AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering

Qi Han and Nalini Venkatasubramanian
Department of Information and Computer Science
University of California-Irvine, Irvine, CA 92697-3425
{qhan,nalini}@ics.uci.edu

Abstract: In this paper, we integrate components for (a) resource provisioning and (b)information collection that allows for cost-effective enforcement of application QoS while tolerating imprecision of system state information. We discuss a family of policies for each component and provide mechanisms for dynamically selecting the appropriate combinations of the policy. Our objective is to select suitable management policies that maximizes the number of concurrent supported users while minimizing the overhead needed to ensure QoS enforcement for these users. Our performance results indicate that the "best" combination of policies is dependent on existing system conditions. However, we generally observe that the highly adaptive dynamic range-based information collection mechanisms exhibit superior performance under most system conditions and user workloads. All of these observations are applied into our integrated middleware framework AutoSeC (Automatic Service Composition). AutoSeC provides support for dynamic service brokering that ensures effective utilization of system resources in a distributed environment.

Keywords: information collection, resource provisioning, middleware framework, service broker

1 Introduction

Traditional network management software manages network resources by monitoring and analyzing network status and activities. In the future, network complexity will increase as the connectivity amongst service providers and clients requesting services increases; data, resources and services on the network are replicated; and applications have Quality of Service (QoS) requirements. To deal with the increasing network complexity, more attention must be paid to providing the system designers/administrators with tools to securely and dynamically manage an adaptable network infrastructure while ensuring user QoS. A set of intelligent system management middleware services is absolutely critical to designing the tools for efficient network management. Fundamental to all the issues in providing these services is the need for mechanisms that allow applications to obtain real-time information about system structure and state, use that information to make configuration decisions, and be notified when information changes. This paper is a step toward this goal.

Increasingly distributed applications have QoS requirements that can be translated to system level resource requirements. Resource provisioning algorithms such as QoS based network routing and server selection utilize current system state information to ensure that applications meet their QoS requirements. The effectiveness of a resource provisioning algorithm depends on a reasonably good approximation of the current resource utilization map, which is provided by a suitable information collection algorithm. Several recent studies evaluate the impact of stale link state on the performance of QoS routing [21], and their simulation results show that the

routing algorithm, network topology, link-state update policy each have a significant impact on the probability of successfully routing new requests, as well as the overheads of ditributing network load metrics. However, given these insights, a family of resource provisioning algorithms and a set of information collection policies, it remains a very complex process or nightmare for the users or application developers to decide which policies to use.

The objective of this paper is to explore effective middleware infrastructures that can be used to support efficient QoS-based resource provisioning. We propose a management framework named AutoSeC (**Auto**matic **S**ervice **C**omposition) that can dynamically select an appropriate combination of information collection and resource provisioning policies based on current system conditions and user requirements. AutoSeC is an important step in understanding and controlling the complex dynamics of QoS-based resource provisioning under different approximation of state information. It facilitates the QoS delivery and relieves the application developer/system administrator from the tedious selection among the families of policies. One of the main components of AutoSeC is the directory service that holds system information. An important issue in managing the directory service is to cost-effectively maintain accurate system state information. We study a family of information collection strategies that use varying data representations and diverse sampling policies. We also study resource provisioning algorithms for dealing with QoS-sensitive requests. Specifically, we analyze the traditional server selection mechanisms (load-based, proximity-based) and the more complex CPSS (combined path and server selection) mechanism [1] that takes both path and server status into consideration. Furthermore, we integrate information collection mechanisms with resource provisioning algorithms. The impact of information collection on resource provisioning is measured in terms of three metrics - request success ratios for multimedia requests with stringent QoS requirements, system overhead involved in maintaining the directory service and overall performance efficiency. We evaluate the impact by combining existing information collection and resource provisioning techniques with varying network/server conditions and under different application workloads. Based on the interaction of information collection and resource provisioning algorithms, we develop a set of dynamic service component composition rules that AutoSeC follows.

The rest of this paper is organized as follows. Section 2 presents the dynamic service broker framework AutoSeC which is a middleware service that facilitates the selection of a information collection policy and a resource provisioning policy based on current system conditions and characteristics of incoming requests. In Section 3 we discuss four different information collection strategies. Section 4 describes server selection algorithms and the CPSS algorithm. We evaluate the impact of four information collection algorithms on the performance of resource provisioning and discuss the results in Section 5. Based on the observations from Section 5, we propose several dynamic service selection rules which are stated. We conclude in Section 6 with related work and future research directions.

2 AutoSeC: Dynamic Service Broker Framework

In highly dynamic environments(e.g. mobile , multiservice networks, e-services), resource management mechanisms must be able to tolerate some information imprecision and be able to work effectively with approximate system state information. Our objective is to provide a user-transparent middleware infrastructure that supports automatic and adaptive selection of optimal combinations of information collection and resource provisioning services, thus sheltering the applications from the complexity of decision making and hiding the diversity of underlying provisioning services. We provide an overview of our framework AutoSeC in Figure 1. Details of the components in AutoSeC are described below.

The *directory service* holds system state information about network parameters (such as residual link bandwidth, end-to-end delay on links, etc.), server parameters (such as CPU utilization, buffer capacity, disk bandwidth, etc.) and client parameters (such as client capacity, connectivity, etc). We support adaptive parameter

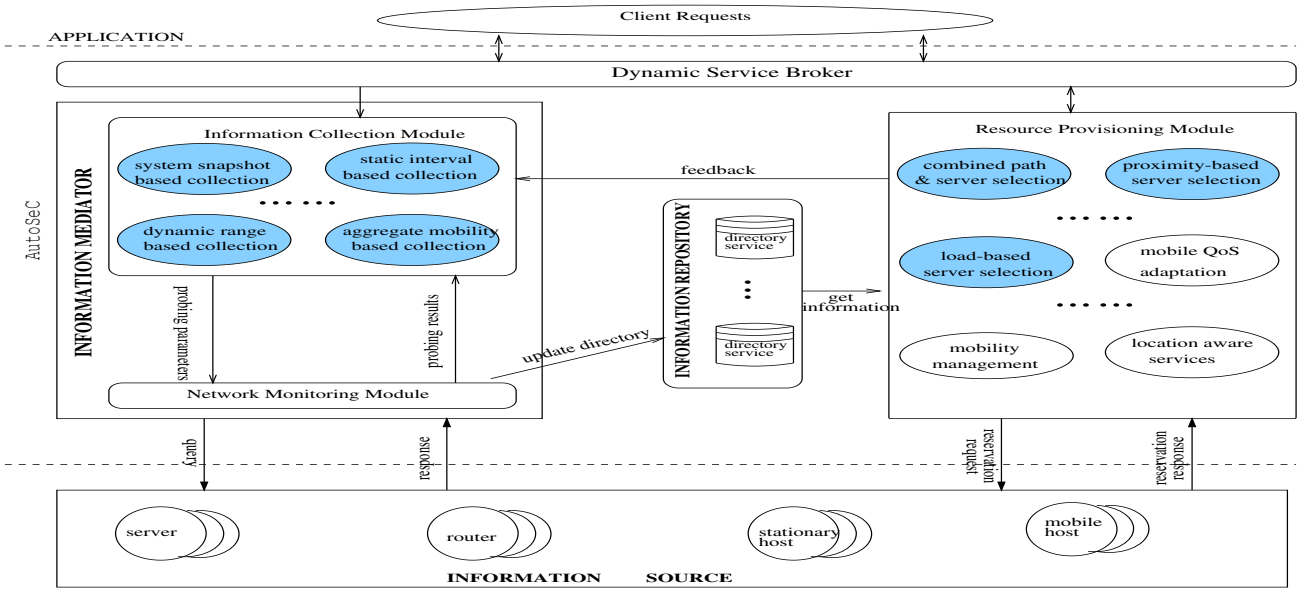


Figure 1: The AutoSeC Dynamic Service Broker Framework

representation within a centralized directory service. Each parameter can be represented by an instantaneous value or a range. In a range-based representation, each parameter is described by an interval bounded by an upper and a lower value. The information in the directory service is updated based on current network conditions and user requirements using different update policies to be discussed in the following section.

The *information collection module* utilizes client, server and network information in the directory service to determine whether to update the directory with the current system image. This involves possibly altering the range of the value in the directory service. To maintain a current system image, the information collection module determines the frequency at which samples are collected. The sampling interval impacts the overhead introduced by the information collection process. Higher accuracy of information in the directory service leads to better resource provisioning, however this implies more frequent sampling that can entail significant overhead. This module maintains a balance between the information accuracy and the directory service maintenance overhead.

The *resource provisioning module* uses information held in the directory service and feedback from resource provisioning module to perform admission control and resource allocation. Given that resources and services are replicated with the network, intelligent mechanisms can be applied to select appropriate resources (network paths, servers etc.) to service the incoming QoS based requests. The resource provisioning module considers current network and server utilization factors while allocating resources within the network to ensure better overall performance. Given the server and/or path assignment, the client proceeds to set up a connection along the assigned network path to the server. During the connection setup, the routers and the servers along the path check their residual capacity and either admit the connection by reserving resources or reject the request. When the connection terminates the client sends the termination requests, and resources are reclaimed along the connection path.

The *network monitoring modules* are distributed in the network and each module monitors portions of the entire domain based on sampling parameters supplied by the information collection module. The monitoring module issues probes to network routers and servers to collect the system state information such as current

policy	sampling period	parameter format
snapshot based(SS)	fixed	instantaneous value
static interval based(SI)	fixed	fixed range
throttle based(TR)	fixed/varying	varying range
time series(MA) based	varying	varying

Table 1: Family of Information Collection Policies Studied

residual bandwidth capacity. The probes consolidate the collected sample values and then hand them to the monitoring module. The monitoring module returns the collected results to the information collection module that updates the directory service.

The *dynamic service broker* decides the best combinations of resource provisioning and information collection policies to match current user requests and system conditions. It also determines if and when to switch among the different policies. In the following section, we discuss several strategies for information collection and resource provisioning. We evaluate the impact of the different strategies on the overall system performance under varying application workloads and develop service composition rules (heuristics).

3 Network and Server Information Collection Policies

Resource provisioning algorithms are based on the knowledge of the network topology, replica map and load information of network links and distributed servers. The network topology can be maintained by routing information exchange, a replica map can be obtained from a distributed domain name service either in an on demand mode or a caching mode, server state can be maintained using push or pull techniques. To deal with the dynamic changes in network and server conditions, an information collection process must be integrated and evaluated carefully together with the overall resource provisioning performance.

Information collection involves two steps: (1) data sampling - sampling period can be fixed (i.e.periodically) and also vary; (2) data updating - data in the directory service is updated based on its data representation such as instantaneous values or range based. We study four different policies for information collection shown in Table 1: (1) system snapshot based information collection (SI); (2) static interval based(SI); (3) throttle based(TR); (4) time series(MA) based.

Although server status is often stable compared to network information, in highly dynamic environments, servers are autonomous entities, not all activities or end servers can be brokered or monitored, it is still necessary to collect server information. The first three collection policies are also suitable for server information collection.

3.1 System Snapshot Based Information Collection

In this policy, information about the residue capacity of network nodes and server nodes is based on an absolute value obtained from a periodic snapshot [22]. During each sampling period, probing is initiated to gather the current information of router nodes; the directory is updated with the collected values. The sampling period solely determines the accuracy of the information stored in the directory. In highly dynamic dynamic traffic, the monitoring module has to sample at a very high frequency to prevent information from being outdated.

3.2 Static Interval Based Information Collection

In this policy, the residue capacity information is collected using a lower bound L, and an upper bound H, and the actual value is assumed to be uniformly distributed in this range with the expected value *(H-L)/2* [1]. We define a fixed interval: B and the residue capacity information is represented using a range $< kB, (k+1)B >$ with $k >= 0$. For each sampling period, probing is initiated to get the current information from router nodes. If the obtained value is out of the current range, the directory is updated with another interval based range, otherwise no update is needed.

We now discuss two dynamic range based information collection policies: throttle based(TR) and time series (MA) based approaches.

3.3 Throttle Based Information Collection

In this policy, the directory holds a range of the monitored parameter with upper and lower bounds that can vary dynamically. If the sampled value falls into the current range for a specified time period, we tighten the range with a pre-defined ratio. Otherwise, we relax the current range to hold the enlarged values observed during the previous sampling period. The collection process in Figure 2 represents a state machine of four states:

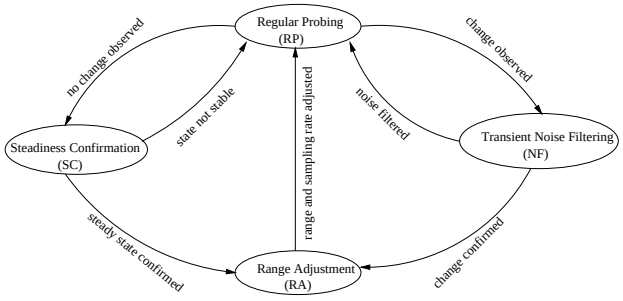


Figure 2: State Transition Diagram for Dynamic Range Based Information Collection

Regular Probing(RP), Steadiness Identification(SI), Transient Noise Filtering(NF) and Range Adjusting(RA). Initially the system is in RP state. We now describe the state transitions caused by the throttle based collection algorithm.

A. *RP state:*The directory holds an active range R_a with upper and lower values $U(R_a)$ and $L(R_a)$ of the link bandwidth. The monitoring module sends out probes periodically according to the initial probing period to collect link status information. Upon getting the sample, s, with the link's status value $v(s)$ and a time stamp $t(s)$, the monitoring module updates its local history table with the value $v(s)$ and the time stamp $t(s)$. And then the monitoring module compares $v(s)$ with R_a to determine the state transition.

A.1 If the value $v(s)$ falls into the existing active range R_a, no change is observed. Proceed to SI state to see if the link state has been steady for some time.

A.2 If $v(s)$ falls out of R_a, change is observed, we proceed to look into it further by entering the NF state.

B. *SI state:*We decide if range R_a should be tightened to provide higher accuracy. To do this, we search the history table in inverse time order to find out the time period tp during which the sampled values fall into R_a.

B.1 if $tp \geq T_s$, where T_s is a given threshold parameter then we anticipate more steadiness for the link in the future, so we proceed to RA state to tighten current range.

B.2 if $tp < T_s$, the process goes back to the RP state and waits for more input.

C. *RA state:* The range size is adjusted based on the values we sampled so far. For the case of range tightening triggered by the SI state, we need to do the following:

- Approximate the next value v_n by taking latest sample values within the specified counting period T_c, and compute the average:
  $v_n = AVG(v(s_i))$, where $t(s_i) > t(current) - T_c$;
- Tighten the range according to a predefined tightening ratio TR to get a new range R_n:
  $R_n = R_a * TR$;
- Adjust the new range R_n such that the average value v_n lies in the middle of the new range:
  $U(R_n) = min(v_n + 0.5 * R_n, capacity(link))$,
  $L(R_n) = max(v_n - 0.5 * R_n, 0)$.

After the range adjustment, the protocol goes back to the regular probing state using the new range $R_a = R_n$.

D *NF state:* The collection process enters the NF state from the RP state when the deviation of the observed value from the predicted value is outside the range identified by R_a. In practice, there could be several reasons for a miss: measurement error, a transient burst or a significant load level change. In order to assist the underlying model to adapt to a confirmed change and filter out high frequency traffic components, the monitoring module simply sends more probes over a probing period P_p, at a probing rate P_r. We take the average of these additional sampled values to see if it falls within the range R_a.

D.1 If yes, we go back to RP state.

D.2 Otherwise, significant change is confirmed, we proceed to the RA state to expand the range like this:
$U(R_n) = max(v(s_i), U(R_a))$,
$L(R_n) = min(v(s_i), L(R_a))$,
where $i = 1, 2, .., \frac{P_p}{P_r}$ and $t(s_i) > t(current) - T_c$.

3.4 Time Series(MA) Based Information Collection

Time series analysis has been used frequently to model network traffic. In this policy, We use a two-phase information collection process that uses simple statistical analysis techniques based on time-series. First, we derive a range such that the deviation between the predicted and observed values remains in the range with a given confidence level. Based on the size of the range and the confidence level, we determine a bound on the sampling rate. Second, the information collection process dynamically adjusts the range as well as the sampling rate based on the burstiness of the incoming traffic. For detailed discussion, please refer to [15].

In the MA model, a stationary stochastic process Z_t is modeled as a linear combination of Gaussian random variables, a_t, that follows a normal distribution $N(0, \sigma_a^2)$ with mean 0 and variance σ_a^2:

$$Z_t = \mu + \sum_{i=0}^{\infty} \psi_i a_{t-i} \qquad \psi_0 = 1, |\psi_i| < 1 \text{ for } i > 0 \tag{1}$$

where μ is the mean of Z_t and ψ_i is the MA model parameter. The random variable a_t is called the noise input at time t. It is also called an estimation residue which is the deviation between actual value Z_t and the prediction.

Suppose we are at time t, the least square error prediction of Z_{t+l} can be estimated by setting future noise input a_{t+1},...,a_{t+l} to be their expected value to zero. That is,

$$\hat{Z}_t(l) = \mu + \sum_{i=l}^{\infty} \psi_i a_{t+l-i} \tag{2}$$

Based on the MA model for network traffic, we can establish a bound on the dynamic range parameter R^* and the sampling interval T^*. Let ε be a confidence level between 0 and 1, and let R^* be defined as:

$$R^* = \min\Big\{R \Big| Pr(|Z_{t+1} - \hat{Z}_t(1)| > R) < \varepsilon\Big\}$$

If Z_t follows the MA model, the following holds:

$$R^* \geq \phi(1 - \varepsilon/2) \cdot \sigma_a \tag{3}$$

where $\phi(\cdot)$ is the inverse cdf of $N(0, 1)$.

Due to the bursty nature of network traffic, series a_t exhibits highly dynamic local behavior. In order to capture this local behavior, we use $\hat{\sigma}_a(t)$ instead of σ_a to derive R^*.

$$\hat{\sigma}_a(t) = \Big(\frac{1}{M} \cdot \int_{t-M+1}^{t} a_x^2 dx\Big)^{1/2} \qquad M > 0, \tag{4}$$

Here, M is the memory size which determines how long ago the prediction model should remember.
Given ε, the maximum sampling interval for range size R^*, which is derived from (3), is

$$T_t^* = \Psi^{-1}\Big(\frac{\hat{\sigma}_a^2}{\sigma_a^2} \cdot \frac{\phi^2(1 - \varepsilon/2)}{\phi^2(1 - \varepsilon)}\Big) \qquad T_t^* \geq T_{min} \tag{5}$$

$$\Psi(t) = \int_0^t \psi_x^2 dx \tag{6}$$

and ψ_x is the continuous form of our MA model parameters. To limit the maximum sampling cost, we set the parameter T_{min} as a cutoff limit imposed on the adaptive sampling period.

We enhance the state transitions in Figure 2 as follows:

- Sampling interval is adjusted using Equation (5) and link range is adapted using Equation (3);
- The way to decide if range R_a should be tightened is: calculate $\hat{\sigma}_a^{new}$ based on samples currently in the memory table by Equation (4). Suppose the old estimation variance that generated the current active range R_a is $\hat{\sigma}_a^{old}$, if $\hat{\sigma}_a^{new}/\hat{\sigma}_a^{old} \leq 1 - TH_s$, where $0 < TH_s < 1$ is between 0 and 1 which serves as one of the administrative tuning knobs, we anticipate more steadiness for the link. Otherwise, the new estimation variance hasn't been significantly reduced, the process goes back to the RP state and waits for more input.

The time series based approach has also been used to model server load in computationally bounded environments [9] and web server scenarios [2, 3] . The AR(16) model has been shown to provide the best fit for computation bounded servers [9].

4 Resource Provisioning Polices

Resource provisioning policies such as QoS based network routing and server selection help to effectively utilize network and server resources to ensure that applications meet their QoS requirements. In this section, we describe (a) server selection (load based); (b) server selection (proximity based); and (c) combined path and server selection (CPSS).

4.1 Server Selection(SVRS)

As distributed information services like the World Wide Web become increasingly popular on the Internet, scalability becomes a very serious issue. The performance degradation is caused by excessive server load due to the request of popular documents, wasted network bandwidth due to redundant document transfer, and excessive latency in delivering documents to the client due to the potential for transfers over slow paths. A promising technique that addresses all of these problems is service/document replication where the same data/service is partially replicated across distributed servers. However, when a service is replicated, clients must be able to discover an optimal replica. Server selection schemes attempt to choose the best replica/server for a given request.

In this section, we describe two server selection policies that we will evaluate in composition with various information collection techniques.

Least Utilization Factor Policy(SVRS-UF): The capacity of a server can be specified as four parameters: CPU cycles, memory buffers, I/O bandwidth and network transfer bandwidth as shown in [23]. The server resources needed by a request r are modeled as $< CPU_r, BUF_r, DB_r, X_r >$, the available server resources can be modeled as $< CPU^s_{avail}, BUF^s_{avail}, DB^s_{avail}, X^s_{avail} >$. The utilization factor (UF) for a server s, given a request r and a parameter n is defined as

$$UF(s,r,n) = \begin{cases} (max(\frac{1}{CPU^s_{avail}-CPU_r}, \frac{1}{BUF^s_{avail}-BUF_r}, \frac{1}{DB^s_{avail}-DB_r}, \frac{1}{X^s_{avail}-X_r}))^n \\ \qquad \text{if available server resources are greater than those requested} \\ \infty \\ \qquad \text{otherwise} \end{cases}$$

The bias factor n in the utilization factor serves as a turning knob, represents the degree to which a lightly loaded resource is favored over a congested resource.

This policy chooses the server with the minimal utilization factor. If there is a tie between the servers, the policy randomly pick one.

Shortest Hop Policy(SVRS-HOP): This policy chooses the nearest server in terms of the number of hops. If there is a tie between the servers, the policy chooses the least loaded one as in Least UF Policy described above.

4.2 Combined Path and Server Selection(CPSS)

Traditionally, the problem of effective resource utilization for networks and servers have been studied independently. At the network level, QoS routing techniques are used to improve the network utilization by balancing the load among the individual network links. At the server end, since data may be replicated across multiple servers, server selection policies direct the user to an optimal server that can handle the incoming requests for information. Server selection mechanisms often treat the networkpath leading from the client to the server as

static. The two techniques(QoS routing and server selection) can independently achieve some degree of load balancing. When applications are highly sensitive to QoS parameters, high-level provisioning mechanisms are required to address the route selection and server selection in a unified way. Such integrated mechanisms can potentially achieve higher system-wide utilization and therefore allow more concurrent users.

In order to deal with server and path selection in a unified way, we also utilization factor(UF) for network links to quantify the residue capacity of links. The UF for a link l with available link bandwidth BW^l_{avail}, given a request r with bandwidth requirement BW_r and a parameter n, is defined as

$$UF(l,r,n) = \begin{cases} (\frac{1}{BW^l_{avail} - BW_r})^n & \text{if } BW^l_{avail} > BW_r \\ \infty & \text{otherwise} \end{cases}$$

The bias factor n in the UF is the same as that for server UF described above.

The basic idea of CPSS algorithm is: given a client request with QoS requirements, we select the server and links that maximize the overall utilization of resources. It allows load balancing not only between replicated servers, but also among network links to maximize the request success ratio and system throughput. For detailed discussion, refer to [14].

Given a client request with QoS requirements $r :< BW_r, CPU_r, BUF_r, DB_r, DL_r >$, an assignment X=$\{p,s\}$ is feasible if and only if the available resources in both server and links are greater than requested. We define a feasible set X_f as set of all the assignments that meet the feasibility condition.

An assignment X^*=$\{p^*, s^*\}$, is optimal if and only if it satisfies the feasibility condition and a policy dependent optimality criteria. For instance, the optimality clause for the BEST UF policy is
$Dist(s^*, r, n) = Min\{UF(s^*, r, n) + UF(p^*, r, n) \forall s \in S$.

5 Performance Evaluation

The objective of the simulation is to determine the best combination of information collection policies (SS, SI, TR, MA) and resource provisioning policy(SVRS-HOP, SVRS-UF, CPSS) under varying application workload.

As we observe, application workload can be classified into two categories based on the types of requests sent to network:

- pure QoS requests: Some servers are designed for a specific purpose. For example, VoD servers only accept multimedia requests, computation bounded servers only accept requests for scientific computation. As mentioned before, for these requests with QoS requirements, AutoSeC works as the mediator and thus all the changes on the server load are exposed to AutoSeC.

- mixed requests: Some servers are general-purpose, therefore accept all types of requests regardless of whether it has QoS requirements. For example, distance learning servers are designed to be of this type, it delivers live course presentation as well as text etc. Generally there is no necessity for requests without QoS requirements to go through AutoSeC, which makes the server load change without the notice of AutoSeC.

Table 2 depicts six different cases we studied in this paper.

To compare those above algorithms under different scenarios, we use three metrics in this paper:

- request success ratio: it is the ratio of the number of successful requests over the the number of whole requests;

application workload	CPSS	SVRS-HOP	SVRS-UF
pure QoS requests	Case A1	Case B11	Case B21
mixed requests	Case A2	Case B12	Case B22

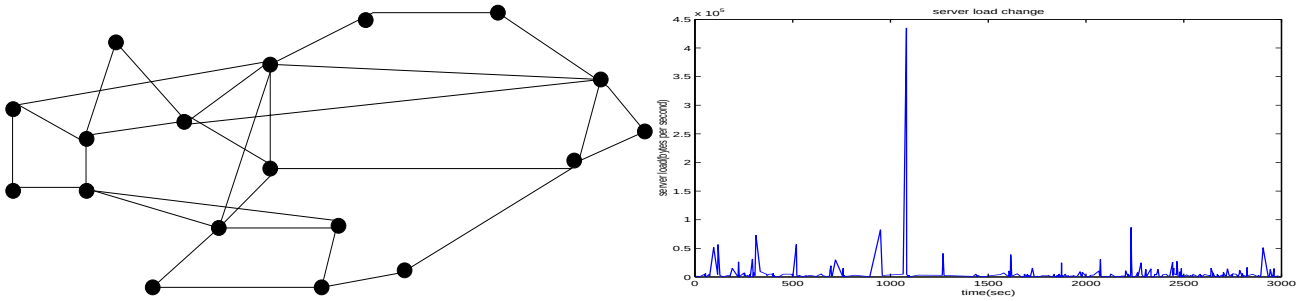
Table 2: Different Cases of Performance Evaluation

- information collection overhead: the monitoring module samples the network and then update the directory service when necessary, so the information collecting overhead is dependent on the sampling and update overhead which requires both network bandwidth and computation at the host. For simplicity, we assume the cost of information collection is proportional to N_{du}(the number of directory updates) and N_s (the number of samplings);
- overall performance efficiency: Given the number of successful requests N_{sr} over a fixed total number of same requests, the overall performance efficiency is $N_{sr}/(N_{du} + N_s)$.

5.1 The Simulation Environment

Our simulator is a message driven multi-threaded simulator intended to study the dynamics of QoS sensitive network environments. It has four components: traffic source, routers, servers and directory service nodes.
Topology and System Configuration: In the simulation, we use the following typical ISP network topology with 18 nodes and 30 links shown in Figure 3A. We assume that each node is a network router, and that the clients and servers are distributed among the network and are directly behind the router nodes (not shown in the graph). The topology is chosen such that there are a large number of alternative paths between source and destinations nodes.

As stated before, server load might change without the notice of the resource provisioning module, we have to take this factor into consideration, therefore, we use the trace [4] to represent the server load change rather than generating web server load using analytical models [2, 3]. The trace contains a day's worth of all HTTP requests to the EPA WWW server located at Research Triangle Park, NC. Since we are only interested in the impact of information collecting policies on resource provisioning algorithms, this representative of server load change should be enough for our evaluation. The change pattern is shown in Figure 3B.



A.network topology B.server load change pattern

Figure 3: Simulation Environment

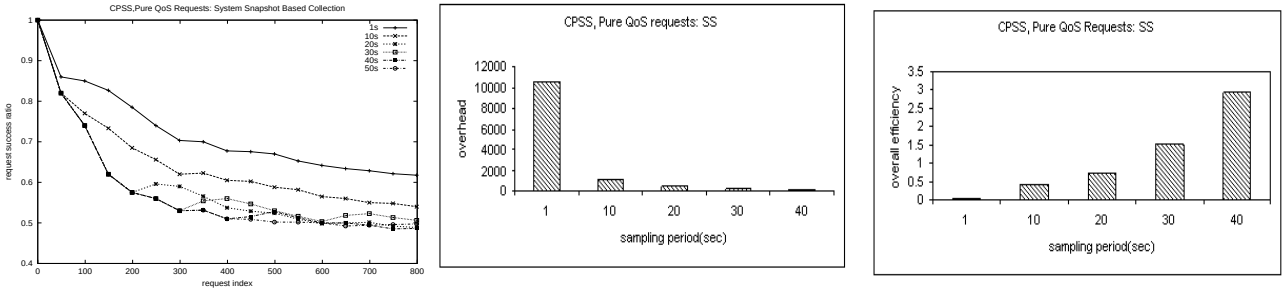
To better emulate the real network, the capacities of network links are selected from various widely used link

types from 1.5Mbps to 155Mbps, with the mean value being 64M. When defining the capacity of the server nodes, we calibrate CPU units using a basic 64Kbit voice processing application, memory units to be 64Kbytes, and disk bandwidth units to be 8Kbytes/s. The server capacities are also selected from popular models of multimedia servers and web servers, with the CPU, memory, disk bandwidth mean to be 1845, 6871 and 5770 calibrated units respectively.

Request and Traffic Generation Model: We model request arrival at the source nodes as a Poisson distribution, and the request holding time is exponentially distributed with a pre-specified average value. The request holding time is the time for which the requested network and server resources such as link bandwidth, CPU, buffer, disk bandwidth etc. are reserved. We pre-define a set of client request templates to capture typical multimedia connection request patterns in terms of network bandwidth, CPU, memory, disk bandwidth and end-to-end delay. For each request generated, the requested parameters are randomly selected from the set of request templates, with the mean requested bandwidth being 2.5Mbps, mean end-to-end-delay being 400ms and CPU, memory and disk bandwidth being 150, 374 and 271 calibrated units respectively. To represent non-uniform traffic, we designate some sets of candidate destinations as being "hot", (i.e. serving popular videos, web sites etc), and they are selected by the clients more frequently than others. To reduce the effect of local and nearby requests, we choose three pairs of source-destination sets from the topology. The requests arrive to these hot pairs, as foreground traffic, at a higher rate than other background traffic. In our non-uniform traffic pattern, we set the foreground arrival rate to be 5 times higher than the background rate, and in uniform traffic pattern, we set them to be equal to each other. Specifically we set the foreground arrival rate to be 10 seconds, and the background rate to 50 seconds.

5.2 Impact of Information Collection on CPSS

In this section, we analyze the impact of information collection mechanisms on the overall CPSS performance. We first analyze the performance of each collection algorithm individually and then compare the performance of the four algorithms under similar conditions.

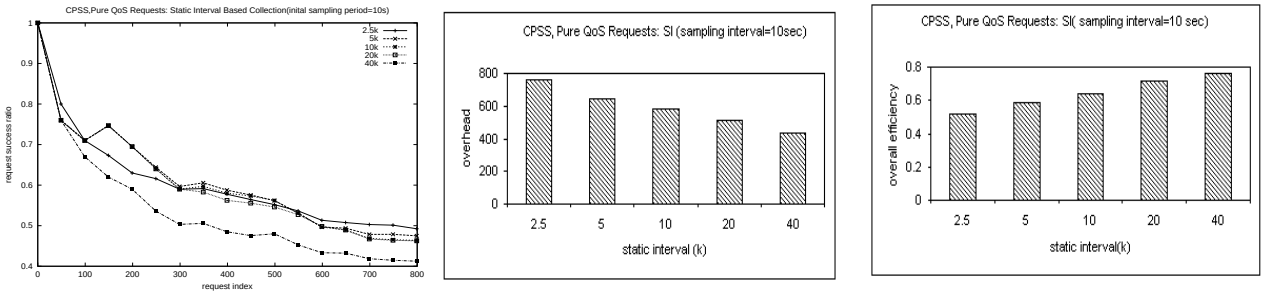


A.request success ratio B. overhead C. overall performance efficiency

Figure 4: *Case A1(CPSS, pure QoS requests)-Snapshot based collection:* Fewer requests are accepted with the increase of arrival request; shorter sampling period results in higher request success ratio and higher overhead.

System Snapshot Based Information Collection: The snapshot based information collection policy can be regarded as a special case of the interval based policies by setting the range to 1. Figure 4 shows that for snapshot-based information collection, in general, we observe that as more requests arrive and compete for network and server resources, fewer requests can get through successfully. A shorter update period results in a higher request success ratio than a larger update period. However, a shorter update period implies that more

probes are initiated for sampling causing an increased sampling and directory update overhead, leading to poorer performance efficiency. With a larger update period, the snapshot policy causes decreased request success ratios since the policy retains a single (inaccurate) value for a larger duration of time causing increased inaccuracy of system state.

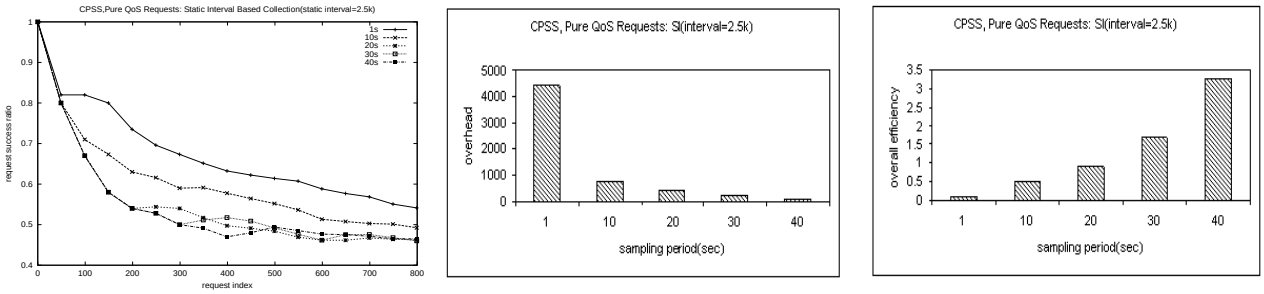


A.request success ratio B. overhead C. overall performance efficiency

Figure 5: *Case A1(CPSS, pure QoS requests)-Static Interval based Collection With Varying Intervals:* With the same sampling period, a smaller interval results in higher request success ratios, but also causes higher overhead.

Static Interval Based Information Collection: In order to reduce the cost of information collection, we introduce interval based collection policies, where the reduction in message overhead cost as compared to the snapshot policy can be seen by comparing Figure 4B and Figure 5B. We study the static interval based policy with (a) large and small intervals (static ranges) and (b) frequent and infrequent sampling.

We can see from Figure 5 that a larger interval(range) leads to lower update overhead since the sampled values frequently fall within the range. However, a smaller interval brings better request success ratios since representing a residual value using a bigger interval introduces information inaccuracy. In other words, a smaller range provides more accuracy at the cost of higher overhead, thus the overall performance is not necessarily better.

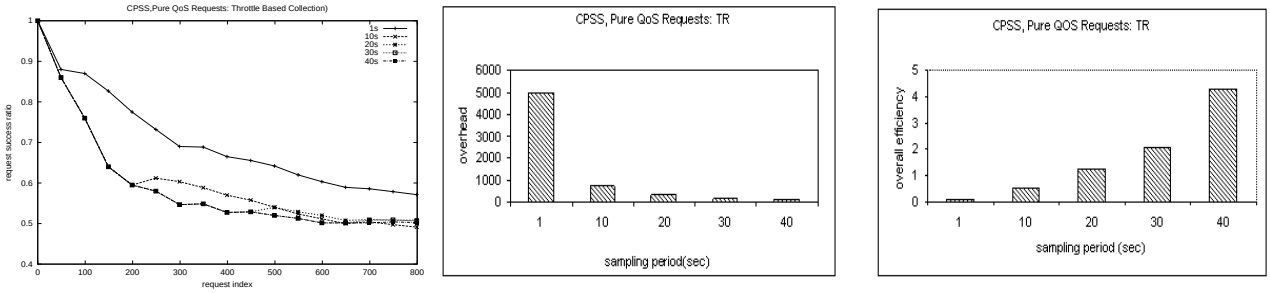


A.request success ratio B. overhead C. overall performance efficiency

Figure 6: *Case A1(CPSS, pure QoS requests)-Static Interval based Collection With Varying Sampling Periods:* With the same static interval, shorter sampling period leads to higher request success ratio and higher overhead.

Figure 6 studies the static interval policy under different sampling periods (with the same range). We can see that very small sampling periods (e.g. 1 sec) exhibit higher request success ratios at the cost of higher overhead. With the increase of the sampling period, the success ratio decreases. This is because with the larger

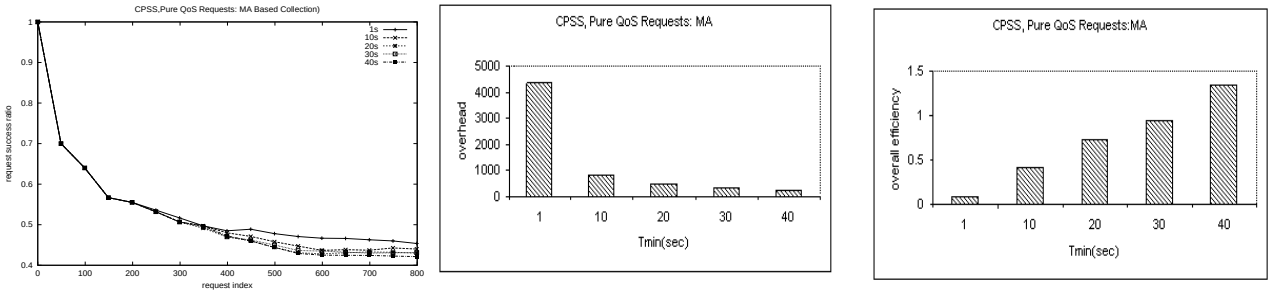
update period, the residual value represented using the interval remains out-dated longer than that with a smaller sampling period, resulting in a more inaccurate system state information. Interestingly, we notice that when the sampling period is large enough (30 and 40 seconds here), the variations of the sampling period do not exhibit an obvious influence on the request success ratio. This is due to the fact that beyond a certain period, almost all information kept in the directory is inaccurate.



A.request success ratio B. overhead C. overall performance efficiency

Figure 7: *Case A1(CPSS, pure QoS requests)-Throttle based Collection:* very small sampling period leads to high request success ratio, but when sampling period is big enough, it results in similar request success ratio.

Throttle Based Information Collection: Figure 7 shows that a very small sampling period(such as 1 second in our results) brings high request success ratios, at the cost of increased overhead. This is because with very frequent sampling, accurate system state information can be obtained; however, this causes a thrashing effect, i.e., it makes the dynamic range change very frequently leading to frequent directory update. When the sampling periods are large enough(like 10, 20, 30,40 seconds in our simulations), not much difference is observed in terms of request success ratio, because the inaccurate system state information is adjusted always by manipulating the range and the directory is updated. A large sampling period also causes fewer sampling probes and also fewer directory updates due to the self-similarity of the network traffic, therefore a larger sampling period brings better overall performance efficiency.



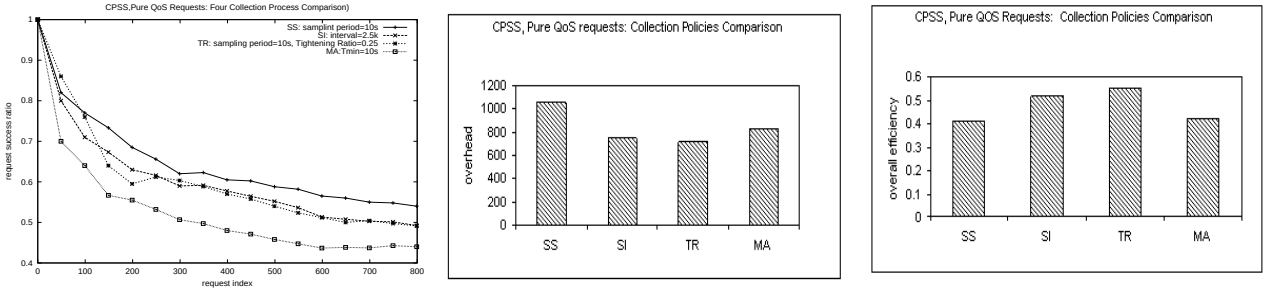
A.request success ratio B. overhead C. overall performance efficiency

Figure 8: *Case A1(CPSS, pure QoS requests)-Time Series(MA) based Collection:* very small T_{min} leads to high request success ratio, but when T_{min} is big enough, it results in similar request success ratio.

Time-Series(MA) Based Information Collection: Figure 8 shows the impact of MA based information collection policy under different T_{min}. We observe similar results to that of throttle- based collection and attribute

the results to the same factors.

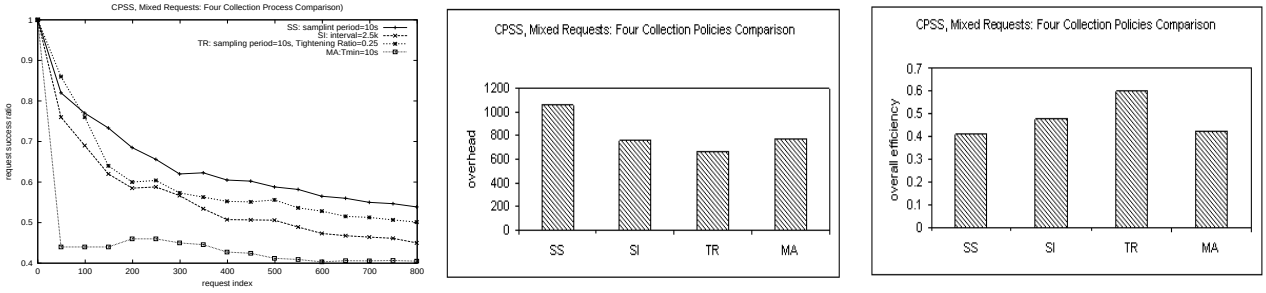
Next, we compare the performance of the four information collection policies with the CPSS algorithm under similar conditions. To make a fair comparison, we choose 10 seconds as the sampling period for snapshot based, static interval based and throttle based collection policies and as the T_{min} for MA based collection policy.



A.request success ratio B. overhead C. overall performance efficiency

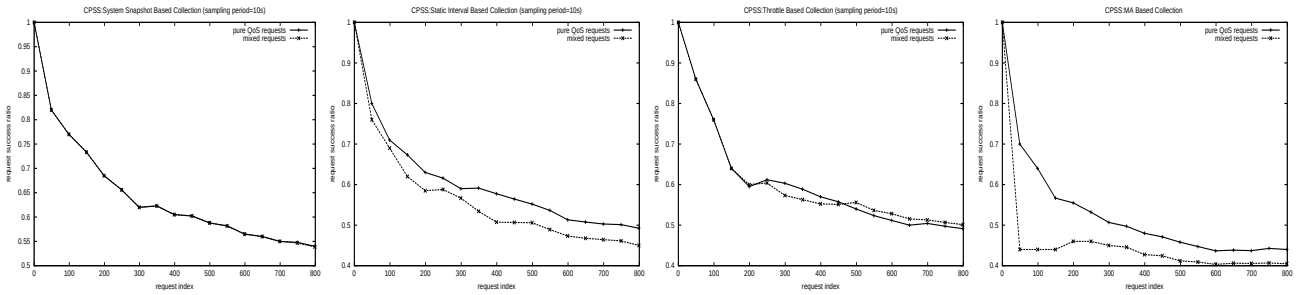
Figure 9: *Case A1(CPSS, pure QoS requests)-Comparison of Four Information Collection Policies:* The snapshot based approach is very sensitive to the sampling period. Given the same sampling period, throttle based approach is superior to other three approaches in terms of performance efficiency.

Figure 9 shows that snapshot based approach is very sensitive to the sampling period; with frequent sampling, it performs better than the other policies in terms of request success ratio, but its overall efficiency is not as high as that of the other policies because of its higher overhead. With infrequent sampling, it performs much worse. While MA based policy does not perform as well as expected, the throttle-based collection algorithm performs uniformly better than the other algorithms both in terms of request success ratio, sampling and update overhead and overall efficiency. We believe that although the network traffic exhibits self-similarity, there still exist bursts, hence it is almost impossible to have a very accurate model to characterize and forecast network traffic (although the time series model has been proven to be reasonable). The time series (MA) based approach uses the inaccurate model to derive the sampling period and residual link bandwidth range, thereby aggravating the inaccuracy. Using prediction is always risky, so we believe the MA model based information collection could be superior to the other approaches sometimes, but not always. Our simulation is a good example.



A.request success ratio B. overhead C. overall performance efficiency

Figure 10: *Case A2(CPSS, mixed requests)-Comparison of Four Information Collection Policies:*Given the same sampling period, throttle based approach is superior to other three approaches in terms of performance efficiency.



A. snapshot based B. static interval based C. throttle based D.MA based

Figure 11: *CPSS, Information Collection Policies under Varying Application Workload:* No difference is observed for snapshot based approach under varying application workload; but for the other three approaches, when all requests go through the resource provisioning module, the request success ratio is higher.

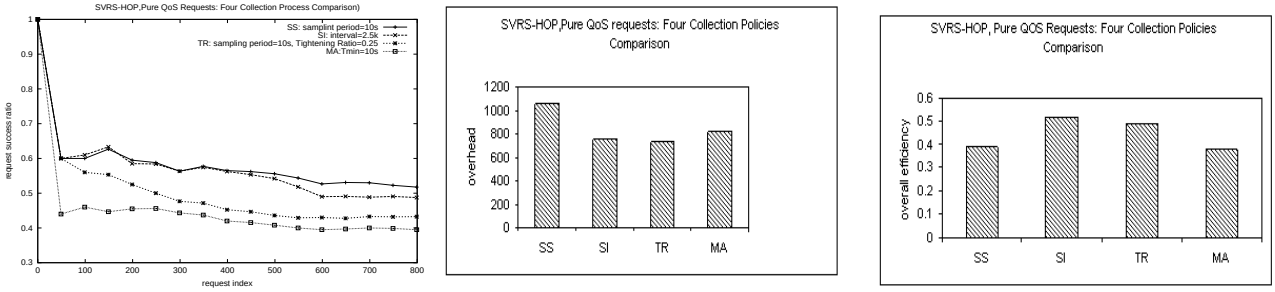
In case A2, the server load is subject to changes without prior notice to resource provisioning module. Figure 10 shows similar results to that of Case A1. Figure 11 depicts the impact of unreported server load changes; we observe that the snapshot based approach shows no difference in the presence of unreported server changes; the other three information collection policies uniformly work better in the case when all requests go through the resource provisioing module. This is because with the snapshot based policy, when the server load dynamically changes without notice, the instantaneous values are always obtained and stored in the directory. While for other policies, a range (static or dynamic) is stored in the directory to represent the parameter; when the server load changes without notice, it is possible that some small requests are accepted and while the the residual resource capacity falls into current range and more requests can be accepted. However, when trying to setup the requests along the path and the server, the server does not have enough resources and rejects the request.

5.3 Impact of Information Collection on Server Selection

The study of each independent information collection algorithm yielded similar results to that of the CPSS case (Case A1); hence in this section, we only focus on the comparison of the four information collection policies on server selection.

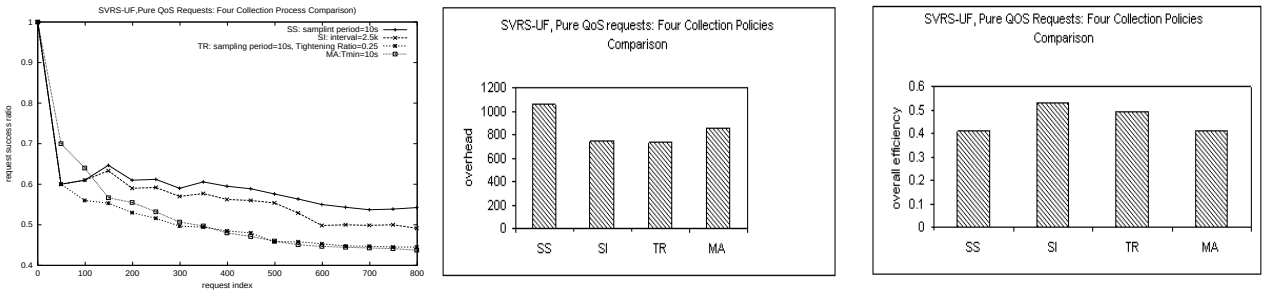
Figures 12 and 13 show that if all requests go through the resource provisioning module, throttle-based and MA model based approaches have similar request success ratios, but the overall performance efficiency of the throttle-based approach is higher. The static interval based algorithm results in higher request success ratio and overall efficiency than other three approaches for both least UF based and shortest hop based server selection algorithms. The reason is that with server selection, only server resource factors are considered in the provisioning algorithm, if all requests are reflected in the resource provisioning module, representing residual link bandwidth with a static interval is accurate enough, we don't need to use the more complex approaches.

Figures 14 and 15 shows that if not all requests go through the resource provisioning module, with fewer requests, the static interval based approach yields higher request success ratios and also higher a performance efficiency than the other dynamic range based approaches. But when more requests arrive, the request success ratio decreases and gets closer to the dynamic range based approaches. With a larger number of requests, the success ratio is more sensitive to the application workload change. Looking at the overall performance efficiency, the throttle-based algorithm is better for both least UF based and shortest hop based server selection algorithms



A.request success ratio B. overhead C. overall performance efficiency

Figure 12: *Case B11(SVRS-HOP, pure QoS requests)-Comparison of Four Information Collection Policies:* The static interval based algorithm results in higher request success ratio and overall efficiency than other three approaches; the throttle-based approach yields higher request success ratio than MA based approach.



A.request success ratio B. overhead C. overall performance efficiency

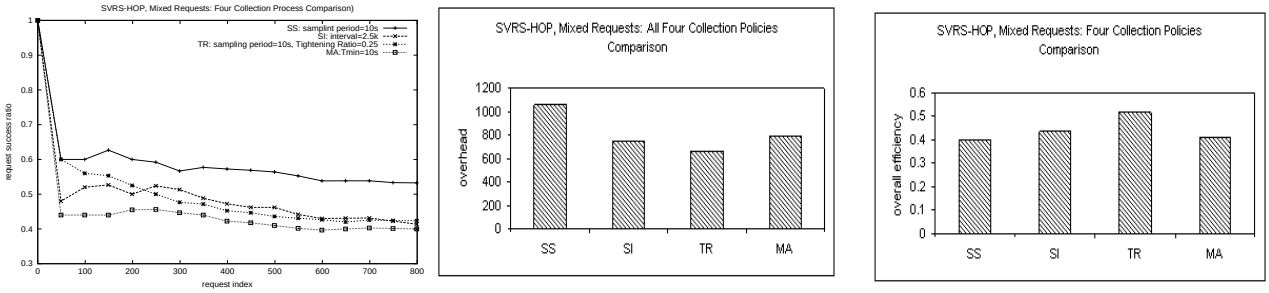
Figure 13: *Case B21(SVRS-UF, pure QoS requests)-Comparison of Four Information Collection Policies:* static interval based algorithm results in higher request success ratio and overall efficiency than other three approaches; throttle-based approach and MA based approach yield almost the same request success ratio.

than other approaches since with the similar request success ratio, the overhead of the throttle-based approach is lower.

5.4 Performance Summary

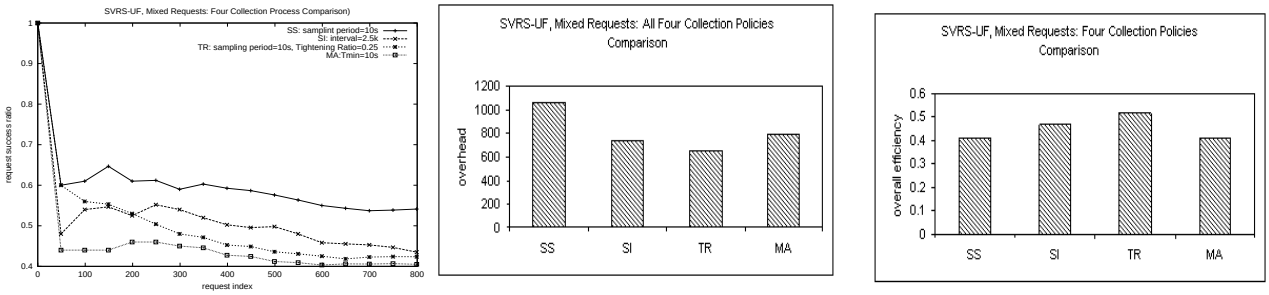
Our evaluation indicates that both the accuracy and overhead of information collection policies have a significant impact on the performance of resource provisioning process. Although snapshot based collection can obtain very accurate information, the huge overhead introduced by frequent sampling and directory updates makes it a bad choice. Due to high dynamicity of network traffic, it is almost impossible to have a very accurate model to characterize it quantitatively, the MA based collection does not always perform very well practically, in terms of how it impacts resource provisioning, although its theoretical analysis outperforms the others. In contrast, the throttle based algorithm turns out to be a very good choice in most cases. Although seemingly naive and ad-hoc, it adapts pretty well to the constantly changing environment.

For CPSS, the throttle based information collection seems to be the most suitable among all the policies we evaluated. For server selection, if all requests go through the resource provisioning module, static interval based



A.request success ratio B. overhead C. overall performance efficiency

Figure 14: *Case B12(SVRS-HOP, mixed requests)-Comparison of Four Information Collection Policies:* Throttle-based approach yields higher request success ratio than MA based approach, its performance efficiency is highest among all the approaches.



A.request success ratio B. overhead C. overall performance efficiency

Figure 15: *Case B22(SVRS-UF, mixed requests)-Comparison of Four Information Collection Policies:* Static interval based approach yields higher request success ratio than the dynamic range based approaches, but the overall performance efficiency of the throttle based approach is highest.

information collection is sufficient; otherwise, the throttle based approach works best. Table 3 lists the best combinations of information collection and resource provisioning policies under different application workload.

Automatic Service Composition Rules Using the results from the performance study, we derive general heuristics to select appropriate composition of information collection and resource provisioning policies for an incoming request. We classify the client requests into:

- web requests that are just normal HTTP, FTP or TELNET requests that do not present stringent and explicit QoS requirements;
- multimedia requests that are large scale graphics/images rendering requests, audio or video streaming requests;
- computation requests that are complex scientific computations requests and usually CPU bounded in nature.

	pure QoS requests	mixed requests
SVRS-HOP	SI	TR
SVRS-UF	SI	TR
CPSS	TR	TR

Table 3: Optimal Combinations of Information Collection and Resource Provisioning Policies

	server types			
request types	web	multimedia	computation	general-purpose
web request	n/a			n/a
multimedia request	*CPSS+TR*	*CPSS+TR*		*CPSS+TR*
computation request			*SVRS+SI*	*SVRS+TR*

Table 4: Service Component Selection Rules

We also classify servers into four types:

- web servers that support web requests and also multimedia requests;
- multimedia servers that deal with multimedia requests;
- computation servers that support high speed computations;
- general purpose servers that support all the different types of servers.

In the domain controlled by a service broker, there can be just pure web servers, pure multimedia servers, pure computation servers or general-purpose servers. Based on the observations of section 5, we propose the service component selection rules illustrated in Table 4. Performance results show that the given combination exhibits uniformly better performance than the others, therefore it is not necessary to switch policies on the fly.

6 Related Work and Future Research Directions

We describe related work in network management, resource provisioning and information collection.

Network Management: Network management projects focus on making both active and passive network statistics and information available to end-users. Recent systems that focus on the measurement of communication resources across Internet wide networks include Network Weather Services(NWS) [25] and topology-d [18]. NWS makes resource measurements to predict future resource availability, while topology-d computes the logical topology of a set of internet computation nodes. Both of these systems actively send messages to make communication measurements between pairs of computation nodes. The Remulac project and Remos in [8] is a generalizable resource monitoring system for network applications, applications interact with a portable interface to the network that includes flow query and logical query topology abstractions. It maintains both static and dynamically changing information based on SNMP measurements on the router nodes in the network. Network information discovery is also included in metacomputing system Globus [12]. The maintained information includes network activity, available network interface, processor characteristics, and authentication mechanisms. Besides providing more efficient network monitoring and measuring algorithms, our focus in this paper is more

on how to provide a middleware service that can help network-aware applications to choose the appropriate mechanism.

Resource Provisioning: Static nearest server algorithms [16] attempt to choose a replica server with the minimum number of hops from the client, with topology information being stored in a topology database. Such policies do not consider server quality as a scheduling parameter resulting in performance degradation in some situations. Dynamic server selection policies [13] have also been studied in the context of replicated web data, one such effort uses the notion of a Predicted Transfer Time(PTT), calculated using parameters such as the available bandwidth and roundtrip delay [5]; this was shown to reduce the response time in some cases. The study also showed that popular servers are heavily loaded and that server load played an important role in practice. The Application Layer Anycasting based protocol [11] aims to improve web performance using a predicted server response time using server and network path quality stored in a resolver. The anycasting work is closely related to ours since it considers both network path quality as well as server quality. While their focus in on generalized Web performance and large predicted response time parameters, our focus is on providing efficient QoS provisioning for multimedia applications, e.g. video/audio streaming, conferencing. Approaches to load-based task assignment in distributed server environments have been developed in [23]. It studied server scheduling for multimedia applications using server resources such as CPU, buffer, disk bandwidth and network interface card parameters to characterize the server quality.

Network resource provisioning has been studied in the context of QoS-based routing[7]. QoS-based extensions of the traditional Bellman-Ford/Dijkstra algorithm incorporate an available bandwidth parameter and use widest-shortest path calculations[6]. Experimental efforts studying the combined performance and overhead cost of different information collecting and routing policies[1] show that maintaining a large number of alternate paths and randomizing the path selection will yield better performance in most topologies and traffic patterns, especially when information update is very infrequent. Link parameters such as delay, available bandwidth, etc. can be described using probability distributions[19] and can be used to find a *most probable* path satisfying the requested bandwidth and end-to-end delay. A heuristic to the Most Probable-Optimal Partition problem has a complexity of $O(|E|^2 D)$; which uses a dynamic programming solution developed in the context of RSP problems[17]. Comparison of different QoS routing algorithms under specific network configurations has been studied, the interplay between link-state update policies, traffic patterns, and network topology are also studied [21].

Information Collection: Measurement architecture has been proposed in [13, 10, 20, 22]. [13] proposes a method to estimate the distance of any two points in the Internet. Linear model based information collecting and prediction infrastructure have been studied, and service APIs for upper level applications which needs such support in establishing network connections have been defined [10]. An interesting monitoring module placement strategy is proposed. By emphasizing the *hot points* in typical ISP network topology, it improves the efficiency of sampling process. In the diffserv architecture [26], state information collected in the directory is used by the bandwidth broker to statistically guarantee the negotiated QoS agreement. Possibly, the directories of adjacent domains can exchange aggregated state information to maintain a more accurate snapshot of the system.

The static range based algorithm used in this paper is proposed in [1]. In this paper, the QoS routing performance is studied in detail using different static range size and sampling intervals. They found that smaller range size performs better if sampling with short intervals , while bigger size gets more cost-effectiveness in longer intervals. Moving object databases deal with the modeling and tracking of a moving object within a database; information collection solutions have been proposed in this context that directly compare the cost of information imprecision with the cost of message passing and make decisions based on the difference [24].

Conclusions: In this paper, we present a middleware framework that helps to dynamically select appropriate information collection and resource provisioning polices in order to enhance system throughput and allow more concurrent users. Resource provisioning mechanisms studied in this paper include traditional server selection

mechanisms (load-based, proximity-based) as well as more complicated CPSS (combined path and server selection) mechanisms that perform composite routing and scheduling in a unified manner. The resource provisioning algorithms are based on state information held within a directory service component in the framework. We also implement a family of information collection techniques that maintain system state information within a directory service using either an instantaneous value or a range-based representation. The policies also differ in the degree of dynamicity with which current state information is obtained and updated. The policy selection rules are proposed based on the impact of varying information collection techniques and resource provisioning mechanisms on the overall performance of the distributed environment.

We will provide a set of network management middleware services for network-aware applications. Current network management techniques and mechanisms need to be adapted to provide a dynamic and relevant set of information to the middleware services. Network instrumentation and measurement infrastructure is also crucial to these middleware services. A promising extension is the incorporation of distributed directory services into the framework to deal with the scalability. We are also studying the applicability of this framework in designing network management mechanisms for mobile environments.

Acknowledgements We thank Zhenghua Fu for useful discussions that led to the framework described here and his contributions to designing the simulation environments.

References

[1] G. Apostolopoulos, R. Guerin, S. Kamat, and S.K. Tripathi. Quality of service based routing: A performance perspective. In *Proceedings of ACM SIGCOMM*, 1998.

[2] M. F. Arlitt and C.L. Williamson. Web server workload characterization: The search for invariants. In *Proceedings of the ACM SIGMETRICS*, 1996.

[3] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS*, 1998.

[4] L. Bottomley. Epa-http single www server trace. http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html.

[5] R.L. Carter and M.E.Crovella. Dynamic server selection using bandwidth probing in wide-area networks. In *Proceedings of IEEE InfoCom*, 1997.

[6] S. Chen and K.Nahrstedt. On finding multi-constrained paths. In *Proceedings of IEEE International Conference on Communications*, June 1998.

[7] S. Chen and K.Nahrstedt. Distributed qos routing in ad-hoc networks. *IEEE JSAC, Special Issue on Ad-hoc networks*, 1999.

[8] T. Dewitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, and J. Subholk. Remos: A resrouce monitoring system for network aware applications. Technical Report CMU-CS-97-194, CMU, 1997.

[9] Peter A. Dinda. The statistical properties of host load. Technical Report CMU-CS-98-175, CMU, 99.

[10] Peter A. Dinda and D. R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, CMU, 1999.

[11] Z. Fei, S. Hnattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of IEEE InfoCom*, 1997.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal on SuperComputer Applications*, 11(2), 1997.

[13] P. Francis, S. Jamin, V. Pasxon, L. Zhang, D. Gryniewica, and Y. Jin. An architecture for a global internet host distance estimation service. In *Proceedings of IEEE InfoCom*, 1999.

[14] Z. Fu and N. Venkatasubramanian. Combined path and server selection in dynamic multimedia environments. In *Proceedings of ACM Multimedia*, Orlando, FL, 1999.

[15] Z. Fu and N. Venkatasubramanian. Adaptive parameter collection in dynamic distributed environments. In *Proceedings of IEEE ICDCS*, 2001.

[16] J.D. Guyton and M.F.Shwartz. Locating nearby copies of replicated internet services. In *Proceedings of ACM SIGCOMM*, 1995.

[17] R. Hassin. Approximation schemes for the restricted shortest path problem. *Math of Operations Research*, 1992.

[18] K.Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. Technical Report TR97-660, USC, Dept. of CS, October 1997.

[19] D.H. Lorenz and A. Orda. Qos routing in networks with uncertain parameters. In *Proceedings of IEEE InfoCom*, 1998.

[20] Nancy Miller and Peter Steenkiste. Collecting network status information for netowrk-aware applications. In *Proceedings of IEEE InfoCom*, 1999.

[21] A. Shaikh, J. Rexford, and K. G. Shin. Evaluating the impact of stale link state on quality-of-service routing. *IEEE/ACM Transactions on Networking*, 9(2):162–176, April 2001.

[22] M. Stemm, R. Katz, and S. Seshan. A network measurement architecture for adaptive applications. In *Proceedings of IEEE InfoCom*, 2000.

[23] N. Venkatasubramanian and S. Ramanathan. Load management for distributed video servers. In *Proceedings of IEEE ICDCS*, May 1997.

[24] O. Wolfson, S. Chamerlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *Proceedings of IEEE ICDE*, 1998.

[25] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forcasting system for metacomputing: The network weather service. Technical Report TR-CS97-540, UCSD, May 1997.

[26] Z.L. Zhang, Z. Duan, L. Gao, and Y. T. Hou. Decoupling qos control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *Proceedings of ACM SIGCOMM*, 2000.