Directory Based Composite Routing and Scheduling Policies for Dynamic Multimedia Environments

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

Abstract: *In this paper, we present and evaluate algorithms to address combined path and server selection (CPSS) problems in highly dynamic multimedia environments. Our goal is to ensure effective utilization of network and server resources while tolerating imprecision in system state information. Components within the framework implement the optimized scheduling policies as well as collect/update the network and server parameters using a directory service. We present and analyze multiple policies to solve the combined path and server selection (CPSS) problem. In addition, we study multiple techniques for updating the directory service with system state information. We further evaluate the performance of the CPSS policies under different update mechanisms and study the implications of the CPSS policies on directory service management.*

The evolution of the Internet and differentiated services has expanded the scope of the global information infrastructure and increased connectivity among service providers and clients requesting services for multimedia applications. As this infrastructure scales, service providers will need to replicate data and resources on the network to serve more concurrent clients. Efficient and adaptive resource management mechanisms are required to deal with highly dynamic environments (e.g. those that involve mobile clients and hosts). Quality of Service (QoS) routing techniques have been proposed to improve the network utilization by balancing the load among the individual network links. Server selection policies direct the user to the "best" server while statically treating the network path leading from the client to the server as pre-determined by the routing tables, even though there may exist multiple alternative paths. While the two techniques can independently achieve some degree of load balancing, we argue that in multimedia environments, where the applications are highly sensitive to QoS parameters like bandwidth and delay, high-level provisioning mechanisms are required to address the route selection and server selection problem in a unified way. Such integrated mechanisms can potentially achieve higher system-wide utilization, and therefore allow more concurrent users. Furthermore, in a highly dynamic environment, cost-effective QoS provisioning techniques for server and network resources must be able to tolerate some information imprecision and be able to work effectively with approximate system state information.

In previous work, we have developed a model that addresses the combined routing and scheduling problem based on system residue capacities[FV99]. In this paper, we

(a) develop and evaluate a family of policies for composite routing and scheduling and

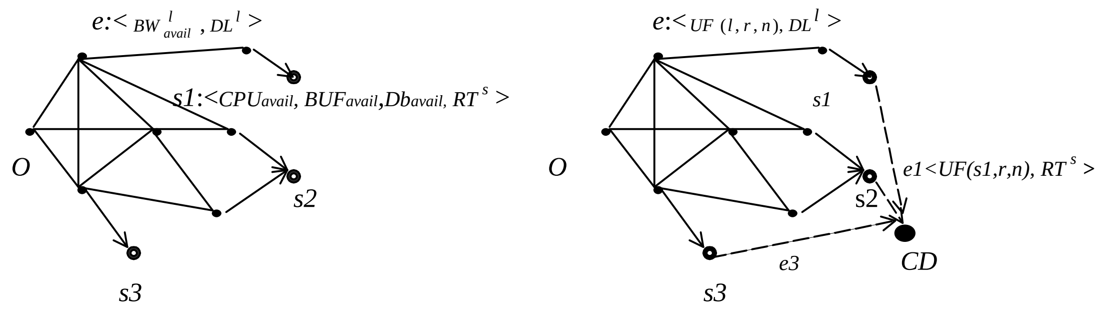
(b) test and understand the performance of the CPSS policies under different levels of information imprecision and varying traffic patterns.

The rest of this paper is organized as follows. Section 2 describes the server and network model and presents the general case of the CPSS algorithm. In Section 3 we develop heuristics to solve the CPSS problem. Section 4 deals with the collection and update of state information using a directory service and discusses several policies for information update. Section 5 contains a performance evaluation of the CPSS policies and update mechanisms. Section 6 describes related work and concludes with future research directions.

2. COMBINED PATH AND SERVER SELECTION (CPSS) ALGORITHM

In this section, we briefly describe the CPSS algorithm. For detailed discussion, please refer to [FV99, FV00, FV00-2]. We model the QoS requirement of the request R from client c as a triple: the path, the server and the end to end quality. R:$< PATH_R, SERV_R, ETOE_R >$.

The capacity of a multimedia server can be specified as three equally important bottleneck parameters: CPU cycles, memory buffers and I/O bandwidth, $SERV_R :< CPU_R,\ BUF_R, DB_R >$. We then model the system as a directed graph *G<N, E>*, where the distributed servers are represented as a node connected with one or more router nodes. A directed edge from router to server is used to represent each connection to a server in graph *G*. For a link *l*, we use a term BW^l_{avail} to note its available bandwidth and term DL^l to note its current delay (the delay includes the propagation delay and the queuing delay at the transmit end). By definition, for a path *p*, we have $BW^p_{avail} = \underset{l \in p}{Min}\left\{BW^l_{avail}\right\}$; $DL^p = \sum_{l \in p} DL^l$. If we use RT^s to denote the



e:< BW^l_{avail}, DL^l >
s1:<CPUavail, BUFavail,Dbavail, RT^s >
O
s2
s3
e:< UF (l, r, n), DL^l >
s1
O
e1<UF(s1,r,n), RT^s >
s2
e3
CD
s3

CPSS Diagram. *Left: Graph G:<N,E> with the client requesting at point O and a set of target servers S: s1,s2,s3. Right: Graph extended from G, adding a point CD and artificial edges e1,e2 and e3*

response time of a server s, then given an assignment *X={p,s}*, with network path p and server s, the end to end delay of assignment X, EED^X, is derived as $EED^X = DL^p + RT^s,\ p,s \in X$.

In order to deal with path and server selection in a unified way, we first define a utilization factor for network links and servers to quantify the residue capacity, and then proceed to define a *Distance* between the client and a server. The utilization factors for a link *l*, given a request *r* and a parameter *n*, is defined as

$$\begin{cases} UF(l,r,n) = \left(\dfrac{1}{BW^l_{avail} - BW_r}\right)^n, \ if\ BW^l_{avail} > BW_r; \\ UF(l,r,n) = \infty, if\ otherwise. \end{cases}$$

The utilization factor for a server *s*, given a request *r* and a parameter *n* is defined as

$$\begin{cases} UF(s,r,n) = \left(Max(\dfrac{1}{CPU^s_{avail} - CPU_r}, \dfrac{1}{MEM^s_{avail} - MEM_r}, \dfrac{1}{DB^s_{avail} - DB_r}) \right)^n \\ \qquad if\ available\ capacities\ greater\ than\ requested \\ UF(s,r,n) = \infty, \qquad Otherwise \end{cases}$$

The parameter *n* in the utilization factor represents the degree to which a lightly loaded resource is favored over a congested resource [LR93]. It serves as a tuning knob and is discussed in detail in [FV00-2].

For an assignment *X={p,s}*, we define the distance of the server *s* to be $Dist(s,r,n) = \sum_{l \in p, p \in X} UF(l,r,n) + UF(s,r,n),\ s \in X$.

The feasibility condition: Given a client request R:< $BW_R, CPU_R, BUF_R, DB_R, DL_R$ >, An assignment *X={p, s}*, is feasible if and only if it satisfies all the following:

$BW^{p*}_{avail} \geq BW_R$,

$CPU^{s*}_{avail} \geq CPU_R, BUF^{s*}_{avail} \geq BUF_R, DB^{s*}_{avail} \geq DB_R$

$EED^{X*} \leq DL_R$

We define a feasible set X_f *as set of all the assignments that meet the feasibility condition.*

Optimality of CPSS: Given a client request R:< $BW_R, CPU_R, BUF_R, DB_R, DL_R$ >, 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

$$Dist(s*,r,n) = Min\{Dist(s,r,n)\}, for\ all\ s\ in\ feasible\ set\ S.$$

We will discuss specific CPSS policies in detail later in Section 3.

2.1 The CPSS ALGORITHM

In this section, we present an algorithm to solve the CPSS problem. Given a network topology *G*, a client request from point *O*, and a target set *S* of replicated servers that contain the information and service requested by the client, we extend the existing topology *G<N,E>* to *G'<N', E'>* by adding one node called Common Destination, *CD*, to graph G, and one artificial edge per server *s* in target set *S*, denoted E_S, from the server *s* to the common destination *CD*, [see CPSS Diagram above]. The weight of *e* is defined as *W(e):<UF,DL>*, two additive parameters representing the load level and the delay respectively. Specifically, the weight function *W* in *G'* is derived as follows:

(a) for edge *e(u,v)* in *E*, define $W(e)= <UF(e,r,n), DL^{e} >$;

(b) for edge *e'(s,CD)* not in *E*, but in *E'*, define $W(e')=<UF(s,r,n), RT^{s} >$.

```
The CPSS algorithm (G'<N', E'>, R, O, n, )
(1)  /* initialization */
(2)  For each edge e(u,v) in G'
(3)     If e is in E,
(4)         if UF(e,R,,n) = INFINITY
(5)             delete edge e from G'
(6)         Else
(7)             W(e).dist = UF(e,R,n); W(e).delay = $DL^e$
(8)     Else /* e is an artificial arc, e=(s,CD). */
(9)         if UF(s,R,n) = INFINITY
(10)            delete edge e from G'
(11)        Else
(12)            W(e).dist = UF(s,R,n); W(e).delay= $RT^s$
(13) /* run the Restricted Shortest Path algorithm for*/
     /*  the feasible path set $X_f$ , */
     /* $X_f$ ={P| P{(O, v1), (v1,v2), .. , (s, CD)}*/
(14) $X_f$ = RSP(G'<N',E'>, W, O, CD, DLr)
(15) calculate optimal assignment X*={ P*\(s*, CD), s*},
     based on CPSS policy
(16) return X*
```

To simplify the graph, we remove from G' those edges in which the available capacity is less than that requested. We calculate a set of paths from start point *O* to *CD* to form a feasible assignment set X_f, subject to the constraint of the end to end delay. In the appendix, we prove the feasibility conditions for an assignment derived from a path *P* thus calculated.

One of the heuristics to calculate a feasible set is to use a dynamic programming technique, assuming the delay value is integer [H92,LO98,CN98]. After the algorithm terminates, a feasible path set can then be derived from the dynamic table. The complexity of this Restricted Shortest Path (RSP) algorithm is *O(D|N|)*. A detailed discussion on our node queue based RSP implementation is presented in [ZV00-2].

3. POLICIES TO CHOOSE AN OPTIMAL ASSIGNMENT

The general flow of the CPSS algorithm is as follows. A request containing QoS parameters is initiated at a source node, a directory service collects the system information and makes the path and server assignments for the client request. Given the assignment, the client node proceeds to set up a connection along the assigned network path to the server. The routes and the servers check their residue capacity and either admit the connection by reserving resources or reject the request. When the connection terminates the client sends the termination requests, and the resources are reclaimed along the connection.

We propose deterministic and non-deterministic CPSS policies that choose an optimal assignment from X_f. Our overall objective is to improve the overall system utilization and number of concurrent users, therefore we focus on policies that minimize the usage of system resource while balancing the load across the links and servers.

Deterministic CPSS Policies:

Shortest Hop: Choose an assignment from the feasible set X_f s.t. the number of hops from source to destination is minimal: *X*:<p*,s*> s.t. Hop(X*) = Min{Hop(X)| For all X in feasible set* X_f *}*. This variation of the traditional shortest path policy provides a shortest widest solution.

Best UF: Choose an assignment from the feasible set such that the utilization factor (UF) is minimal: *X*:<p*,s*> s.t. Dist(X*) = UF(p*)+UF(s*)=Min(Dist(X) | for all X in feasible set* X_f .

The Best UF policy is a variation of online algorithms that maximizes the overall utilization of resources in the system without knowledge of future requests

Non-deterministic CPSS Policies: Non-deterministic policies attempt to achieve a better degree of load balance among the links and servers. The probablistic policies presented here allow a differential treatment of network and server resources instead of treating the server and network resources in a unified manner. This helps ensure that the more constrained resource can be treated preferentially, yielding better adaptation under dynamic conditions.

Random path selection: select *X*:<p*, s*>* randomly from the feasible set X_f. By randomly picking a choice from the feasible set, this policy tries to avoid oscillations that are often characteristic of more static policies.

Weighted Differerential Probabilistic Policy (Prob1): From the feasible set X_f, we calculate a selection probability for each *Xi* based on its residue capacity relative to other assignments. The probability is defined as $Sel\_Prob(X_i) = R_1 \cdot \frac{UF^{-1}(s_i)}{\sum_{X_k \in X} UF^{-1}(s_k)} + R_2 \cdot \frac{UF^{-1}(p_i)}{\sum_{X_k \in X} UF^{-1}(p_k)}$ *R1* and *R2* decide how the server and path resources should be emphasized in calculating the selection probability of the assignment. For instance, in order to avoid the situation where we have a good server with a bad path or a good path with a bad server, we can set *R1 = R2 = 0.5*.

Load Sensitive Differential Probabilistic Policy (Prob2): The Prob2 algorithm works as follows.

```
Prob2(X)
1.  From the feasible assignment set X, X={ $X_1 :< p_1, s_1 >,\dots, X_n :< p_n, s_n >$}, calculate distinct server set S,
    S={ $s_1, s_2, \dots, s_k$ }, $s_i \neq s_j, \forall i, j \in [1,k], i \neq j$ .
2.  From the feasible assignment set X, calculate average UF value of network and server[1], $UF_{AVG}^{NW}$ and $UF_{AVG}^{SVR}$
3.   /*Select s* from server set S according to $UF_{AVG}^{NW}$ and $UF_{AVG}^{SVR}$ :*/
4.  If $UF_{AVG}^{NW} / UF_{AVG}^{SVR} > r$, r is a threshold parameter,
        /* Which means the network is much more
           loaded than the servers */
5.      Weight all servers in S equally as $\frac{1}{k}$.
6.      Select a server s* from server set S.
7.  else
        /* Which means the servers are much more
           loaded than the network */
8.      Weight server $s_i$ in S as: $\frac{UF^{-1}(s_i)}{\sum_{j \le k} UF^{-1}(s_j)}$
9.      Select a server s* from S with max weight
    /* Select a best path leading to the selected server */
10. Determine set X' $\subseteq$ X, s.t. X'={ $X_i :< p_i, s_i > | s_i = s^*, i=1,2,\dots n.$ }
11. Weight $X_i$ in X' as $\frac{UF^{-1}(p_i)}{\sum_{j \le k'} UF^{-1}(p_j)}$
12. Select an assignment X* from X' with max weight
13. Return the selected assignment X*
```

From the feasible set X_f, the first phase calculates the average UF values of path and server components to decide which of the two resources constitute the bottleneck for the client's request. Based on the bottleneck factor determined in the first phase, the second phase executes as follows. (a) If the server resource is determined to be the bottleneck, we emphasize on balancing the load between the servers. To do this, we first eliminate multiple paths in *X* to the same server, and build a reduced server set S that contains distinct feasible servers. Next, we probabilistically select a server, s*, from server set *S*, according to each server's relative UF value in *S*; Finally from all the paths leading to that server *s**, we probabilistically

[1] Here, in order to be comparable to server, $UF(p_i) = Max\{UF(l) | l \in p_i\}$.

select a path, *p**, according to relative UF value among all such paths. (b) If the network is the restricting element, we emphasize on balancing the load between the alternative network links. Experiments reveal that many paths in the network share large amounts of network links, so the most effective way of balancing the load between network links is to distribute the request to different servers. Therefore we select a server *s** first from the server set S using a uniform probability distribution, and then probabilistically select a path *p** leading to that *s**. Our goal is to randomize the usage of the network to avoid hotspots and maximize load balance among the various network paths.

4. DIRECTORY ENABLED PARAMETER COLLECTION AND UPDATE

The above CPSS policies are based on the knowledge of network topology, replica map, and the load information of network links and distributed servers. In our model, this information is maintained in a directory service accessed by the CPSS module. The network topology can be maintained by routing information exchange, and a replica map can be obtained from a distributed domain name service either in an on demand mode or a caching mode ([FJPPCGJ99], [KSS98]). To deal with dynamic changes in network and server conditions, a parameter collection process must be integrated and evaluated carefully together with the overall CPSS performance.

We assume that the directory is updated with current system state information by traffic probes that are distributed within the network. One example of probing among typical ISP networks can be found at [GTE99], where probe modules are placed at each data center in the backbone network. In dynamic multimedia environments, delay and bandwidth information can be highly dynamic and often follows a heavy tailed distribution [CC97]. However, it can be observed, that for a given period of time, the probability of the delay or available bandwidth taking certain values is high, and the trend of these "mean" values doesn't change dramatically. Hence we can use a predicted range of most probable values to approximate the state information with tolerable accuracy [FV00-1].

A request may get rejected at the directory service if it can be determined a priori that there are insufficient resources to satisfy the request. Alternately, a request may be admitted by the directory service, but can encounter poor QoS along the assigned network path or at the server. This is because the directory service contains only approximate state information about the network and system resources. An assignment for a request is not successful unless the assigned nodes indeed have sufficient resources to meet the QoS requirement of the request.

Information Update Policies: It is evident that higher accuracy of information in the directory service leads to better server/path selection performance. The information collection and update process can however entail significant overhead. Our objective is to improve the request success ratio while controlling the overhead cost of the information updates. We used two update techniques in our performance evaluation to explore the dynamics between the update behavior and CPSS policies.

(a) System Snapshot**:** Here, information about the residue capacity of network nodes and server nodes is based on an absolute value obtained from a periodic snapshot. For each update period, probing is initiated to gather the current information of router nodes and server nodes; the directory is updated with the collected values.

(b) Range Based Collection: The residue capacity information is collected using a lower bound, L, and a upper bound, H, and the actual value is assumed to be uniformly distributed in this range with the expected value *(H-L)/2*. Range based information collection can have many variations [AGKT98, FV99, FV00]; our performance evaluation focuses on one of them - i.e. the fixed interval based policy. In the Fixed Interval Based policy, instead of using a range *<L,H>*, 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 update period, probing is initiated to get the current information from router and server nodes. If the current value is out of the current range, the directory is updated with another interval based range, otherwise no update is sent. We further explore 3 variations of the fixed interval update policy. The value of *UF* or *DL* of an edge in *G'* can be calculated using one of the following policies:

- Pessimistic (PESS) : uses a lower bound of a uniform distribution corresponding to the current interval.
- Optimistic (OPT) : uses an expected value of a uniform distribution corresponding to the current interval, i.e. *(H-L)/2*
- Optimistic Favor Stable (OPT2) : uses an expected value multiplied by a "*Degree of Stability*" calculated as $\frac{H-L}{Capacity}$.

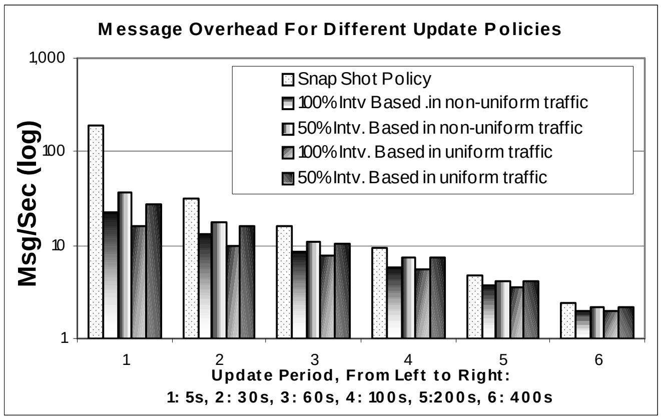


Fig 1: Message Per Seconds For Update Policies

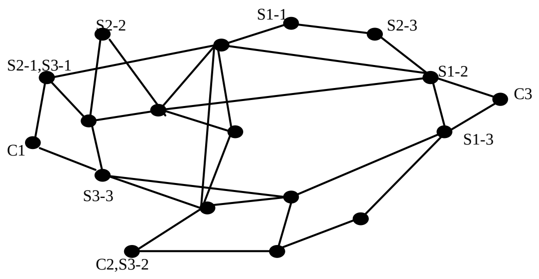


Fig 2. Topology - For *non-uniform traffic*, the three hot pairs are: 1) C1:{S1-1,S1-2,S1-3}, 2) C2: {S2-1,S2-2,S2-3}, 3) C3: {S3-1,S3-2,S3-3} Assume that the clients and servers are directly behind the router nodes being addressed.

Interesting tradeoffs exist with the above information update mechanisms. Firstly, there is a tradeoff between message overhead cost and accuracy of state information in the directory. The message overhead cost will generally increase with a shorter update period for all update policies, but for the same update period, range based update policies will generate less message overhead than the simple system snapshot. For instance, in interval based policies, the larger the interval, fewer update messages are needed, thereby reducing the overhead cost. Figure.1 shows the message overhead cost of three information update methods: system snapshot, 50% interval(smaller interval) and 100% interval(larger interval).

5. PERFORMANCE EVALUATION

The objective of the simulation is to study in detail the performance of policy based CPSS algorithms under different workload patterns and load situations and to correlate the performance results to the information collection costs. This will help us understand the dynamics and the tradeoffs underlying our CPSS algorithm in a distributed environment. Our performance study consists of 2 steps:

(a) Evaluation of policies for optimal path and server selection in the CPSS process: We focus on different policies for *optimal* path and server assignment among multiple feasible assignments which all satisfy our base CPSS requirements. In the previous discussion, the base CPSS algorithm finds all feasible assignments that satisfy the QoS requirements of the client, which includes network and server parameters as well as end to end delay requirements.
(b) Evaluation of update techniques for CPSS decision making: Our simulation will further focus on situations where system state information is updated very infrequently, i.e., the information collection period is relatively long, to study the cost-effectiveness of different polices in different traffic patterns and load situations.

5.1 The Simulation Model and Environment

We use a simulator call "QSim", developed at UC Irvine. QSim is a message driven multi-threaded simulator intended to study the dynamics of QoS sensitive network environments. Since we are interested in the overall behavior of traffic flows, in particular, flow setup and termination, the simulator doesn't deal with details of packet transmission in the network. QSim has the following components: traffic source, routers, servers and directory service nodes. The reservation in QSim is rather simple, the actual implementation could use standard RSVP to make reservation in a distributed global network. Because QSim doesn't go into detail of packet passing , we model the delay characteristic of a link and a server as exponentially correlated to their residue capacities (See [FV00-2] For Deriving Delay Values).

Topology and System Configuration: In the simulation, we use the following typical ISP network topology with 18 nodes and 30 links. 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.
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[2], and the request holding time[3] is exponentially distributed with a pre-specified average value. 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 model traffic, we generate two types of traffic patterns: non-uniform traffic and uniform traffic. 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 equal. Specifically we set the foreground arrival rate to be 10 seconds, and the background rate to 50 seconds. In order to simulate medium sized multimedia sessions, we set the average request hold time to be 10 min.

5.2 Path and Server Scheduling

Given a set of feasible assignments, we start by studying the following policies for choosing the optimal assignment X*- Best UF, Shortest Hop, Random, Prob1 and Prob2.

5.2.1 The Best UF Policy

The performances of Best UF policy under different information update frequencies is depicted in Fig 3.a. Given near current system state information, the Best UF policy is a variation of online algorithms that optimize the current assignment (i.e. maximize overall resource utilization) without knowledge of future requests. We examine the performance of Best UF in near current system state information, i.e., with very short update periods. In Fig [3].c we show that requests are rejected by the directory service (DS) when it tries to find an assignment for the requests and encounters limitations in the network and server capacity. Most requests admitted by the DS are eventually committed by network and servers, so the network and server rejection rate is very low. . This results in a higher utilization of the network and server resources which can be observed from Fig [3].b. When the state information in the DS is very inaccurate, i.e., long update period, the directory uses the outdated load information to assign the *same* "best" paths and servers to the clients resulting in quickly congesting these nodes and links. This causes the network or server to reject the assignment chosen by the Directory Service. Fig[3].b shows that request rejections cause the overall system utilization to fall; the drop in utilization is recorded in the DS during an update; the lightly loaded paths and servers are quickly filled by the DS resulting in more rejections. This is reflected in the rejection pattern graph with update period 1000s, Fig [3].d. In this case, the outdated information makes DS keep repeating the same assignment, resulting a high rejection ratio from the network and server resources. In our specific experiment settings, server rejection appears dominating the overall performance, because the servers resources are fewer than network resources and are quicker to be saturated.

5.2.2 The Shortest Hop Policy

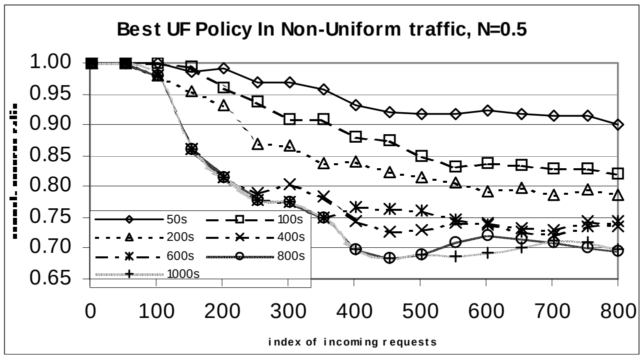
The Shortest Hop Policy is interesting because it provides a shortest widest solution. From the feasible assignment set *X* calculated by base CPSS algorithm, the policy chooses the nearest server from the client in terms of number of hops. If multiple such servers exist, the policy chooses the least loaded one, i.e. the one with least UF value. In general, we can see that the performance of the Shortest Hop Policy is worse than Best UF under the conditions of the experiment (large update period). This is because Best UF subsumes shorter, wider paths Fig [5]. The longer the path, the bigger the resulting UF value, since for a path *p*, $UF(p) = \sum UF(l), l \in p$. So this makes the shortest widest path a strong candidate to be selected in Best UF policy. The shortest hop policy only considers the length of the path and thus tries to optimize the usage of network

[2] Since our study focuses on generalized traffic patterns, we do not model requests for the specific MM objects. We intend to explore more sophisticated traffic generation models (e.g. Zipf like) that account for popularity based generation of specific requests when we consider object placement techniques and scheduling specific requests for objects.

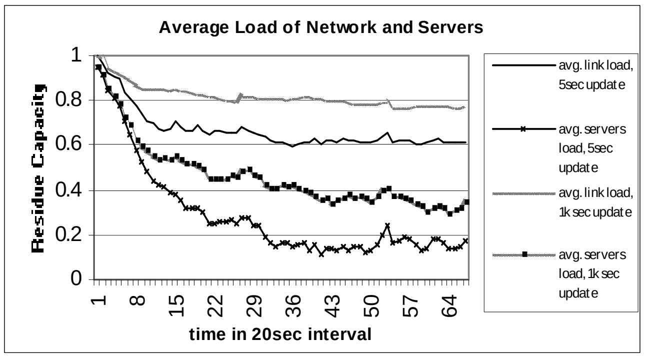
[3] The request holding time is the time for which the requested network and server resources such as link bandwidth, CPU, buffer, disk bandwidth etc. are reserved.

resource by using the least number of network links without considering current load situations on the links or servers. When state information is not current, as is the case with a large timer, the shortest hop policy tends to initiate congestion earlier because it potentially overloads some paths and servers that are already very congested. Hence, a large number of request rejections in the Shortest Hop policy result from path rejections in the network Fig[4]. With a large update timer, the best UF policy will always route the request along less loaded path to less loaded servers, and hence congestion is expected to arrive slower. This behavior is confirmed by the reject pattern in the 20 second detail graph of 1000s update period. Initial path rejections are caused in the Shortest Hop policy while the Best UF policy initiates the network path rejections later in time. After the update at 1000s, the Shortest Hop policy again exhibits a large number of path rejects; in addition, there are server rejects due to server saturation.

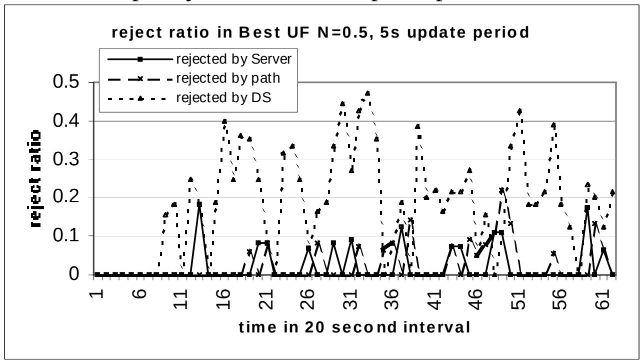
Fig 3: Best UF Policy In Non-uniform Traffic Environment



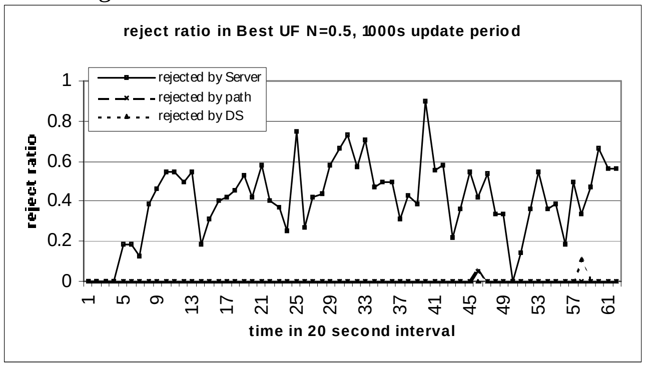
A. Best UF policy with different update periods



B. average load of network and servers

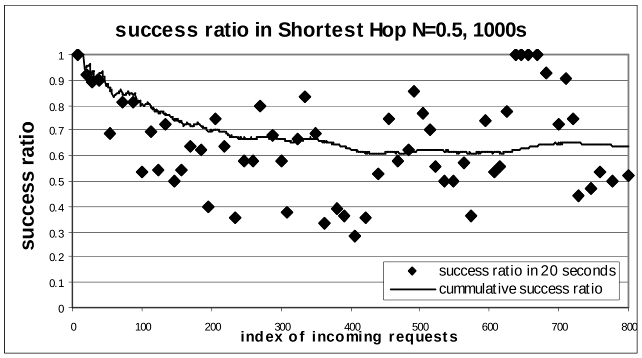


C: reject pattern when update period equals 5 sec.

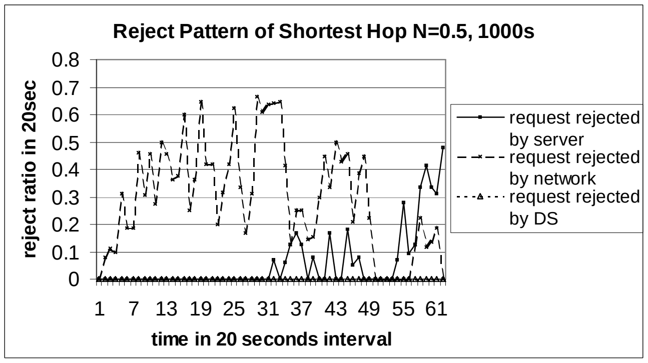


D: reject pattern when update period is 1000sec.

Fig 4: Shortest Hop Policy In Non-uniform Traffic

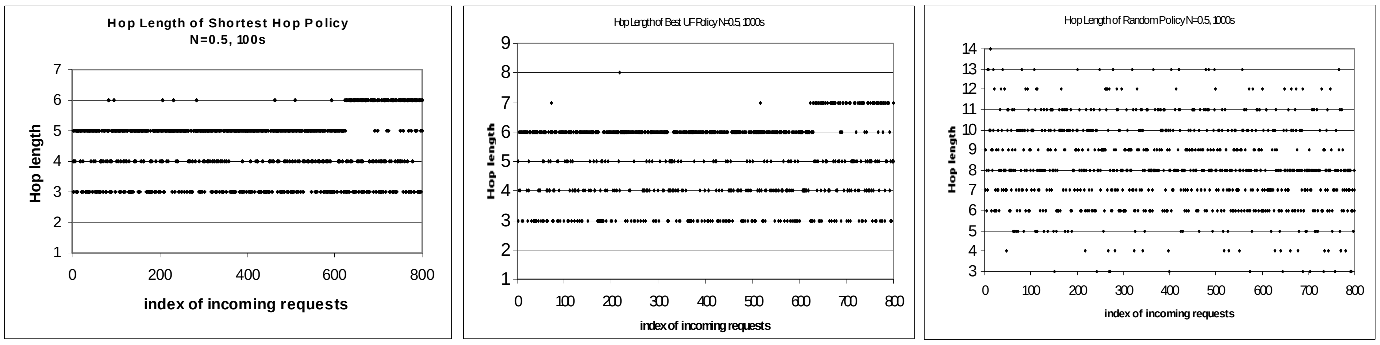


A: Performance with 1000s update period



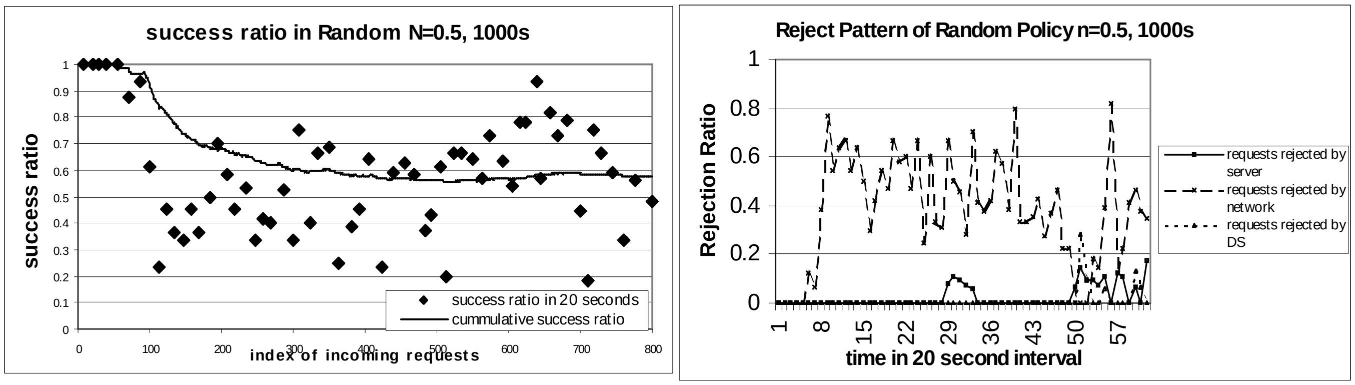
B: Reject Pattern with 1000s update period

Fig 5: Hop Length of CPSS Policies



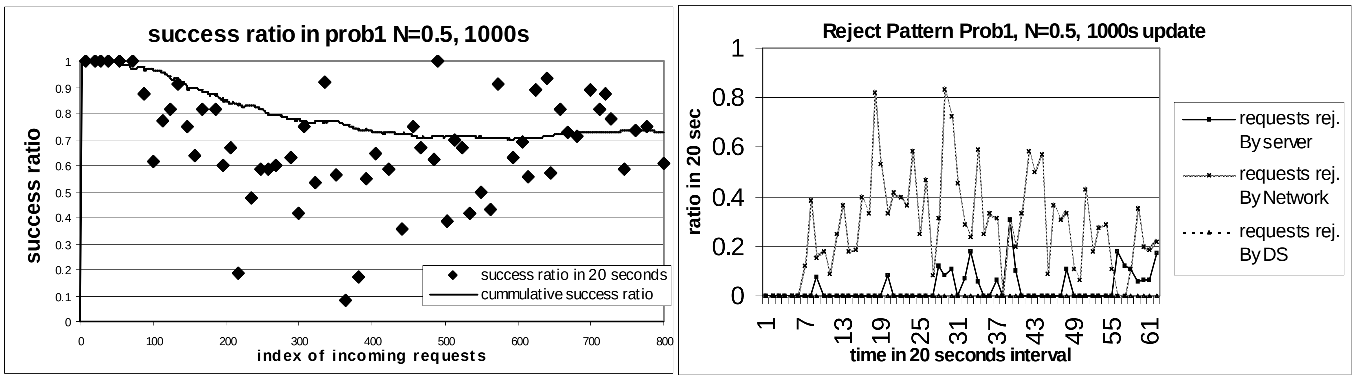
A: Best UF Policy B: Shortest Hop Policy C: Random Policy

Fig 6: Random Policy In Non-Uniform Traffic Environment



A: Performance with 1000s update period B: Reject Pattern with 1000s update period

Fig 7: Prob1 Policy In Non-uniform Traffic Environment



A: Performance with 1000s update period B: Reject pattern with 1000s update period

5.2.3 The Random Policy

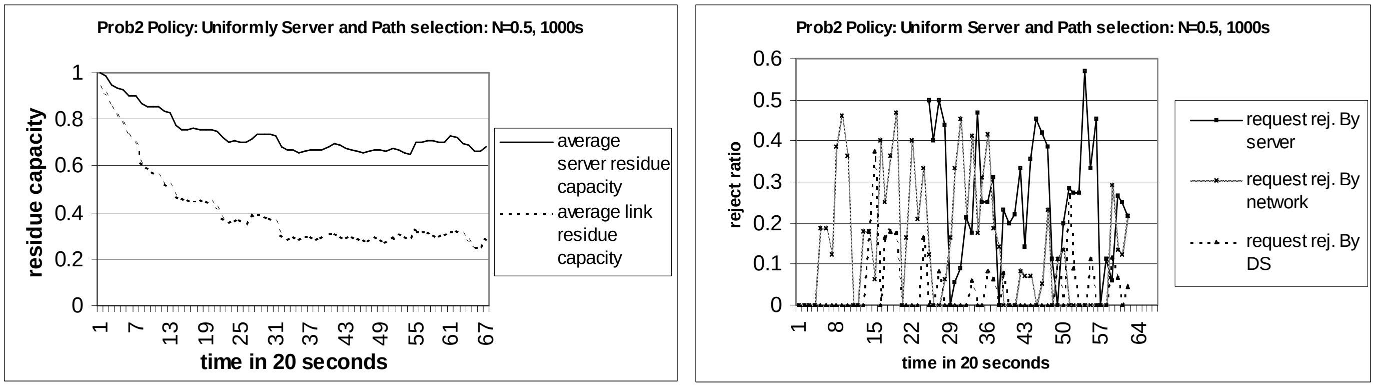
The random policy tries to avoid the oscillation of static polices and balance the load of the system by randomly picking one choice from the feasible assignment set X calculated by CPSS. The random policy does balance the load between the servers, but because it doesn't use the load information of the assignments, it often results in longer network paths and hence quickly gets rejected by the network nodes. Fig. 5 shows the path lengths of Best UF, Shortest Hop, and Random policies. The 20 second (instantaneous) detailed graph also confirms that the path rejection dominates the overall request reject ratio. This is because the random policy does not differentiate between all the feasible assignments in the set X, repeatedly picking some feasible but marginal assignment, leading to congestion. In summary, the random policy performs consistently worse than the Best UF and Shortest Hop policies.

We next evaluate the probabilistic policies under different traffic patterns and load levels. Our results indicate that the probabilistic policies exhibit better success ratios than deterministic policies, i.e Best UF. This is because the probabilistic policies tend to distribute the load among feasible assignments under a given system state to avoid oscillation. With a large update timer, the deterministic policies tend to repeatedly pick the least loaded resources that are quickly exhausted.

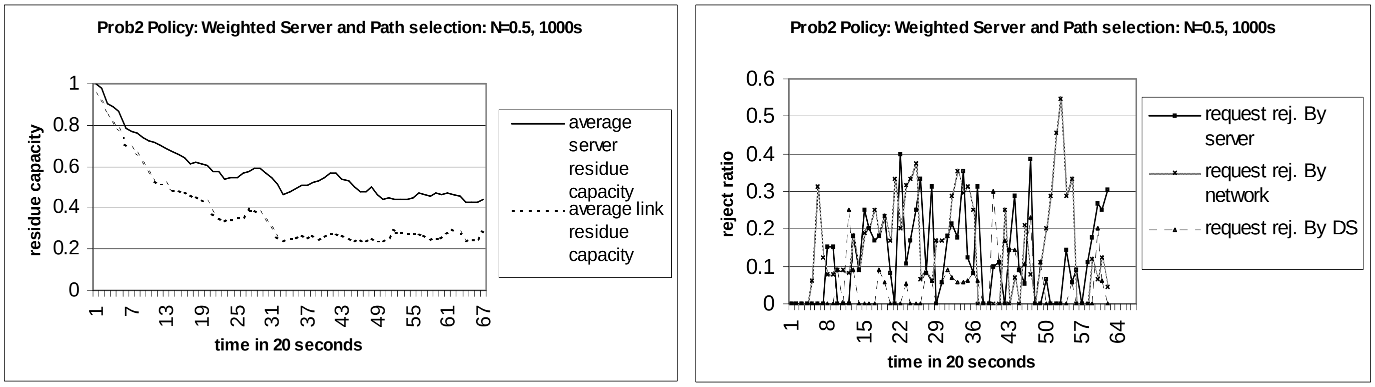
5.2.4 The Prob1 Policy

Among probabilistic policies, the Prob2 policy further improves on the performance of Prob1. Prob2 differentiates between the network and server resources and ensures that the more bottlenecked resource is selected with a lower probability. With the Prob1 policy, an assignment with a lightly loaded server has a high probability of being selected even if the network path leading to that server is highly congested. This side-effect is eliminated in the second phase of the Prob2 policy where all paths leading to that server are weighted according to their residue capacity.

Fig 8: Prob2 In Non-uniform Traffic Environment



A and B: Servers weighted equally, update period is 1000s. A: (left), average system load, B: (right), request reject pattern



C and D: Servers are weighted according to their UF value, update period is 1000s. C: (left) average system load, D: (right) request reject pattern

5.2.5 The Prob2 Policy

We further experimented with variations of the Prob2 policy. The Prob2 policy uses either a weighted or uniform (i.e. weighting all servers equally) probabilistic method when choosing an optimal assignment from the feasible server set. Uniform and weighted probabilistic policies produce varying effects under a large update timer. In the presence of congestion, the weighted policies tend to provide better load-balance in the short term; however, they can cause further congestion over a period of time. The uniform policy will not alter the existing load conditions dramatically resulting in more rejects in the short term. The first set corresponds to the uniform policy and the second set corresponds to the weighted policy. In the first graph, the relative loads of the server and the network do not change much over the entire duration, with the servers being consistently more loaded than the network. The instantaneous 20 second reject pattern indicates that the server and network rejects alternate and are comparable in number. In the second graph, the weighted policy tends to pull the two resource utilization levels closer resulting in better balance.

5.3 Information Update

To further evaluate the performance and efficiency of our previous CPSS policies, we compare them with a static “nearest” server algorithm, which implements server selection by counting the number of hops from the client to all candidate servers and selecting the nearest one. The results in Fig[10] shows that in non-uniform traffic environments, the CPSS policies are 20%-30% better than the static algorithm, while in a uniform traffic environment, CPSS policies outperform the static algorithm by 15% on average. The performance gain of CPSS algorithm is obtained at the cost of increased message overhead caused by periodic system state updates, Fig [1]. The update period dominates the information accuracy in the directory and thus influences the performance of the CPSS algorithm. In the following we focus on the influence of the information update overhead on CPSS policies.

Snapshot Based Information Update: The snapshot based information update can be regarded as a special case of interval based update by setting the range to 1. Our simulation shows that the interval-based policies do not have a significant influence when used together with deterministic CPSS policies. Thus, we focus our discussion of snapshot based information update with deterministic CPSS policies. In general, we observe that a shorter update period results in better performance than a larger update period. This holds only when the information update period is smaller than the average

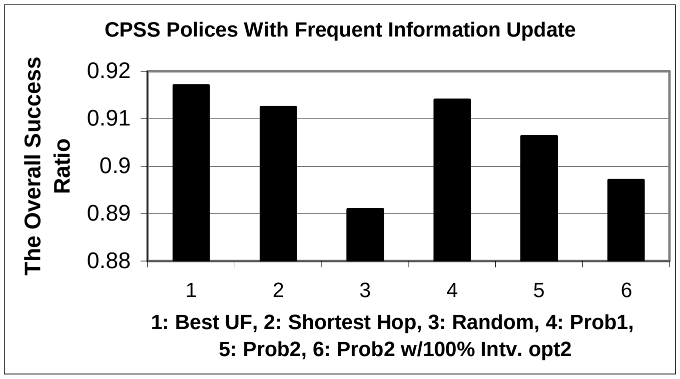


Fig 9: Policies w/ frequent info updates (period = 5 sec.)

connection holding time. For instance, in our experiments the average connection holding time is set to 10min (600s). Update period values larger than 600s show negligible effects on the performance of CPSS policies.

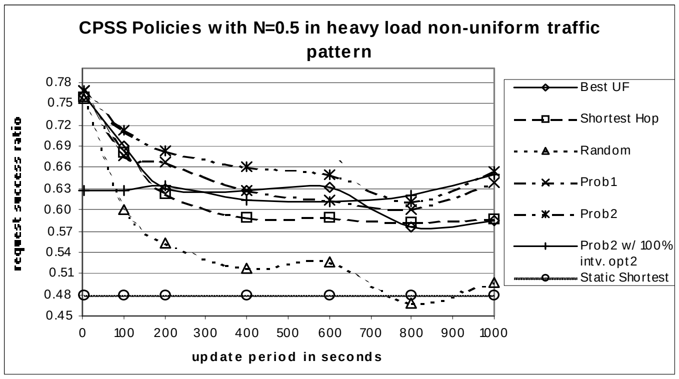
If the system state information is updated very frequently, our results show that Best UF policy is superior Fig[9]. This is because the Best UF policy tries to find an optimal tradeoff between the shortest and widest paths (See 5.2.2). In situations where information update is infrequent, the deterministic policies, Best UF and Shortest Hop, inevitably go into oscillation, limiting the overall system throughput. The non-deterministic policies, Random, Prob1, and Prob2, distribute load between multiple feasible servers and network paths, thus prevent the oscillation and improve the overall resource utilization. It should be noted that there are significant performance differences among these three policies; Prob2 consistently performs best and the Random Policy always performs the worst.

Interval Based Information Update: In order to further save the cost of information update, we introduced interval based information update policies; the reduction in message overhead cost can be seen from Fig[1]. Our simulation shows that the interval-based policies do not have a significant influence when used together with deterministic CPSS policies. Hence, we restrict our description to the study of the interval-based update policies when used together with probabilistic policies, i.e. Prob2. Fig[10.B,D,F] presented in shows the performances of interval based update policies in various traffic environments.

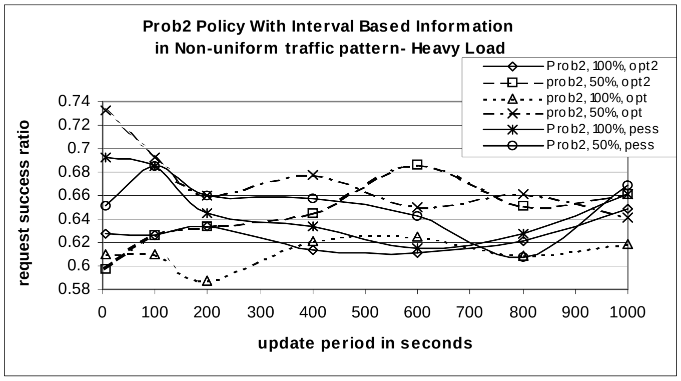
With frequent updates, a smaller range for the interval, i.e., 50% of the maximum requested size, brings better CPSS performance than a bigger interval, i.e., 100% of the maximum request size.
For large update period, the interval based policies are attractive because we believe it is natural to represent a residual value using a range (instead of an instance value over a long period of time). For a larger update period, a bigger range brings better CPSS performance. This is more obvious in uniform traffic pattern as shown in Fig [10.B]. The reason is that when the update period is short, representing a residue value using a big range introduces information inaccuracy, a shorter range is better. However, when the update period is very long, a residue value represented using a small range often gets out-dated sooner than a larger one, resulting in a more inaccurate system state information.

An analysis of the variations of the interval based update shows the following results. In a lightly loaded non-uniform traffic environment, the OPT2 variation performs better than other variations, i.e. OPT and PESS. In a heavily loaded non-uniform traffic environment, the variations of interval based update do not exhibit an obvious influence on the performance of Prob2. However, in general, the performance of Prob2 is better than other CPSS policies with either interval based or snapshot information update.

Fig 10: Overall Performance of CPSS with information update
A-B: In heavily loaded non-uniform traffic environment

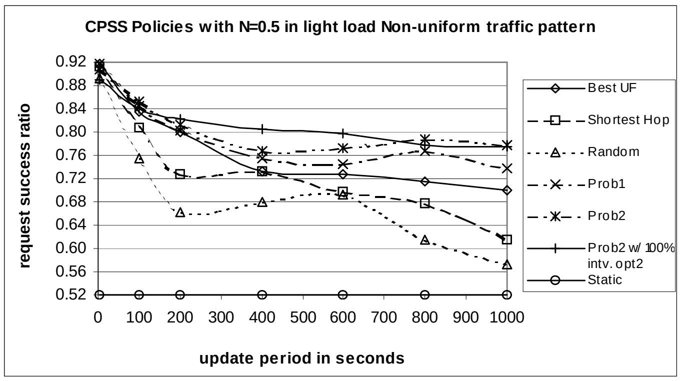


A: Overall performance results.

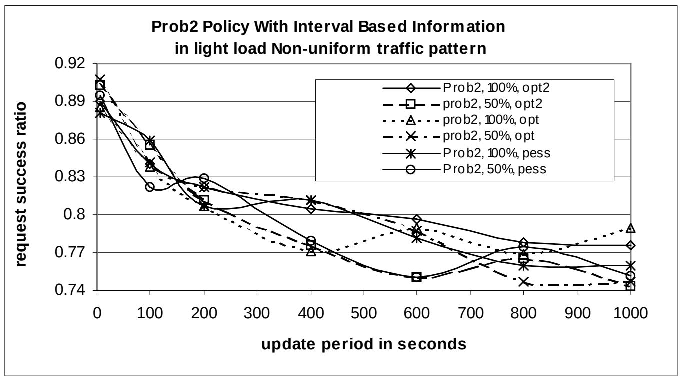


B: interval based update policies with Prob2

C-D: In lightly loaded non-uniform traffic environment

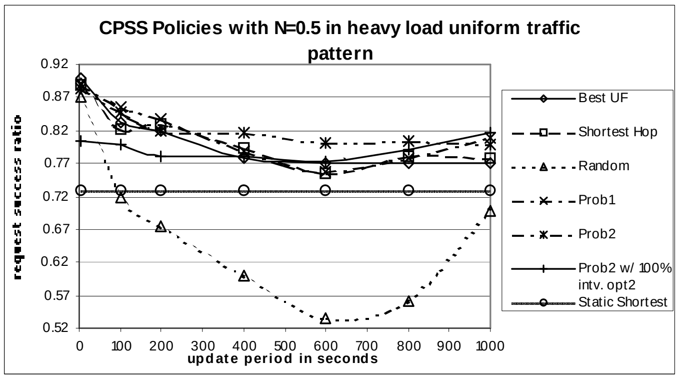


C: Overall performance results.

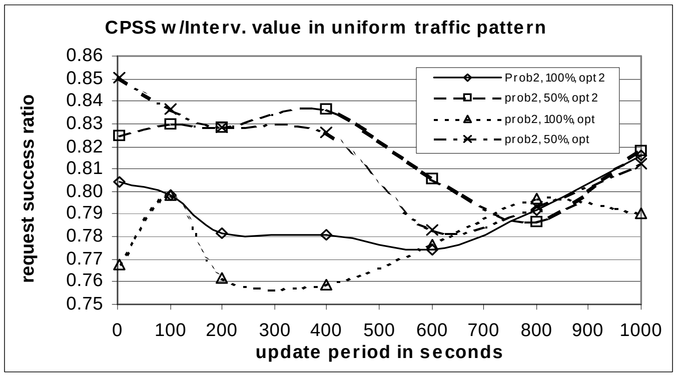


D: interval based update policies with Prob2

E-F: In uniform traffic pattern



E: Overall performance results.



F: interval based update policies with Prob2

5.4 Performance Summary: Our evaluation indicates that CPSS based policies perform about 20-30% better on average than non-CPSS policies where server selection is performed using a static, nearest-server policy. The performance of the CPSS policies explored is sensitive to the frequency of update of system state information. With a short update timer, state information is up-to-date, Best UF and Shortest Hop policies result in near-optimal assignments because of CPSS calculation. However, the Best UF Policy performs better than the shortest hop under a large update timer. In general, with a large update period, deterministic policies suffer from oscillation that introduces "hot spots" into the environment causing congestion. The random policy performs a lot worse than other policies consistently. The probabilistic policies (Prob1 and Prob2) perform consistently better with a large update timer. In general, deterministic policies like Best UF normalize server and link loads into one unified utilization measure – thus making it more difficult to handle situations where network and server conditions vary dramatically. Such dramatic variations are better accomodated by probablistic policies like the Prob2 (load sensitive differential policies) that treat network and server loads independently in the decision-making process.

Snapshot based update mechanisms introduce larger overhead while no significant performance gains can be observed. Our study indicates that range based update mechanisms are more cost-effective in QoS-sensitive dynamic environments. Furthermore, our experiments reveal that the probabilistic Prob2 policy performs significantly better under both snapshot and interval based update techniques than other CPSS policies. The variations of the interval based update mechanisms (PESS, OPT and OPT2) exhibit varying performance with OPT2 performing better on average with a large update timer.

6. RELATED WORK AND FUTURE RESEARCH DIRECTIONS

We describe related work in the areas of replicated service selection and QoS routing. Static nearest server algorithms [GS95] 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 [FPLZ98, FJPZGJ99, KSS98, MDZ99], 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[CC97]; 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 [FBZA97] 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 [VR97,HCM98]. [VR97] 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.

QoS-based routing techniques have been explored in depth[CN99, ZT98,CRS97,BS98,BEZ93,MSZ96]. QoS-based extensions of the traditional Bellman-Ford/Dijkstra algorithm [BG92] incorporate an available bandwidth parameter and use widest-shortest path [GOW98,CN98] calculations. Experimental efforts studying the combined performance and overhead cost of different information collecting and routing policies [AGKT98] 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 [LO98] 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[H92].

Our work differs from traditional QoS routing in the several aspects. We presented and evaluated a directory-enabled approach to provide combined path and server selection in a dynamic multimedia environment. We exploit architectural considerations of network links and servers to treat them differentially in the overall path/server selection process. We analyze request rejection patterns at various points in the directory-enabled architecture (i.e. directory service, network links and servers) under dynamic conditions. This allows us to understand the ability of the CPSS policies to accurately reflect these dynamic conditions under large update periods. Specifically, we have studied parameter update mechanisms such as range based update policies in the directory service for effective performance of the CPSS strategies.

We are working on applying our CPSS model to a differentiated service IP network where a Bandwidth Broker performs high level measurement based admission control for stateless routers within the autonomous region. We are also developing a model to measure the cost of “re-routing” and “re-scheduling” to deal with handoffs or location dependent error bursts in a mobile environment. We are also exploring object placement policies using dynamic replication and migration in a wide area scenario. Further work on QoS-based provisioning in wide-area distributed environments is being studied in the context of a QoS-enabled middleware framework, CompOSE|Q [V99] currently being developed at UC Irvine. We believe that the simultaneous execution of multiple resource management mechanisms is key to effective system utilization. The work presented in this paper is a step in trying to integrate such policies into a uniform framework

REFERENCES

[AGKT98] G.Apostolopoulos, R.Guerin, S.Kamat, S.K.Tripathi, Quality of Service Routing: A Performance Perspective. In Proc. of ACM Sigcom’98

[BEZ93] L.Breslau, D.Estrin, and L.Zhang. A simulation study of adaptive source routing in Integrated service network. USC CSD Technical Report Sep. 1993

[BG92] D. Bertsekas and R. Gallager. Data Networks, Prentice Hall, Englewood Cliffs, N.J., 1992

[BS98] Lee Breslau and Scott Shenker, Best-Effort versus Reservations: A Simple Comparative Analysis, Sigcomm ’98

[CC97] R.L.Carter and M.E.Crovella, "Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks", in Proceedings of Infocom '97.

[CN98] S.Chen, K.Nahrstedt, On Finding Multi-Constrained Paths, in Proceedings of IEEE International Conference on Communications (ICC 98), June 1998, Atlanta.

[CN99] S.Chen and K.Nahrstedt, Distributed Quality-of-Service Routing in Ad-Hoc Networks, IEEE Journal on Special Areas in Communications, Special Issue on Ad-Hoc Networks, 1999.

[CRS97] I.Cidon, R.Rom, and Y.Shavitt. Multi-path routing combined with resource reservation. Infocom’97

[FBZA97] Zongming Fei, Samrat Bhattacharjee, Ellen W.Zegura, Mostafa H.Ammar, A Novel Server Selection Technique for Improving the Response Time of a Replicated Service, Infocom ‘97

[FJPZGJ99] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, Y. Jin. An Architecture for a Global Internet Host Distance Estimation Service". INFOCOM '99, March 1999.

[FPLZ98] A. Fei, G. Pei, R. Liu, L. Zhang. "Measurements on Delay and Hop-Count of the Internet". GLOBECOMM 98.

[FV99] Z.Fu, N.Venkatasubramania. Combined Path and Server Selection In Dynamic Multimedia Environments, ACM MM ’99

[FV00-1] Z.Fu, N.Venkatasubramania. Directory Based Information Collection for QoS Provisioning in Dynamic Multimedia Environments. Submitted for publication.

[FV00-2] An Evaluation of Composite Routing and Scheduling Policies for Dynamic Multimedia Environments, extended version, UC Irvine Technical Reports

[GOW98] Roch A. Guerin, Ariel Orda, and Douglas Williams, QoS Routing Mechanisms and OSPF Extensions,Globcom’98

[GTE99] GTE Internet Access Services: Managed Hosting with Traffic Distributor, http://www.bbn.com/services/hosting/traffic

[GS95] J.D.Guyton and M.F.Schwartz, “Locating nearby copies of replicated internet servers,” in ACM SIGCOMM, 1995

[H92] R.Hassin. Approximation schemes for the restricted shortest path problem. Math. of Operations Research, 1992

[KSS98] S.Keshav, R.Sharma, and R.Siamwalla, “Project octopus: Network Topology discovery”, http://www.cs.cornell.edu/cnrg/topology/Default.html,May 1998

[LR93] K.Lang and S.Rao. Finding Near-Optimal Cuts: An Empirical Evaluation. The 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 212-221, 1993, Austin, Texas

[LO98] D.H.Lorenz and A.Orda. QoS Routing in Networks with Uncertain Parameters. INFOCOM ’98

[MSZ96] Q.Ma, P.Steenkiste,H.Zhang. Routing high-bandwidth traffic in max-min fair share networks. SIGCOMM’96

[WCDJM98] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, G. Mendez, "Cost and Imprecision in Modeling the Position of Moving Objects", Intl. Conference on Data Engineering, 1998.

[W95] O. Wolfson, P. Sistla, S. Dao, Kailash Narayanan and Ramya Raj. “ View Maintenance in Mobile Computing”. SIGMOD RECORD, Dec 1995, Invited Article.

[V99] Nalini Venkatasubramanian, " ComPOSE|Q - A QoS-enabled Customizable Middleware Framework for Distributed Computing", Distributed Middleware Workshop, Proceedings of the IEEE Intl. Conference on Distributed Computing Systems (ICDCS '99), June 1999.

[VR97] N.Venkatasubramanian and S.Ramanathan, " Load Management for Distributed Video Servers", Proceedings of the Intl. Conference on Distributed Computing Systems (ICDCS '97), May 1997.

[HCM98] Mor Harchol-Balter, Mark E.Crovella, and Cristina D.Murta, On Choosing a Task Assignment Policy for a Distributed Server System, In Proceedings of Performance Tools ’98, Lecture Notes in Computer Science, Vol. 1468, pp. 231-242, Sep 1998

[MDZ99] Andy Myers, Peter Dinda, Hui Zhang, Performance Characteristics of Mirror Servers on the Internet, Globalcom '99
[ZT98] Wei Zhao, Satish K. Tripathi Routing Guaranteed Quality of Service connections in Integrated Service Packet Network, 1998

APPENDIX I– FEASIBILITY PROOF OF CPSS

Lemma 1: If path P_n, P_n =*{(O,* $u_{n,1}$*),(* $u_{n,1}, u_{n,2}$ *),…,(* $u_{n,k-1}, u_{n,k}$ *),(* $u_{n,k}$ *,CD)}*, is the RSP from origin *O* to Common Destination *CD* with the delay *n*. There will be one and only one server node *s* on path P_n, and it must be $u_{n,k}$.

Proof: When constructing graph *G'*, we have $\forall(u,CD)\in E'$, u is a server node. This shows that there is at least one server node on P_n, and on the path *{(O,* $u_{n,1}$*),(* $u_{n,1}, u_{n,2}$ *),…,(* $u_{n,k-1}, u_{n,k}$ *),(* $u_{n,k}$ *,CD)}*, node $u_{n,k}$ is a server node. Suppose there are two servers *s* and *s'* on path P_n. This implies that there is an edge from a server node s' to some vertices $u_{n,j}$, (s', $u_{n,j}$). But the server nodes don't have outgoing edges when constructing the graph G' and this is impossible. This proves the Lemma.

Theorem 1 An assignment with end to end delay d, $X_i^d :< p_i, s_i >$, where p_i ={(O, u_1), (u_1, u_2) ,…,($u_{k-1,} s_i$) }, satisfies the feasibility condition if $DIST_{CD}[d] < \infty$, $d \le DL_r$, and $P_d = p_i \cup (s_i, CD)$, where P_d is the corresponding feasible path with delay d.

Proof: If $DIST_{CD}[d] < \infty$, and P_d is the corresponding path, $P_d = p_i \cup (s_i, CD)$. We show that an assignment $X_i^d :< p_i, s_i >$ derived from P_d satisfies the feasibility condition. 1) Feasibility condition (1),(2) is satisfied otherwise the links and server nodes would have been removed from graph G', and because P_d is a path of graph G', so p_i and s_i satisfy the first two feasibility condition. 2) Feasibility condition 3 is satisfied because $d < DL_r$, $DIST_{CD}[d] < \infty$ and
$Delay(P_d) = EED^{X_i^d} = DL^{p_i} + RSP^{s_i} \le DL_R$.

Theorem 2 The CPSS algorithm finds an optimal assignment in *O(|* DL_r *|E')*

Proof: From a dynamic programming table structure, the update is done for each outgoing edges of a vertex for each delay constraint value. So the total time complex is *O(|* DL_r *|E').*