An Evaluation of 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.

1. INTRODUCTION

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. An efficient and adaptive resource management mechanism is highly desired in dynamic environments. There are many existing efforts in this direction. The Quality of Service (QoS) routing techniques is proposed to improve the network utilization by balancing the load among the individual network links. As a result, possible poor service quality that an end user perceives can be more and more caused by the scanty of server resources, such as “server maximum capacity reached”. Many server selection policies were also proposed based on the measurement of distance or response time. Such policies direct user to a “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 exists multiple alternative paths. While both of the two techniques can achieve some degree of load balancing, we argue that in a typical multimedia environment, where the applications are highly sensible to QoS parameters like bandwidth and delay, some high-level provisioning mechanism is required to address both the routing selection and server selection problem. Moreover, such mechanism can potentially achieve a higher system-wide utilization, and therefore allow more concurrent users, than those that just address one aspect of the problem.

In a wireless environment, with the increasing amount of mobile clients and highly dynamic network topologies, optimizing resource utilization becomes complicated. Since clients are mobile, location dependent error bursts or handoffs can frequently occur. To minimize the interruption perceived by end users, a decision whether to re-route the connection to the current servicing server or to migrate the service to a nearby server node should be reached quickly. In this case, the cost of these two operations needs to be derived and evaluated. This points to a necessity of a model to quantitatively evaluate the pair, the server and the network path leading from the client to the server, as a whole in the current system. Based on this model, we will be able to search for a best pair in the current system, and the decision of service migration can be reached.

In this paper, we present a model to address such a pair in terms of system residue capacities, and develop algorithms for searching an optimal pair per the client’s request from the current system. Our goals are in the following two points:

- By applying our Combined Path and Server Selection (CPSS) algorithm, maximize the client request success probability so that the network and server utilization can be improved.
- Load sensitive routing or scheduling techniques must be able to tolerate some information imprecision. By extensive simulation, we test and understand the performance of our CPSS algorithm and policies under different levels of information imprecision.

The remaining part is organized as the followings. Section 2 describes the server and network model used in developing the CPSS algorithm. Section 3 describes the general case of the CPSS algorithm and develops specific 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. We conclude in Section 7 with future research directions.

2. THE MODEL

For a typical multimedia application (e.g. video streaming), a request from client is sensitive to following QoS parameters: network bandwidth, server capacity and end-to-end delay. We model the request R from client c as a triple: the path, the
server and the end to end quality. \( R: \langle \text{PATH}_R, \text{SERV}_R, \text{ETOE}_R \rangle \). In [VR97], the capacity of a multimedia server can be specified as three equally important bottleneck parameters: cpu cycles, memory buffers and I/O bandwidth:

\[ \text{SERV}_R \leftarrow \text{CPU}_R, \text{BUF}_R, \text{DB}_R, \text{NIC}_R \.
\]

Given the request model, we model the system as a directed graph \( G\langle N,E \rangle \), where the distributed servers are represented as a node connected with one or more router nodes with directed edges from router to server. For a link \( l \), we use a term \( BW_{\text{avail}}^l \) 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

\[ BW_{\text{avail}}^p = \min_{l \in p} \left\{ BW_{\text{avail}}^l \right\}; \quad DL^p = \sum_{l \in p} DL_l \]

We model the server using four parameters. The first three parameters correspond to server resources: available capacity of CPU, buffers and I/O bandwidth. In order to satisfy the end to end requirement, we are also interested in a fourth parameter, the response time of a server \( s \), \( RT^s \). So given an assignment \( X=\{p,s\} \), We use the term \( EED^X \) to note the end to end delay of assignment \( X \) using path \( p \) and server \( s \). By definition, we have

\[ EED^X = DL^p + RSP^s, \quad p,s \in X . \]

In order to deal with path and server selection in a unified way, we introduce the notion of a Distance Function, that is a measure of utilization of resources in a server or a path. The distance function represents the degree of congestion and is defined using the residue capacity after assigning a client request to the server or path. We first define a utilization factor for network links and servers to quantify the residue capacity, and then proceed to define the Distance Function. The utilization factors for a link \( l \), given a request \( r \) and a parameter \( n \), is defined as

\[
UF(l,r,n) = \begin{cases} 
\frac{1}{BW_{\text{avail}} - BW_r} & \text{if } BW_{\text{avail}} > BW_r; \\
\infty & \text{otherwise}
\end{cases}
\]

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

\[
UF(s,r,n) = \begin{cases} 
\max\left(\frac{1}{CPU_{\text{avail}} - CPU_r}, \frac{1}{MEM_{\text{avail}} - MEM_r}, \frac{1}{DB_{\text{avail}} - DB_r}\right) & \text{if available capacities greater than requested} \\
\infty & \text{Otherwise}
\end{cases}
\]

It needs to be noted that proper normalization methods are needed to make the server parameters comparable to the network link parameter. We present a normalization method later in the simulation section, which derive a unit value for the server parameters from a common benchmark program.

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 will be discussed later in the simulation.\(^1\)

Given the definitions of the utilization factors, for an assignment \( X \) of both network path \( p \) and server \( s \), \( X=\{p,s\} \), we define the distance of the server \( s \) to be

\[ \text{Dist}(s,r,n) = \sum_{l \in p, p \in X} UF(l,r,n) + UF(s,r,n), \quad s \in X \]

The feasibility condition: Given a client request \( R: \langle BW_R, CPU_R, BUF_R, DB_R, DL_R \rangle \), An assignment \( X=\{p,s\} \), is feasible if and only if it satisfies all the following:

\[
\begin{align*}
BW_{\text{avail}}^p & \geq BW_R; \quad (1) \\
CPU_{\text{avail}} & \geq CPU_R, BUF_{\text{avail}} \geq BUF_R, DB_{\text{avail}} \geq DB_R; \quad (2) \\
EED^X & \leq DL_R; \quad (3)
\end{align*}
\]

\(^1\) When \( n \) equals 1, no preference is made to different UF values. For \( 0 < n < 1 \), the difference between UF values are smoothed and those relatively load resources get a higher chance to be selected, while for \( n > 1 \), the difference is amplified and those lightly load resources are highly preferred. In our simulation, we study the effects of tuning parameter \( n \) between 0.5 and 2 in the overall CPSS performance, and found that in a non-uniform traffic pattr, setting \( N<1 \) gets an optimal performance.
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 $R:\langle \text{BW}_R, \text{CPU}_R, \text{BUF}_R, \text{DB}_R, \text{DL}_R \rangle$, 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 $\text{Dist}(s^*, r, n) = \min \{\text{Dist}(s, r, n) \text{ for all } s \text{ in feasible set } S\}$.

We will discuss the CPSS policies in detail later in section 3.

3. COMBINED PATH AND SERVER SELECTION (CPSS) ALGORITHM

In this section, we present an algorithm for 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 $e_{s} = (s, CD)$, from the server $s$ to the common destination $CD$. The weight of $e$ is defined as $W(e): \langle \text{UF}, \text{DL} \rangle$, two additive parameters representing the load level and the delay respectively. In the following CPSS algorithm, we will use $\text{UF}$ to correspond to the distance (length) of the link, $\text{DL}$ to be a constraint. Specifically, the weight function $W$ in $G'$ is derived as follows: for edge $e(u,v)$ in $G'$,

$$W(e) = \langle \text{UF}(e, r, n), e_{\text{DL}} \rangle;$$

for edge $e'(s, CD)$ not in $E$, but in $E'$, define $W(e') = \langle \text{UF}(s, r, n), s_{\text{RT}} \rangle$. To simplify the graph, we remove from $G'$ those edges in which the available capacity is less than that requested. Finally we calculate a shortest path $P$ from start point $O$ to $CD$ according to the UF value of each edge, subject to the constraint of the end to end delay.

**The CPSS algorithm** ($G' \langle N', E' \rangle, R, O, n,$)

1. /* initialization */
2. For each edge $e(u,v)$ in $G'$
3. If $e$ is in $E$,
4. if $\text{UF}(e, r, n) = \infty$
5. delete edge $e$ from $G'$
6. Else
7. $W(e).\text{dist} = \text{UF}(e, r, n); \ W(e).\text{delay} = \text{DL}^e$
8. Else /* $e$ is an artificial arc, $e=(s, CD)$. */
9. if $\text{UF}(s, r, n) = \infty$
10. delete edge $e$ from $G'$
11. Else
12. $W(e).\text{dist} = \text{UF}(s, r, n); \ W(e).\text{delay} = \text{RT}^s$
13. /* run the Restricted Shortest Path algorithm for the feasible assignment set $X_f$ */
14. $X_f = P\{ (O, v_1), (v_1, v_2), \ldots, (s, CD) \} = \text{RSP}(G' \langle N', E' \rangle, W, O, CD, DLr)$
15. /* pick an optimal assignment based on policy */
16. return optimal assignment $X^* = \{ P^* \setminus (s^*, CD), s^* \}$
While it can be proved that the general case of such a Restricted Shortest Path problem is NP-hard [H92], there exists heuristic techniques (e.g. using dynamic programming) to solve the RSP problem by assuming an integer value of the delay parameter [H92, LO98, CN98]. A detailed implementation of the RSP heuristic algorithm described in [LO98] is presented in the Appendix I using a node queue technique. In this paper, we apply the RSP heuristic in our CPSS algorithm to find a feasible set $X_\gamma$ from the extended graph $G'$. We present a formal proof on the satisfaction of the feasibility condition by using our CPSS algorithm in Appendix II.

The general flow of the CPSS algorithm is as follows. A request contains 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 the resources or reject the request. When the connection terminates the client sends the termination requests, and the resources are reclaimed along the connection.

**Policies to choose an optimal assignment**

In the followings, we propose deterministic and non-deterministic CPSS policies that choose an optimal assignment from $X_\gamma$. 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 server.

**Deterministic Policies:**

- **Shortest Hop**: Choose an assignment from the feasible set $X_\gamma$ such that number of hops from source to destination is minimal.

  \[ X^* = \{ p^*, s^* \} \] \[ \text{s.t. } \text{Hop}(X^*) = \text{Min}\{\text{Hop}(X) \} \text{ for all } X \text{ in feasible set } X_\gamma. \]

- **Best UF**: Choose an assignment from the feasible set such that the utilization factor (UF) is minimal.

  \[ X^* = \{ p^*, s^* \} \] \[ \text{s.t. } \text{Dist}(X^*) = \text{UF}(p^*)+\text{UF}(s^*) = \text{Min}\{\text{Dist}(X) \} \text{ for all } X \text{ in feasible set } X_\gamma. \]

**Non-deterministic Policies:**

- **Random path selection**: select $X^* = \{ p^*, s^* \}$ randomly from the feasible set $X_\gamma$.

- **One Phase Probabilistic Policy (Prob1)**: From the feasible set $X_\gamma$, we calculate a selection probability for each Xi based on its residue capacity relative to other assignments. The probability is defined as

  \[ \text{Sel \_ Prob}(X_i) = R_1 \cdot \frac{\text{UF}^{-1}(s_i)}{\sum_{X \in X} \text{UF}^{-1}(s_i)} + R_2 \cdot \frac{\text{UF}^{-1}(p_i)}{\sum_{X \in X} \text{UF}^{-1}(p_i)} \]

  $R_1$ and $R_2$ decides 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 $R_1 = R_2 = 0.5$.

- **Two Phase Probabilistic Policy (Prob2)**: The Prob2 algorithm works as follows. From the feasible set $X_\gamma$, the first phase calculates the average UF values of path and server components for $X_\gamma$ 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 by first building a server set $S$ from $X_\gamma$, to contain distinct feasible servers. (This step is required to eliminate multiple paths in $X$ to the same server). 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 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.
Prob2(X)
1. From the feasible assignment set X, \( X = \{ X_1 :< p_1, s_1 >, \ldots, X_n :< p_n, s_n > \} \), calculate distinct server set S, \( S = \{ s_1, s_2, \ldots, s_k \} \), \( s_i \neq s_j \), \( \forall i, j \in [1, k], i \neq j \).
2. From the feasible assignment set X, calculate average UF value of network and server\(^2\), \( U_{\text{AVG}}^{NW} \) and \( U_{\text{AVG}}^{SVR} \).
3. /* Select \( s^* \) from server set S according to \( U_{\text{AVG}}^{NW} \) and \( U_{\text{AVG}}^{SVR} \):*/
4. If \( \frac{U_{\text{AVG}}^{SVR}}{U_{\text{AVG}}^{NW}} > r \), \( r \) is a threshold parameter,
5. /* Which means the network is much more loaded than the servers */
6. Weight all servers in S equally as \( \frac{1}{k} \).
7. Select a server \( s^* \) from server set S.
8. Otherwise
9. /* Which means the servers are much more loaded than the network */
10. Weight all server \( s_i \) in S as: \( \frac{U^{-1}(s_i)}{\sum_{j=1}^{k} U^{-1}(s_j)} \) to their UF value,
11. Select a server \( s^* \) from server set S
12. /* Select a best path leading to the selected server */
13. Extract \( X' \) from X, where all the server assignment in \( X' \) is \( s^* \): \( X' = \{ X_1 :< p_1, s^* >, \ldots, X_k :< p_k, s^* > \} \), and \( X' \subseteq X \).
14. Weight assignment \( X_i \) in \( X' \) as \( \frac{U^{-1}(p_i)}{\sum_{j=1}^{k} U^{-1}(p_j)} \).
15. Select an assignment \( X^* \) from \( X' \).
16. Return the selected assignment \( X^* \).

4. PARAMETER COLLECTION AND UPDATE IN A DYNAMIC ENVIRONMENT

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 ([FJPPCG99], [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. The current state information together with the topology and replica map from the directory service is then used to make QoS server and path selections. We assume that the directory is updated with current system state information by traffic probes that are distributed within the network to actively collect system state information. One example of probing module placement among typical ISP networks can be found at [GTE99], where probe modules are placed at strategic data centers to monitor servers and links within a network region. This particular placement of traffic probes can potentially capture the hotspots within the system in a cost-effective way.

Delay and bandwidth information can be highly dynamic and often follows a heavy tailed distribution [CC97]. We assume that these values are changing discretely, in contrast to moving objects where the deviation function can be continuous [WCDJM98]. 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.

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 an 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.

\(^2\) Here, in order to be comparable to server, \( UF(p_i) = \max \{ UF(l) \mid l \in p_i \} \).
4.1 Information Update Policies:

In this section, we describe two major techniques for information update to refresh state information in the directory service –
(a) snapshot based and (b) range based collection methods.

1. System Snapshot: The residue capacity information of network nodes and server nodes is based on an absolute value
obtained from a periodic snapshot. For each update period, the probes send packets to get the current information of
router nodes and server nodes, and update the directory with the collected values.

2. Range Based Collecting: 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 to be (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>, k>=0. For each update period, probes send packets to get the
current information from router and server nodes. If it finds the current value is out of the range of last updated value in
the directory, the directory is updated with another interval based range, otherwise no update is sent\(^3\). 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 Policy (PESS) : uses a lower bound of a uniform distribution corresponding to the current interval
- Optimistic Policy (OPT) : uses an expected value of a uniform distribution corresponding to the current
  interval, i.e \((H-L)/2\)
- Optimistic Favor Stable Policy(OPT2) : uses an expected value multiplied by a “Degree of Stability” calculated
  as \( \frac{H^e - L^e}{Capacity^e} \).

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 the
number of update messages needed reducing the overhead cost. The following graph shows the message overhead cost of the

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 studies 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.

\(^3\) A hysteresis algorithm is applied to dampen the oscillations between the border of two adjacent intervals.
(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 collecting period is relatively long, to study the cost-effectiveness of different policies 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 Appendix - 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.

![Topology](image.png)

Fig 2. Topology - For non-uniform traffic, the three hot pairs are: 1) C1: S4, S1:{2,5,14}, 2) C2: S1, S2:{13,15,17} 3) C3: S3:{2,3,8} Assume that the clients and servers are directly behind the router nodes being addressed.

To better emulate the real network, the capacity 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:

(a) Modeling Requests: 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. 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.

(b) Modeling 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

---

4 The node number in the graph
5 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.
6 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.
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, \( X: \{X1:<p1,s1>, X2:<p2,s2>, \ldots, Xn:<pn,sn>\} \), calculated by CPSS, 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

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.

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 period. 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 the network and servers, so the network and server rejection rate is very low. It can also be observed from Fig [3].b that in heavy load non-uniform traffic environment, the servers are more saturated than the network; hence server capacity appears to be the restricting factor. When the sstate 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 the graph, where the DS rejection is almost zero, and server rejection dominates 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

Fig 4: Shortest Hop Policy In Non-uniform Traffic

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] in general because $UF(path)=\sum(UF(link of path))$. The longer the path, the bigger the resulting UF value. 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 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 which are already very congested. Hence, a large number of request rejections in the Shortest Hop policy result from path rejections in the network. 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 5: Hop Length of CPSS Policies

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

The random policy tries to avoid the oscillation of static policies 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. The following graphs show 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 which 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 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.
5.2.5  **The Prob2 Policy**

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

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[11] 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.

5.3.1 **Snapshot Based Information Update:**

The snapshot based information update can be regarded as a special case of interval based update 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 connection holding time. For instance, in our experiments the average connection holding time is set to 10 min (600s). Update period values larger than 600s show negligible effects on the performance of CPSS policies.

Fig 9: CPSS Policies with frequent information updates. Update period = 5 sec.

If the system state information is updated very frequently, our results show that Best UF policy is superior. 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.

5.3.2 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. 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 [11.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 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.
5.4 Effect of the parameter “n”:

Fig[10] depicts the performance of Best UF with different values of parameter N, N=0.5, 1,1.5,2,3, under lightly loaded non-uniform traffic patterns. We observe that smaller N values, (N=0.5) consistently perform better with non-uniform traffic patterns than values of N>1 in larger update periods (t>100s). For smaller update periods, t = 5s, N=1 or 2 performs best. With non-uniform traffic and large N values, either the server or network becomes a restricting resource, since larger values of N imply strongly favoring lightly loaded resources, eventually leading to congestion. A smaller N value (N<1) tends to smooth the differences of UF values between different resources, balancing the relative load and reducing congestion – this comes at the expense of sub-optimality in the individual assignments. With uniform traffic patterns, the influence of parameter N is less evident, since uniform traffic exhibits self-balancing properties.
The relationship of N with the update period is interesting. With a smaller value of the update period, larger N values tend to assign requests to resources with a large residue capacity so as to reduce the overall system congestion. So it can perform better provided updated system state. With a large update period, a larger N values strongly favors the same “optimal” assignment; over time these resources become congested and the optimal assignments become suboptimal. We notice that the actual value of N (N>1) in this case, does not exhibit significant difference. With a larger update period and smaller N values (N < 1), the differences of the UF value of each link and server tend to be smoothed, \((UF=(1/residue)^N)\), and oscillation effect is reduced.

5.5 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 are 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 which 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. 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. The overview of the system model

We propose a system layout based on the notion of QoS Broker [diffserv,klara, BB perf], in which the broker maintain a topological view of network as well as a yellow page listing all the services available at each server under provisioning. Since we are interested in a multimedia context, especially video streaming applications, the services available at a server are to be viewed as titles of the available video programs stored at the server. In addition to the topological knowledge, the broker also maintains an approximation of current load situations of both network links and servers. We will discuss the performance implication of maintaining such system state parameters in section 4, where we evaluated the performance gain of having such knowledge against the overhead of the parameter collecting process.

When a request with certain QoS requirement \(R: <PATH,SERV,ETOE>\), for a particular video program \(V\), comes to a client, the client direct the request to the broker. The broker calculates an optimal assignment \(X*: <p*, s*>\) using CPSS algorithm, and reply the client with \(X*\). The client then proceed with a source routing into the network. A nice feature of this scheme is that the routers and the servers don’t have to (although they may elect to) perform admission control and reserve resources along the forwarding path. In stead, they may retain their stateless property and rely on the broker to do a measurement-based admission control (ref[ref]) in this case, the core routers don’t have to maintain a per-flow state information, all they need to do is to forward packets along a pre-specified path. In a differentiated service environment, the
forwarding techniques for source route packets among stateless routers are proposed and evaluated, for example in [huiZhang]. Fig N.a depicts such a process. The probes that distributed among the network collect the system state information and update the broker when necessary. For autonomy and scalability considerations, the probes only report to the brokers of their own autonomous region. According to some pre-specified agreement, brokers in different region may exchange information to share resource, so that when the request can’t be satisfied within its original autonomous region, the broker can take a forwarding decision. The information exchanged between brokers can be aggregated for efficiency and scalability considerations [pnni, qos-aggr]. In this case, the broker decides which region to forward the request to, but can’t perform admission control for the resources in other region.

The strength of our system model is that it requires a minimum change of the existing network, and will fit easily into the future differentiated service networks. Meanwhile, in this architecture, the service quality that an end user perceives can be represented and provisioned in a well defined way. And the utilization of both the network resources and service node resources can be improved in a balanced manner so that the overall system throughput in terms of maximum concurrent users can be further improved. Meanwhile, that it requires a minimum change of the existing network, and will fit easily into the future differentiated service networks.

7. RELATED WORK

We describe related work in the areas of replicated service selection, QoS routing and mobile information update. 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 [FPLZ98, FJPZGJ99, KSS98]. Dynamic server selection policies 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. In this protocol, the server actively pushes load information to the resolver; network load information obtained by probes scattered in the network are also stored in the resolver. Tradeoffs studied include selective update of the resolver in order to save the overhead and maintain the accuracy. The anycasting work is closely related to ours since it considers both network path quality as well as server quality. While their focus is 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. [MDZ99] Performance studies of Mirrored Web Servers on the Internet [MDZ99] indicated that server performance varies widely and that a server’s performance relative to other servers is more stable and independent of time scale. The study determined that clients wishing to achieve near-optimal performance might only need to consider a small number of servers rather than all mirrors of a particular site. 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. This work introduced the notion of a Load Factor. LF(Si, Ri) that calculates how congested a particular server Si will become if it is assigned request Ri. To complement the server scheduling, [VR97] also developed a predictive placement strategy, which actively determines when, where and how many replicas to create based on a prediction of future request patterns. Our work expands on this by considering wide-area distributed servers and including path selection into the server selection process.

QoS-based routing techniques have been explored in depth[CN99, ZT98,CRS97,BS98,BEZ93]. The Maxmin QoS Routing Algorithm[MSZ96] provides load sensitive routing for best effort flows to deliver packets along least utilized links. The proposed Maxmin technique routes the best effort traffic along those links on which there are fewer existing flows so that they have a large fair share bandwidth for a connection under consideration. QoS-based extensions of the traditional Bellman-Ford/Dijkstra algorithm [BG92] incorporate an available bandwidth parameter and use widest-shortest path [GOW98,CN98] calculations. Such extensions can easily be applied to OSPF-based routing making it bandwidth sensitive. Experimental efforts study the combined performance and overhead cost of different information collecting and routing policies [AGKT98]. The studies 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]. This characterization can be used to find a most probable path satisfying the requested bandwidth, delay, etc. based on the probability distributions of each network link. While the problem of finding a most probable path satisfying the additive QoS parameters, like delay is NP-Hard, efficient heuristics have been developed. A heuristic to the Most Probable-Optimal Partition problem, MP-OP [LO98] has a complexity of $O(|E|^2 |D|)$ in finding a most probable path for uniform distribution; the heuristic uses a dynamic programming solution developed in the context of RSP problems[H92].

8. Future Research Directions

In this paper, we have described our approach to providing combined path and server selection in a dynamic MM environment using state information held in a directory service. We have analyzed a number of CPSS policies and evaluated their performance under different conditions of imprecision. We are currently extending our work in the following directions.

In order to perform effective path and server selection based on uncertainty parameters, we are currently enhancing the basic CPSS algorithm by incorporating a probability model for network link and server parameters. We have developed two methods to enhance the basic CPSS: the Simple and Statistical methods [ZV99]. In the Simple Method, the objective is to derive values from a chosen probability distribution according to a policy, and use them to fit into the overall structure of the CPSS calculation. The Statistical Method uses an extended optimality condition (ε - Optimality) that attempts to satisfy bottleneck and additive constraints simultaneously using a multistep process. We are also exploring variations of the range based update techniques such as exponential interval and adaptive dynamic range based techniques. While the earlier techniques deal with fixed ranges and are history insensitive, the adaptive dynamic range based scheme alters the range dynamically using a history sensitive update procedure [ZV99]. We are also actively studying information collection policies, such as probe-based, server push etc. and evaluating the update and collection overheads associated with information collection. In this paper, we assume a given replica/placement model that is used in server and path selection. We intend to explore specific placement policies using dynamic replication and migration in a wide-area scenario. We will then integrate these placement techniques with request scheduling and allocation in the presence of a specific data placement.

We hope to eventually apply and evaluate the developed techniques to provide QoS-enabled multimedia in mobile and wireless environments. This will involve an understanding of issues such as mobile object location [WCDJM98], dynamic re-routing and rescheduling of multimedia requests to accommodate the mobility and maintenance of mobile object directory services[W95]. 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


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


[GO98] Roch A. Guerin, Ariel Orda, and Douglas Williams, QoS Routing Mechanisms and OSPF Extensions, Globecom'98


APPENDIX I– THE RSP IMPLEMENTATIONS

RSP Algorithm: \( (G'\times N', E'\times W, O, CD, DL, rDL) \)

1. For \( d = 0 \) to \( |DL| \)
2. \( \text{DIST}_v[d] = \infty \), for all nodes \( v \in N \)
3. Empty \( N_Q \)
4. \( \text{DIST}_O[0] \leftarrow 0 \); Push \( \text{DIST}_O[0] \) into \( N_Q \)
5. While \( N_Q \) not empty
6. Pop an entry \( \text{DIST}_v[\text{delay}_v] \) from \( N_Q \)
7. For all adjacent node \( u \)
8. If \( \text{delay}_v + W(u,v).\text{delay} \leq rDL \)
9. If \( \text{DIST}_u[\text{delay}_v+W(u,v).\text{delay}] > \text{DIST}_v[\text{delay}_v] + W(u,v).\text{dist} \)
10. \( \text{DIST}_u[\text{delay}_v+W(u,v).\text{delay}] = \text{DIST}_v[\text{delay}_v] + W(u,v).\text{dist} \)
11. \( \text{PRE}_u[\text{delay}_v+W(u,v).\text{delay}] = v \)
12. If \( u \) not in \( N_Q \)
13. Push \( u \) to \( N_Q \)
14. /*calculate feasible assignment set \( X \) based on \( CDDIST \) array and the corresponding path*/
15. For \( d=0 \) to \( |DL| \)
16. if \( CDDIST[d] < \infty \)
17. \( \text{calculate } P_d, P_d = \{(O, u_{d,1}), (u_{d,1}, u_{d,2}), \ldots, (u_{d,k}, CD)\} \), from \( \text{PRE} \) table
18. \( X^d_i :< p_i, s_i > = \langle\{(O, u_{d,1}), (u_{d,1}, u_{d,2}), \ldots, (u_{d,k-1}, u_{d,k})\}, u_{d,k} > \)
19. return \( X \)

(a) Input: an extended graph \( G' \), an array of the weights of each edge \( W \), the source and destination node \( O, CD \), and a requested end to end delay constraint \( DL \), for a request \( r \).

(b) Output: An array, \( CDDIST[n] \) where \( n \leq DL \), and the corresponding feasible path from \( O \) to \( CD \) with delay \( n \), \( P_n, P_n = \{(O, u_{n,1}), (u_{n,1}, u_{n,2}), \ldots, (u_{n,k-1}, u_{n,k}), (u_{n,k}, CD)\} \), where \( CDDIST[n] < \infty \). (If there are multiple RSPs for a particular delay value, the first calculated path is retained and other equivalent RSPs are discarded).

(c) The feasible assignments set \( X, X=\{X^d_i :< p_i, s_i >\} \), where \( p_i \) is the path assignment, and \( s_i \) is the server assignment, and \( d \) is the end to end delay, is defined as:
\[ \forall X^d_i :< p_i, s_i > \in X : d \in (0, DL), \text{DIST}_{CD}[d] < \infty \).

Let the corresponding path \( P_d = \{(O, u_{d,1}), (u_{d,1}, u_{d,2}), \ldots, (u_{d,k-1}, u_{d,k}), (u_{d,k}, CD)\} \). We define \( p_i = \{(O, u_{d,1}), (u_{d,1}, u_{d,2}), \ldots, (u_{d,k-1}, u_{d,k})\} \), and \( s_i = u_{d,k} \).

The running time of the algorithm is \( O(|DL| \cdot |E'|) \) [LO98]. Observe that as the delay constraints are relaxed, more options are open and hence the algorithm requires more running time.

APPENDIX II– FEASIBILITY PROOF OF CPSS

Lemma 1: If path \( P_n, P_n = \{(O, u_{n,1}), (u_{n,1}, u_{n,2}), \ldots, (u_{n,k-1}, u_{n,k}), (u_{n,k}, CD)\} \), 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 $P_u$, and on the path $\{(O, u_{n,k}), (u_{n,1}, u_{n,k}), \ldots, (u_{n,k-1}, u_{n,k}), (u_{n,k}, CD)\}$, node $u_{n,k}$ is a server node. Suppose there are two servers $s$ and $s'$ on path $P_u$. 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 $d$, \( X^d_i: <p_i, s_i>, \) where $p_i = \{(O, u_i), (u_i, u_{i+1}), \ldots, (u_{i,k}, s_i)\}$, satisfies the feasibility condition if $\text{DIST}_{CD}[d] < \infty$, $d \leq DL$, and $P_d = p_i \cup (s_i, CD)$, where $P_d$ is the corresponding feasible path with delay $d$.

**Proof:** If $\text{DIST}_{CD}[d] < \infty$, and $P_d$ is the corresponding path, $P_d = p_i \cup (s_i, CD)$. We show that an assignment $X^d_i: <p_i, s_i>$ 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 $d < DL$, $\text{DIST}_{CD}[d] < \infty$ and $\text{Delay}(P_d) = \text{EED}^{X_i} = DL^p + RSP^p \leq DL_R$.

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

**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 $O(|DL|E')$.

### APPENDIX III- The Queuing Delay Calculation:

Here, we propose two simple approximations to calculate queuing delay based on the residue capacity. To better emulate the network in the real world, the proposed models should 1) be exponential to residual capacity; 2) not use a fixed value to characterize link parameters, e.g., bandwidth, but is different to each flow with different packet size and burstiness.

The two delay calculation models assume different network environments.

In the network which supports only best effort traffic, i.e. there is no buffer space or link capacity reserved for each flow, and no fixed packet scheduling mechanisms. All the flows on the same link will statistically meet the same delay situation. Adding a new flow causes the delay situation to change for all the existing links. In such network, assume that a network link $l$ has residual capacity $R_l$ and all the existing flows on link $l$ have delay $D(l)$. Now assume that a flow $f$ with mean rate $BW_f$, packet size $p_f$ and burst size $\delta_f$, is added to the link $l$. The expected delay of flow $f$, if added to link $l$, noted as $D_f(l)$, can be approximated as:

$$D_f(l) = D(l) + \frac{\delta_f \cdot p_f}{R_l} \cdot \frac{BW_f}{R_l}.$$  

The justification is that 1) The delay increases monotonically, 2) $\frac{\delta_f \cdot p_f}{R_l}$ approximates the maximum queuing delay in a burst, and $\frac{BW_f}{R_l}$ is the probability that this flow gets in such burst.

In a network that supports resource reservation, we expect different queuing delays for different flows on the same link. For each flow on the link, once the flow is admitted, the bandwidth capacity and buffer space are allocated accordingly. We assume that there is enough buffer space on the link, so the queuing delay of each individual flow should mostly be related to its own traffic pattern. Again, assume that a network link $l$ has residual capacity $R_l$. Assume that a flow $f$ with mean rate $BW_f$, packet size $p_f$ and burst size $\delta_f$, is added to the link $l$. The expected delay of flow $f$, if added to link $l$, noted as $D_f(l)$, is approximated as:

$$D_f(l) = \frac{\delta_f \cdot p_f}{BW_f} \cdot \frac{BW_f}{R_l} = \frac{\delta_f \cdot p_f}{R_l}.$$
The justification is that 1) The delay value of individual flow depends on its own traffic pattern, 2) $\delta_f \cdot P_f$ approximates the maximum waiting time of a packet in case of burst, and $\frac{BW_f}{R_f}$ approximates the probability that this flow gets into such a burst.

The following graph show the numerical results of our second delay model (i.e. with resource reservation). It shows the approximate delay values of two flows with maximum burst being 10 and 20 respectively and packet size being 750 bytes. The queuing delay of both flows grows exponentially with the residue capacity and the more bursty flow consistently has a higher queuing delay then the less bursty one.

![Queuing Delay Approximation](image)

Fig 12: Queuing Delay Approximation