node */
5       v ← max_v{G_v}, ∀v ∈ S_u
6 return u
```

Fig. 3. Finding the node to place the replicated server

Lemma 2. *The average runtime complexity of searching phase is* $O(\log(n))$.

[3] The proof of the lemma is omitted here due to page limit. Please refer to [8].

4.2 Distributed Service Replication Algorithm

Though 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 more efficiently. The distributed algorithm achieves the same result as the centralized algorithm by sending messages among the nodes of SDT. Fig.4 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.

```
DISTRIBUTED-NODE-UPDATE (u)
 1  upon receiving ⟨request_update⟩
 2  if S_u = ∅                                          /* if u is not leaf node */
 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          /* ask all children to update */

 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   /* wait until all children are updated */
12        if parent(u) ≠ ∅                              /* if not root node */
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)   /* send updated information to parent */
15        else send⟨exec_search⟩ to self                /* start searching phase */

16  upon receiving ⟨exec_search⟩
17  if S_u = ∅ output u                                 /* if leaf node, output u */
18  else send ⟨request_G⟩ to all v ∈ S_u                /* else ask each child v to send G_v */

19  upon receiving ⟨request_G⟩
20  send ⟨reply_G : G_u⟩ to parent(u)                   /* reply G value to parent node */

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
```

Fig. 4. Distributed algorithm running at each node u

The DISTRIBUTED-NODE-UPDATE() procedure can be implemented as an event driven program running at each node. The information exchange between nodes can be implement 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 $\langle reply\_request \rangle$ to their parents with the updated values of D, F, G (line 5). Each node, upon receiving $\langle reply\_request \rangle$ 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 $\langle reply\_request \rangle$ 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 $(O(n^2))$.

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:

$$Q_u = \begin{cases} 1 & u \text{ is root node} \\ q_u \cdot q_{parent(u)} & \text{otherwise} \end{cases} \tag{9}$$

Therefore, we redefine the G_u to be:

$$G_u = F(T_u)(1 - Q_u) \tag{10}$$

To deploy this new algorithm, each node needs to maintain an extra variable Q_u, and the following procedure NODE-IMPROVE which served 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 start the searching phase, will first initiate the updating of Q_u as well G_u value of each node. Fig.5 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.

5 Experiments

In this section, we perform two experiments [4] so as to evaluate the performance and effectiveness of our service replication algorithm. The first experiment evaluates the

[4] There are two more experiment in [8] to demonstrate our replication algorithm, which are omitted here due to page limit.

```
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 = ∅                                   /* if u is leaf node */
4        G_u ← F_u(1 − Q_u)
5        send ⟨improve_G : G_u⟩ to parent(u)    /* ask the parent node to update the G value */
6  else                                         /* if u is internal node */
7        send ⟨improve_Q : Q_u⟩ to all v ∈ S_u  /* ask the children to update Q */

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   /* wait till all children updated G value */
11       if parent(u) ≠ ∅                       /* if u is not root node */
12             G_u ← max{G_u, F_u · (1 − Q_u)}
13             send ⟨improve_G : G_u⟩ to parent(u)   /* ask parent node to update G */
14       else                                   /* if the root receives all updates */
15             send ⟨exec_search⟩ to self       /* the root node start searching phase */
```

Fig. 5. Add-on module for improved distributed algorithm

quality of the results obtained by our algorithm as compares to random selection and exhaustive selection the replication. The second experiment illustrates the scalability of the service replication algorithm when we increase the size of the SDT.

Experiment 1: (Comparing the quality of the distributed service replication algorithm with random selection and exhaustive search): In this experiment, we show the quality of our distributed algorithm comparing with the other two algorithms. The random selection algorithm will arbitrarily pick an internal node of SDT for service replication. Obviously, it has the least computational overhead as compare to other algorithms. The exhaustive search algorithm will search throughout the whole SDT tree and find the optimal node for replication. This algorithm has the largest computational complexity and is not scalable.

In this experiment, we randomly generate 100 instances of SDTs of 500 nodes each. The average number of children of each internal node is set to 3. Each client node has a random traffic demand uniformly distributed within a range from 1 to 1000 unit. The link capacity is provisioned in the way such that the loading on each link (D/c) is a constant. We compare the gain of placing a replicated server at variable link loading.

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

Table 1- 3 illustrate the result of our experiments. From these tables, we can conclude that our improved distributed algorithm, which has a much lower computational complexity than the exhaustive search, has a performance 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	0.61%	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 2. when $q_u = 1 - (D_u/c_u)^2$.

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 3. when $q_u = 1 - (D_u/c_u)^4$.

Experiment 2: (Illustration of the scalability of our algorithm): In this experiment, we illustrate the performance of our algorithm when the size of the SDT grows from 100 nodes to 2000 nodes. For each size of the SDT, we generate 100 instances of SDT and compute the average performance gain. All the link capacity are set to the value such that the D/c is 0.8. As shown in Fig.6, 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 even the network size grows.

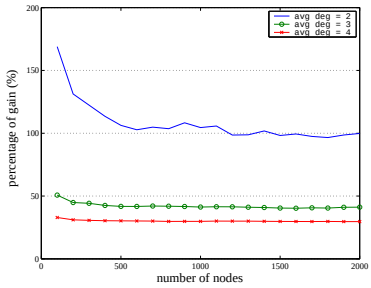

Fig. 6. Illustration on the performance gain of our service replication algorithm when the size of the tree grows from 100 to 2000 nodes.

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

6 Conclusions

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.

In this paper, we proposed to resolve this problem by service replication in the service delivery tree. 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.

References

1. 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.
2. D. Mitra and Q. Wang, "Stochastic traffic engineering, with applications to network revenue management," in *IEEE Infocom 2003*, (San Francisco, USA), 2003.
3. M. S. Kim, S. S. Lam, and D.-Y. Lee, "Optimal Distribution Tree for Internet Streaming Media," in *23rd IEEE ICDCS*, May 2003.
4. Z. Duan, Z.-L. Zhang, and Y. T. Hou, "Service Overlay Networks: SLAs, QoS and bandwidth provisioning," tech. rep., Computer Science Department, University of Minnesota, Feb. 2002.
5. 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)*, pp. 197–212, Oct. 2000.
6. 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.
7. 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.
8. K. Y. Liu, J. C. Lui, and Z.-L. Zhang, "Distributed algorithm for serivce replication in service overlay network." unpublished. http://www.cse.cuhk.edu.hk/˜ykliu/research/dason.ps.