Scalable QoS-Based Resource Allocation in Hierarchical Networked Environment *

Sourav Ghosh† Ragunathan (Raj) Rajkumar‡ Jeffery Hansen§ John Lehoczky¶

Abstract

In this paper, we study the problem of allocating end-to-end bandwidth to each of multiple traffic flows in a large-scale network. We adopt the QoS-based Resource Allocation Model (Q-RAM) [9], whereby each flow derives an utility based on the amount of its allocated bandwidth. Our goal therefore is to maximize the total utility derived across all network flows. The NP-hard nature of the resource allocation problem is compounded by the need to select an appropriate path between each source-destination pair. We propose a hierarchical decomposition scheme that allows the resource allocation problem to be solved in a decentralized and scalable fashion. The hierarchy we use is based on a (natural) partitioning of the network into subnets, with resource allocation decisions made on a subnet-by-subnet basis. A novel distributed transaction scheme is used to ensure that resource allocations are consistent across all the subnets traversed by each flow. We provide both analytical and experimental evidence to show that our scheme is very scalable and yet does not sacrifice the quality of the allocations.

1. Introduction

Examples of distributed networked systems include the Internet, sensor networks, autonomous systems and overlay networks. In order to provide QoS to tasks executing on these systems, we need to guarantee the allocation and scheduling of resources. The resources include computational cycles, storage and *network bandwidth* across a *route* between the source and the destination. For example, a typical video transmission application requires a certain amount of network bandwidth and CPU cycles from various network links and routers respectively. Higher quality in terms of its frame rates and resolutions requires a greater quantity of these resources.

For a large number of tasks to be deployed on a system consisting of a large number of resources, we designed a hierarchical scheme in [6] that provides near-optimal resource allocation in a scalable manner. The hierarchical technique divides the problem into smaller independent sub-problems. Specifically, it divides the system into identical subsystems, assigns tasks to these subsystems in an equitable fashion so that each subsystem obtains an (nearly) identical number of tasks of the same *type*, and then makes resource allocation decisions within each subsystem *independently*. Implementing this scheme on a networked system, however, presents two major difficulties. First, it is difficult to divide a networked system into a number of identical subsystems if the architecture is heterogeneous (even if it is hierarchical). Secondly and most importantly, it is not possible to isolate the subsystems in the network. This is because the route of a task can potentially span a very large number of links and routers over the entire network. If we consider each network subdomain as a subsystem, many tasks can have routes across multiple sub-domains and thus the resource allocation in one subsystem may be dependent on that obtained in another and vice versa. Hence, multiple subsystems need to *negotiate* with each other in order to determine near-optimal resource allocations.

2. Related Work

There have been several contributions in the field of QoS in networks, especially in the context of the Internet and ATM. These contributions to network QoS are coarsely divided into four categories: (1) the selection of a route between the source and the destination, (2) the bandwidth reservation across the route, (3) the scheduling of network packets at each router across the route and (4) the choice of QoS-based resource allocation algorithms.

For example, Ma and Steenkiste investigated several route selection schemes for flows with a fixed bandwidth requirement [11]. The goal of their QoS routing is to select a feasible route if one exists, and the route leading to the best re-

* This work was supported by a DARPA Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research under Grant N00014-01-1-0576 and by DARPA under contract number F33615-00-C-1729.

† Carnegie Mellon University, Department of Electrical and Computer Engineering, sourav@cs.cmu.edu

‡ Carnegie Mellon University, Department of Electrical and Computer Engineering, raj@ece.cmu.edu

§ Carnegie Mellon University, Institute for Complex Engineered Systems, hansen@cmu.edu

¶ Carnegie Mellon University, Department of Statistics. jpl@stat.cmu.edu

source efficiency is chosen if multiple routes are available. Nahrstedt *et al* also made contributions in the field of QoS-aware routing. First, they used topology aggregation of hierarchically structured networks in order to provide routing for tasks involving QoS requirements related to bandwidth and delay guarantees [3, 10]. In a hierarchical routing, nodes are clustered into groups, which are further clustered into higher-level groups, creating a multi-level hierarchy. Second, they also presented distributed *ticket-based* routing, which is designed to work with imprecise state information [2]. As a signaling protocol for network bandwidth reservation, *Resource ReserVation Protocol* (RSVP) is a popular example[18]. It provides a mechanism to establish a reservation over a route between the destination and the source. It is designed to work in conjunction with existing routing and scheduling protocols. There are also many QoS-aware packet scheduling disciplines available as extensions of the Generalized Processor Sharing algorithm [14], and otherwise [12][8][5].

2.1. Our Contribution

In the context of network QoS, we make our contribution in network bandwidth allocation and route selection. However, our model differs in two fundamental ways. First, our Q-RAM-based QoS model allows a task/flow to specify multiple levels of bandwidth and delay requirements for different levels of service. Second, our resource allocation scheme determines the allocation of a *near-optimal* route and a *near-optimal* network bandwidth along the route for each flow. The scheme relies on a signaling protocol such as RSVP and packet scheduling policies across the network in order to satisfy the network bandwidth reservation. In addition, it can also exploit the existing routing protocols to perform efficient optimization.

3. Modeling of Networked System

In this section, we describe our model of a distributed networked system. We first briefly describe our generic resource allocation model based on Q-RAM. Next, we introduce a graph-theoretical model of the network and demonstrate how to formulate and solve the network QoS optimization problem in Q-RAM.

3.1. Network Model and QoS

We assume the network is a distributed system consisting of multiple resources where each resource corresponds to the link capacity in terms of the available bandwidth of the link[1]. We consider a set of tasks or flows that transfer data from one node in the network to another. Each task has a set of QoS set-points in terms of bandwidth and delay requirements. In addition, there is a utility associated with each of its set-points. In general, a higher bandwidth provides higher quality and hence higher utility for a task. If a network is modeled as an undirected graph, these tasks can be modeled as flows across the graph with variable capacity requirements.

Q-RAM optimization in a network works as follows. Using the edges of the graph as network links with a certain amount of bandwidth R, we construct a resource capacity vector $\vec{R} = R_1, \ldots, R_m$ where m is the total number of weighted edges of the graph and R_i is the bandwidth of the i^{th} edge. We enumerate the operational dimensions of each task as follows.

3.1.1. Set of bandwidth settings : The number of choices of bandwidth settings of a task τ_i is given by:

$$B_i = \{b_{i1}, \cdots, b_{iN_i^B}\}, \tag{1}$$

where, N_i^B = number of possible bandwidth settings for task τ_i. The bandwidth maps directly to the resource requirement on the network link.

3.1.2. Set of delay settings : The number of choices of delay settings of τ_i is given by:

$$D_i = d_{i1}, \cdots, d_{iN_i^D}, \tag{2}$$

where N_i^D = number of delay levels for τ_i.

The network delay encountered by a flow is dependent on the value of total bandwidth (or speed) of the network link(s) used. It is expressed as the sum of three components: (1) circuit delay (propagation delay of 1 bit), (2) transmission delay, and (3) switching delay [15]. The switching delay is the dominant factor in the overall delay, which is in turn dependent on the scheduling policies across the routers. Since our QoS model deals with resource allocation that separates it from the scheduling concern at the lower level, we only need to consider the bandwidth of the links for our model. We assume that once the bandwidth has been allocated, the router will have enough processing cycles to process the packets between its incoming and outgoing links, and its lower level packet scheduler can schedule the packets appropriately so that each flow meets their deadlines [2]. In other words, we express delay as the number of hops along a route in this paper.

3.1.3. Set of routes : The number of choices of routes of a task τ_i is given by:

$$P_i = p_{i1} \times \cdots \times p_{iN_i^P} \tag{3}$$

For a connected graph, we always have $|P_i| \geq 1$. The set of routes can be derived in several possible ways. For example, in order to determine all possible routes between a source and a destination, the source node can broadcast its route discovery request to all of its neighbors. If a neighbor is not the destination node itself, it forwards the request to its other neighbors.

1 It is relatively straightforward to extend our formulation to include processing resources but we do not do so for simplicity of presentation.

2 A lot of work in packet scheduling has been done in the past with varying degree of schedulable utilization bounds on the routers [16, 8, 5].

3.1.4. Basic Q-RAM Algorithm By combining Equations (1), (2) and (3), we obtain the set-points of the tasks $\{S_i : B_i \times D_i \times P_i\}$. The utility of a set-point is obtained from the QoS dimensions as $\{B_i \rightarrow u\}$, while the corresponding resource requirements are obtained as $\{B_i \times D_i \times P_i \rightarrow R\}$. Thus a set-point is represented by $\{q_j, u_j, (r_{j_1}, \ldots, r_{j_m}), h_j\}$ where

$q_j =$ Quality level,

$u_j =$ Utility level,

$(r_{j_1}, \ldots, r_{j_m}) =$ resource vector representing resource requirement at each edge of the system, and

$h_j =$ compound resource describing the cost of allocating resources.

The procedure is detailed in Algorithm 1. It is the most

input : profiles of tasks with bandwidth and network routes
output : route and bandwidth allocation of tasks by maximizing utility
for *Each task* $i = 0$ *to* n **do**
  Determine QoS points as bandwidths B_i;
  Determine P_i as the set of resource options ;
  Generate set-points $S_i = B_i \times D_i \times P_i$ for τ_i and map to resource requirements $S_i \rightarrow R$ in terms of link bandwidths;
  Determine "compound resource" as a scalar cost metric for each set-point;
  Determine concave majorant of the set-points based on their (compound resource, utility) values and the corresponding gradient;
end
Merge set-points of n tasks with decreasing values of their gradients;
Perform a global resource allocation starting with the point of highest gradient;

Algorithm 1: Basic Global QoS Optimization For Networks

direct way of solving the problem of network bandwidth allocation in Q-RAM. However, there are two main drawbacks to this approach.

First, it requires each task to *enumerate* all of its set-points, which, in turn, requires them to determine all possible routes P_i between the source and destination. As the size of the network increases, $|P_i|$ increases exponentially, and the complexity of the whole route discovery process supersedes the complexity of the optimization, making the process intractable for large networks. Therefore, we must use an efficient route discovery technique that can exploit the architecture of the network, namely hierarchical route discovery [7][10].

Second, suppose that each task has a *small* set(≤ 10 for example) of QoS levels for the sake of simplicity. Even in this case, since P_i is the enumerated list of *all* routes between two nodes in the network, it can potentially be very large. Therefore, we must select a few routes to make the problem tractable. The challenge is to pick these few routes such that the resulting utility is close to what would be achieved if the exhaustive lists of routes were considered.

4. Hierarchical Network Architecture

In this section, we first formulate the hierarchical network architecture using Graph-theoretical techniques. Next, we describe how this formulation can be used in decomposing our optimization process.

4.1. Graph-Theoretical Representation

We follow the description of the hierarchical network model as presented for the Internet [1, 7, 17].

The entire network is represented as a connected undirected graph $G = (V, E)$ as shown in Figure 4.1, where V denotes the set of vertices and E denotes the number of edges. The nodes or vertices of a graph represent switches, and the edges represent links. The **bandwidth** across each link e_j is expressed as the **capacity** c_j of an edge in the graph. If the network is hierarchically organized, G_p represents the network architecture at a particular layer p.

The nodes get clustered to form the graph of the next layer. The nodes of the same layer that are clustered into the same higher layer are said to belong to the same *peer group* [4]. At a particular layer, a set of edges partition the graph into multiple induced subgraphs, whose vertices form peer groups. This set of edges defines the edges of the graph at the next higher layer. We call these edges *backbone-edges*. If two subgraphs are connected by a single edge, their connecting backbone-edge becomes a cut-edge of the graph.

If we collapse all the vertices and edges of a subgraph G_i of G into a single vertex, it is called a *supervertex*. Thus the graph at a higher layer is the supervertex graph of that of the next lower layer. This layered architecture is illustrated in Figure 4.1. Expanding each supervertex at any layer reveals the entire network of nodes in that subgraph at the lower layer.

Let us consider a task that sends data from a source node x to a destination node y. We define $P_G(x, y)$ to be the set of all possible routes from x to y. For a connected graph, we have $|P_G(x, y)| \geq 1$. Let us also define $p_G(x, y) \in P_G(x, y)$ as a particular route from x to y. This is formed by concatenating a set of edges that connect x and y. This includes the edges inside multiple sub-graphs and the backbone edges connecting them. Let us assume that V_x and V_y are sets of vertices of two subgraphs of G such that $x \in V_x$,$y \in V_y$ and $V_x \cap V_y = \emptyset$. Let the supervertices v'_x and v'_y of the supervertex graph G' represent the sets of vertices V_x and V_y in the original graph G. By definition, $P_{G'}(v'_x, v'_y)$ denotes the set of routes between the supervertices v'_x and v'_y. Therefore, for every $p_{G'}(v'_x, v'_y) \in P_{G'}(v'_x, v'_y)$, there is at least one corresponding $p_G(x, y) \in P_G(x, y)$.

Definition 1 (Border vertices). *The vertices in two different induced sub-graphs that are connected by one or more backbone-edges are known as border vertices.*

Definition 2 (Sub-Route). *The set of edges of a particular route connecting two border vertices of an induced sub-*

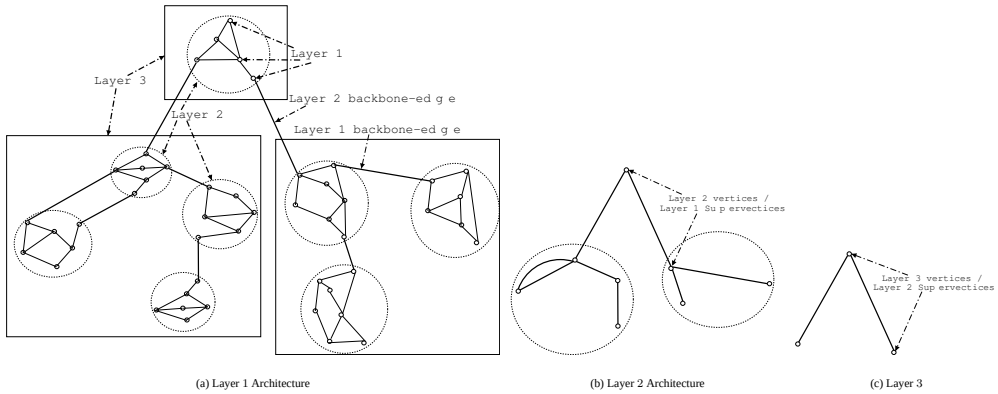

Figure 1. Hierarchical Graph Model of Network

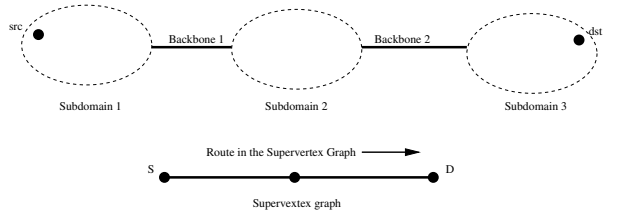

Figure 2. Sub-domain and Supervertex Graph for $|P_{G'}(v'_x, v'_y)| = 1$

graph between two backbone-edges is called a "sub-route" or a "child-route".

Definition 3 (Parent Route). *The route in the supervertex graph that connects the source and the destination supervertices is called the "parent route" of the "sub-routes" internal to each supervertex of the (supervertex) graph.*

According to the above definitions, each parent route has sub-routes within each supervertex it connects. Using the same notation, $P_{G'}(v'_x, v'_y)$ denotes the set of parent routes, and each element in $P_G(x, y)$ consists of a concatenation of the edges from a route in $P_{G'}(v'_x, v'_y)$ and its sub-routes one from each of the supervertices it traverses. As an example, in the case of the Internet, border vertices denote the edge routers that connect two sub-domains, a parent route represents a route corresponding to "Inter-domain routing" and a sub-route represents that corresponding to "Intra-domain routing".

Next, we state Lemmas dealing with route selection for a given flow with a fixed capacity (or bandwidth) constraint.

Lemma 1 (Backbone edge and Route selection). *If all routes in $P_G(x, y)$ share the same set of backbone edges, in Graph G, then $|P_{G'}(v'_x, v'_y)| = 1$.*

Proof. If all routes in P_G share the same set of backbone edges, they go through the same set of subgraphs. In the supervertex graph G', these subgraphs are replaced by vertices. Thus all routes in $P_G(x, y)$ collapse to having the same set of supervertices and hence are connected by the same set of edges in G'. Therefore they collapse to a single route. In other words, $|P_{G'}(v'_x, v'_y)| = 1$. □

Let us consider the network of 3 sub-domains illustrated in Figure 2. The source node is present in Sub-domain 1 while the destination node is present in Sub-domain 3. As can be seen from the figure, every route connecting the source "src" and the destination "dst" has to go through the same sub-domains 1, 2, 3 and the backbone edges 1 and 2 connecting those sub-domains. Hence, in the supervertex graph, all routes collapse to a single route that traverses across 3 supervertices.

Next, we would like to determine the routes internal to each sub-domain. Using the same example in Figure 2, we build a complete route between the source and the destination by selecting a sub-route within each sub-domain that connects the backbone edges. We can have multiple possible choices of sub-routes inside each sub-domain. If the selection of the sub-route in one sub-domain does not affect the same at another, we say that the sub-routes can be chosen *independently* of each other. Based on that, we state Lemma 2 under the situation where we would like to determine a route of a particular bandwidth for a flow.

Lemma 2 (Independent Sub-Route Selection). *For a fixed route $p_{G'}(v'_x, v'_y) \in P_{G'}(v'_x, v'_y)$ in the supervertex graph with a fixed capacity (bandwidth) requirement, the sub-routes inside each sub-graph can be chosen independently of each other.*

Proof. Let us consider a hierarchical Graph G consisting of multiple induced subgraphs and backbone edges joining them. The source node and the destination node of a particular task are denoted by x and y respectively. Any route $p_G(x, y) \in P_G(x, y)$ traverses a fixed set of subgraphs $g_1, .., g_l$ and a fixed set of backbone edges $L_1, .., L_{l-1}$. If $p_{g_1}, .., p_{g_l}$ are the sub-routes in the respective subgraphs $g_1, .., g_l$ of the route $p_G(x, y)$, then we express $p_G(x, y)$ as

$p_G(x,y) = p_{g_1} \cdot L_1 \cdot p_{g_2} \cdot \ldots \cdot L_{l-1} \cdot p_{g_l}$ and the corresponding $p_{G'}$ as $p_{G'}(v'_x, v'_y) = L_1 \cdot \ldots \cdot L_{l-1}$.

The maximum capacity of the route $p_G(x,y)$ is given by

$$c(p_G(x,y)) = min(c(p_{g_1}), c(L_1), ..., c(L_{l-1}), c(p_{g_l})), \quad (4)$$

and that of $p_{G'}(v'_x, v'_y)$ is given by

$$c(p_{G'}(v'_x, v'_y)) = min(c(L_1), c(L_2), ..., c(L_{l-1})). \quad (5)$$

Combining Equations (4) and (5), we obtain:

$$\Rightarrow c(p_G(x,y)) \leq min(c(p_{g_1}), c(p_{g_2}), ..., c(p_{g_l}), \quad (6)$$
$$\Rightarrow c(p_G(x,y)) \leq c(p_{g_i}), \ \forall 1 \leq i \leq l. \quad (7)$$

This shows that selecting edges inside each subgraph can be performed independently under a fixed capacity constraint. □

Delay and Hierarchical Routing Lemma 2 holds true when delay is not considered. The approximated delay is the main drawback of hierarchical routing [10]. In order to satisfy the delay constraint in terms of the number hops as mentioned in Section 3.1, we divide the delay requirements equally in each subgraph falling in the route, similar to what is done in [8].

Based on Lemma 1 and Lemma 2, we state a lemma on the complexity of route selections.

Lemma 3 (Complexity of Route Selections). *Suppose all routes in $P_G(x,y)$ share the same set of backbone edges $L_1, \ldots, L_{l-1}$, and hence the same set of subgraphs $g_1, \ldots, g_l$ in Graph G. Furthermore, suppose that the set of edges for the route within a subgraph g_i can be chosen in s_i different ways under a bandwidth constraint. Then the number of possible routes is $\prod_{i=1}^{l} s_i$ and the number of computational steps required to choose a route is $\sum_{i=1}^{l} s_i$.*

Proof. Using the notation from (4), the set of links p_{g_1} satisfying the bandwidth constraint from the sub-graph g_1 can be chosen in s_1 different ways. From Lemma 2, for each choice in g_1, we can choose the set of links in g_2 by s_2 different ways and so on. Therefore, the maximum number of possible ways a route can be selected is $s_1 \times \ldots \times s_l = \prod_{i=1}^{l} s_i$.

Next, the number of steps required to choose the *near-optimal* set of edges inside a subgraph g_i (sub-route) is s_i. Since all routes map to a single route in the supervertex domain, Lemma 2 proved that the selection of edges in each subgraph can be done independent of each other under a fixed capacity requirement. Therefore, the maximum number of steps required to choose a suitable route is $s_1 + \ldots + s_l = \sum_{i=1}^{l} s_i$.

□

Based on Lemma 3, we describe our hierarchical route discovery method next. Later, we will also discuss how it also assists in hierarchical QoS optimization.

4.2. Hierarchical Route Discovery

We employ the hierarchical route discovery that is currently employed in the Internet. We obtain the set of routes for a task at its highest level of network hierarchy. Next, for each of the (super)vertices in each route, we obtain the sub-routes inside the subgraphs represented by those vertices. The process starts with the highest level of the task and continues to the lowest level of the hierarchy. From Lemma 3, if we would like to determine η_{th} number of sub-routes for each sub-domain, the complexity of hierarchical route discovery is $O(p\eta_{th})$, where p is the number of sub-domains. On the other hand, a flat route discovery will have the complexity of $O(\eta_{th}^p)$ for the same set of routes.

5. Selective Routing

As proved in Lemma 3, the hierarchical scheme is able to reduce the complexity of the route discovery process. However, it does not reduce the overall number of routes per task. In order to reduce the complexity of the Q-RAM optimization, we must also limit the number of routes per task.

The route discovery process employed in our scheme is developed in three phases, starting from generating the exhaustive lists of routes for each task to a smart discovery of a fewer routes, with the aim of improving on the execution time without incurring any significant loss in overall utility.

5.1. Broadcast Routing

Broadcast routing is the basic approach that uses flooding from the source across the network to determine *all* possible routes to the destination. It assumes that each node only knows its neighbors. This process can potentially yield an exponentially large number of routes, and can therefore become intractable as the size of the network increases.

5.2. Smart Route Discovery

Instead of choosing all possible routes between a source and a destination, we would like to select only a few *best* or least-cost routes. We use a metric called *Route Count Threshold.*

Definition 4 (Route Count Threshold). *The route count threshold is defined as the maximum number of choices of routes for a particular source-destination pair.*

We denote this limit by η_{th}. We assume that the number of hops is the measure of the cost of a route. Using this principle, for $\eta_{th} = 1$, the only route between the source and the destination is the shortest one. In our routing scheme that we call "Smart Route Discovery", we use a modified version of the Bellman-Ford algorithm within each sub-domain of a network, where we determine η_{th} shortest routes for each source-destination pair.

5.3. Route Caching

In a distance vector routing algorithm a router learns routes from neighboring routers' perspectives and then ad-

vertises the routes from its own perspective. We implement a *reactive* distance vector routing protocol in our simulation.

According to this protocol, each node (router) is initialized with the routes of its next hop neighbors. The algorithm discovers routes of a task starting from its source. Once a route is established, each node across the route adds the entry to its routing table. The existing routing table, in turn, is exploited in route discovery. During this process, at any intermediate node, we sort the neighboring vertices in increasing order of the minimum cost of routing to the destination based on their routing tables, and reject the neighbors with more expensive routing in their tables once the number of routes reaches the limit η_{th}. This algorithm can provide a potentially sub-optimal route compared to the exhaustive discovery of the best routes. Therefore, we would like to use this routing information to assist in this step only after we finish discovering routes for a sufficient number of tasks. We define a parameter called *Task Count Threshold* T_{th}, which should be *sufficiently* large so that the optimality of the solution does not reduce significantly.

Definition 5 (Task Count Threshold). *The task count threshold is defined as the number of tasks whose routes are determined by exhaustive search using only the next-hop routing information for each node.*

5.4. QoS Optimization in Large Networks

So far, we have discussed a single centralized optimization scheme that distributes bandwidth among tasks. In a large network, a centralized scheme is likely to be infeasible. In addition, it may not scale well with a very large number of tasks. In the next section, we will describe a hierarchical QoS optimization technique that exploits the inherent hierarchy of the network. It can also be distributed across the entire system, thus making the QoS optimization feasible and scalable for a large network using a large number of tasks or flows.

6. Hierarchical QoS Optimization

In this section, we present H-Q-RAM for networks that utilizes the hierarchical architecture of networks [17]. We confine our discussion to only 2 levels of hierarchy for ease of presentation. The process is divided into two major steps. They are: (1) hierarchical concave majorant operation, and (2) distributed resource allocation. The process is described in detail in the following sections.

6.1. Hierarchical Concave Majorant Operation

This process is divided into two steps. First, we generate separate profiles for each task in each of the sub-domains containing its sub-routes. Second, we combine information from each sub-domain and update the set-points.

6.1.1. Creation of Multiple Profiles At the lowest level for each sub-graph, we obtain the set of tasks whose routes include the sub-graph. Next, we generate *local* set-points

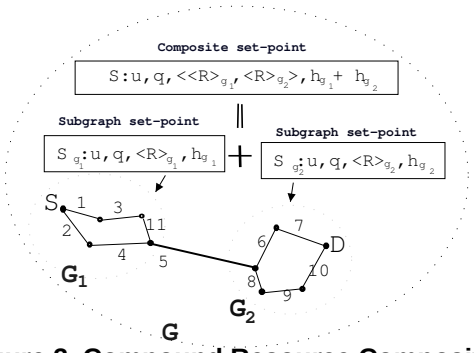

Figure 3. Compound Resource Composition

$S_i = B_i \times D_{i_g} \times P_{i_g}$ for these tasks, where P_{i_g} is the set of sub-routes inside the subgraph g and D_{i_g} is the delay assigned for the route inside subgraph g. As mentioned before, a set-point consists of a utility value, a corresponding QoS level and a resource vector specifying the route inside the subgraph and the bandwidth requirement of the links of that route. Thus each task has distinct profiles within each subgraph.

Next, we evaluate the compound resources for set-points. Using compound resource values, we prune the list of set-points and discard the ones that are "inefficient". A set-point is called inefficient if it has a larger compound resource value than another point at the same utility level. In other words, if we have multiple set-points for a particular value of utility, we keep the one that has the smallest compound resource value and discard the rest. If there is more than one set-point with the same minimum compound resource value at a utility level, we keep all of those points as co-located set-points [6].

6.1.2. Creation of Composite Profiles We next merge the profiles of multiple subgraphs or sub-domains into a single profile for each task. First, we choose a single set-point for each utility value from each subgraph for each parent route, and then *combine* the compound resource values of all subgraphs. Since all the resources in this case are considered to be of identical type (as network links), the compound resource of the global set-point of a task spanning two subgraphs g_1 and g_2 is given by:

$$h_{comp} = h_{g_1} + h_{g_2}, \tag{8}$$

where h_{g_1} and h_{g_2} are the compound resource values of the task(or flow) at its particular quality setting in the two subdomains g_1 and g_2. The generation of a composite set-point is illustrated in Figure 3, where the local set-points of the subgraphs are assumed to be $(S_{g_1} : u, q, < R >_{g_1}, h_{g_1})$ and $(S_{g_2} : u, q, < R >_{g_2}, h_{g_2})$ for a particular value of utility u and quality level q.

We determine the concave majorant of these global set-points after that. Next, we replace the compound resource values of the local set-points in each sub-domain by the cor-

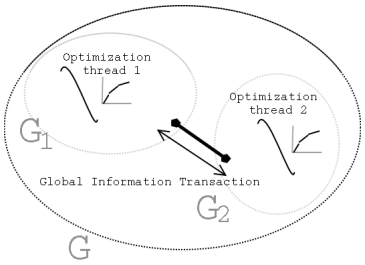

Figure 4. Distributed QoS Optimization

responding composite compound resource values. For example, as shown in Figure 3, the set-points for a task in subgraphs g_1 and g_2 are changed from $(S_{g_1} : u, q, < R >_{g_1}, h_{g_1})$ and $(S_{g_2} : u, q, < R >_{g_2}, h_{g_2})$ to $(S_{g_1} : u, q, < R >_{g_1}, h_{g_1} + h_{g_2})$ and $(S_{g_2} : u, q, < R >_{g_2}, h_{g_1} + h_{g_2})$ respectively. In addition, since the concave majorant operation eliminates set-points, a few global set-points may be discarded. In that case, we also discard the corresponding local set-points in the subgraphs.

Finally, we merge all the local set-points of tasks in each sub-domain to create lists of set-points called *slope lists*[6], which are going to be traversed for resource allocation purposes. The set-points in the slope list are ordered by increasing slope or marginal utility values.

for *Each sub-domain in the network* **do**
  for *Each task in the sub-domain* **do**
    Determine set-points $Q_i = B_i \times D_i \times P_g(i)$;
    //$P_g(i)$ = number of sub-routes for task τ_i in the domain;
  end
end
for *Each task in the entire network* **do**
  Generate global set-points by combining compound resource at each utility level;
  Perform concave majorant on global set-points;
end
for *Each sub-domain in the network* **do**
  for *Each task in the sub-domain* **do**
    Discard the set-points whose global counter-part has been eliminated by concave majorant operation;
  end
  Merge the remaining set-points of all tasks in the sub-domain in a single list;
end
for *Each sub-domain in the network* **do**
  Execute transaction-based resource allocation as described in Figure 15;
end

Algorithm 2: Hierarchical Distributed QoS Optimization

6.2. Transaction-based Resource Allocation

We perform concurrent resource allocation within each sub-domain. Thus, the entire global resource allocation problem is partitioned into multiple sub-problems within each subgraph, similar to the situation in [6]. However, the sub-problems are not completely independent of each other in this case, since some tasks may be present in more than one sub-problem. Such tasks must be assigned the resources to achieve the *same* utility value (or quality setting) in all the sub-problems that they are present in. This requires coordination between these sub-problems, since a resource allocation in one sub-domain may be infeasible in another sub-domain. In this context, we define three parameters.

Definition 6 (Local Task). *A task is called a local task if its source and destination nodes are in the same sub-domain.*

Definition 7 (Global Task). *A task is called a global task if its source and destination nodes are in different sub-domains.*

Definition 8 (Locality of Tasks). *The locality is the fraction of tasks that are local,*

6.2.1. Distributed Negotiation The resource allocator in each sub-domain sequentially goes through its slope list. If it finds the set-point in the list belonging to a local task, it determines its feasibility of allocation locally, and accepts or rejects it based on the availability of local resources. Hence it works independently for local tasks assuming that the best route for a local task Is available within the sub-domain it belongs to.

When the allocator comes across a set-point of a global task that needs to have a route spanning multiple subgraphs, it does the following. First, it checks if the corresponding global set-point has already been rejected. It happens when another sub-domain that is included in the parent route of the task fails to allocate the corresponding local set-point. In that case, the current allocator also discards the set-point and moves on. Otherwise, it marks the set-point as allocable and waits until every other sub-domain along the route decides the allocations of their corresponding set-points. During this time, it goes to sleep and and wakes up only when all other sub-domains make their decisions. Upon waking up, it checks if the allocation has been successful. The allocation becomes successful when all sub-domains are able to allocate their corresponding local set-points that complete the route with a specific utility value. The allocation is unsuccessful if one of the sub-domains fails. Upon a successful allocation, it finalizes the local allocation. Otherwise, it rejects the initial tentative allocation. Next, it proceeds further to complete the operation of QoS-based resource allocation.

6.2.2. Deadlock Avoidance in Negotiation Since allocators negotiate the allocation for set-points belonging to global tasks, it is important to ensure that a deadlock never happens. Since an allocator follows the slope list that is ordered in the increasing marginal utility[3] values, it is feasible to have the same marginal utility values for multiple set-points belonging to different tasks or flows or for different routes of the same task. In that case, we must implement

3 The marginal utility of a task is defined as the ratio of the difference between the utility values and the compound resource values between two successive set-points of different utility values.

an ordering mechanism of set-points to avoid any deadlock.

We implement two levels of ordering to avoid the deadlock. First, we assign a global number to each flow or task in the entire network. This global number can obtained as a combination of IP addresses of the source and the destination nodes, and the corresponding port numbers.

Second, we also assign a global number to each "Parent Route" within a flow. Using these numbers, we resolve the contention in the slope list when multiple set-points have the same marginal utility value. First, we order them in the increasing order of their global flow IDs. Next, for multiple co-located set-points of the same flow, we order them in the increasing order of their Parent Route IDs. For the co-located points of the same Parent Route of the same task, we do not require any ordering since their selections are independent in sub-domains, as proved in Lemma 2. The allocation process is illustrated in Figure 4 and is detailed by a flow-chart in Figure 15 the appendix.

6.3. Complexity of Network QoS Optimization

In this section, we compare the complexities of the Q-RAM and the H-Q-RAM optimization.

6.3.1. Q-RAM Complexity Suppose there are n tasks in the entire network. Using the same notation as before, let us assume that $|Q_m|$ denotes the maximum number of QoS settings, $\eta_{th} = max_{i=1}^{n}|P_G(i)|$. This definition yields the the maximum number of set-points $L = |Q_m||eta_{th}|$. Hence, the complexity of the concave majorant operation is $O(n|Q_m|\log|Q_m|)$, and the complexity of the merging operation is $O(n|Q_m||eta_{th}|\log(n))$.

Since the complexity of the Q-RAM optimization is the sum of the complexities of the concave majorant and the merging operation, we have the total complexity as $O(n|Q_m|(\log|Q_m| + |eta_{th}|\log(n)))$

6.3.2. H-Q-RAM Complexity For H-Q-RAM, initial local set-point pruning has $O(ln_l|Q_m|\eta_{th})$ complexity per sub-domain, where l equals the number of sub-domains and n_l equals the maximum number of tasks per sub-domain. Unlike the Q-RAM optimization, η_{th} denotes the upper limit on the number of routes inside each sub-domain for a task.

Next, we have the concave majorant operation that has the *global* complexity of $O(n|Q_m|\log(|Q_m|))$. The second pruning operation after the concave majorant also has the same complexity $O(ln_l|Q_m|\eta_{th})$.

The merging operation requires $O(ln_l|Q_m|\eta_{th}\log(n_l))$ steps, and the distributed transaction requires a maximum of $O(n_l\eta_{th}|Q_m|)$ steps per sub-domain.

We can now express the generic complexity expression for H-Q-RAM, namely: $O(ln_l|Q_m|\eta_{th}) + O(n|Q_m|\log(|Q_m|))+O(ln_l|Q_m|\eta_{th})+O(ln_l|Q_m|\eta_{th}\log(n_l))+ O(n_l\eta_{th}|Q_m|) = O(n|Q_m|\log(|Q_m|)+O(n|Q_m|(\log|Q_m|+ l\frac{n_l}{n}\eta_{th}\log(n_l)))$.

From the expression, in the worst case, when every task has a profile in every sub-domain, we have $n_l = n$. Then, the complexity of H-Q-RAM is *higher* than that of Q-RAM. In the best case, which corresponds to the case when every flow is a local task that does not span sub-domains, we have $n_l = n/l$, which is better than that of Q-RAM. However, in a very large network (the size of the Internet), it is very unlikely that a task traverses across all sub-domains. Therefore, H-Q-RAM performs better than Q-RAM for practical cases. Since H-Q-RAM computations can be distributed (one node per sub-domain), we can further reduce the complexity to $O(\frac{n}{l}|Q_m|(\log|Q_m|+\frac{n_l}{n}\eta_{th}\log(n_l)))$. Thus, H-Q-RAM can scale well with large networks.

7. Experimental Evaluation

Our experimental evaluation is intended to quantify the performance of H-Q-RAM and Q-RAM in terms of the trade-off between optimality and scalability. We focus on measuring two main parameters:

- the global utility obtained by the optimization, and
- the total execution time of the algorithm.

First, we investigate the efficiency of our enhancements in route discovery. We determine how a selective set of routes obtained through our smart route discovery process can eliminate the necessity of selecting a large number of routes for the optimization purposes. We also investigate the performance of the optimization when we vary the parameter T_{th}. Second, we compare the performance of H-Q-RAM optimization with respect to Q-RAM optimization.

7.1. Experimental Configuration

QoS dimensions	Bandwidth and delay
Length of bandwidth dimension	$random(1,4)$
Length of delay dimension	1
Minimum Bandwidth(B_{min})	$min((Rayleigh\ Distr.: \mu = 152\ Kbps)$, $8000.0\ Kbps)$
Bandwidth Increment	$0.3B_{min}$
Maximum Delay	$random(16,20)\ hops$
Utilities for QoS dimension ($u(q)$)	(0.5,0.7,0.8)

Table 1. Settings of Tasks

In order to validate our technique, we generate network topologies using BRITE [13] a topology generation tool. The bandwidth distribution of the network links is presented in Table 2.

The specifications of the tasks are presented in Table 1. As seen from the table, the minimum bandwidth is ran-

Network topology generator	BRITE [13]
Intra-domain link bandwidth	$10.0\ Mbps$
Inter-domain link bandwidth	$10000.0\ Mbps$

Table 2. Settings of Networks

domly chosen following a Rayleigh distribution with $\mu = 152\ Kbps$. This distribution ensures a positive value for the minimum bandwidth of any task. For simplicity, we choose a single value of delay, which is expressed by a certain maximum number of hops for a route. The source and the destination nodes of a task are chosen randomly across the entire network. The experiments are performed on a $2.0\ GHz$ Pentium IV processor with $768\ MB$ of memory.

7.2. Performance Evaluation of Selective Routing

In this section, we evaluate the performance of the selective routing algorithms.

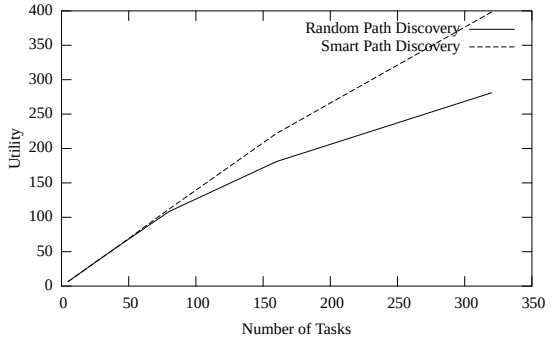

Figure 5. Utility of Smart and Random Route Discoveries

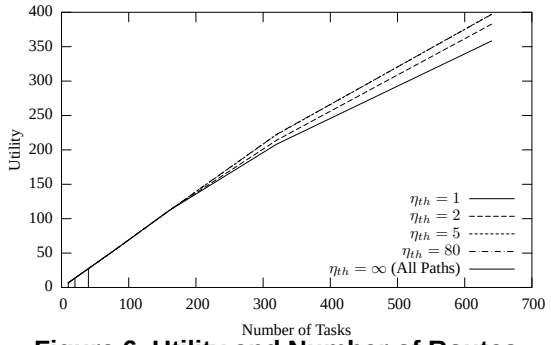

Figure 6. Utility and Number of Routes

7.2.1. Results on Smart Route Selection In this experiment, we demonstrate the effectiveness of smart route selection as described in Section 5.2.

First, we compare the smart route discovery algorithm with the random route discovery algorithm, where we randomly select η_{th} routes out of all possible routes. We vary the number the number of tasks in the system in geometric progression as $N = 10, 20, 40, \ldots, 640$. We plot the accrued utility against the number of tasks for $\eta_{th} = 5$ under both schemes in Figure 5. The results show that a random route selection scheme yields a much lower utility (29.5% for $N = 320$) compared to the smart route selection.

Next, we compare smart route selection for different values of ηth. In this case, we use 5 values of η_{th} as $[1, 2, 5, 80, \infty]$. The value ∞ signifies that all possible routes are chosen for each source-destination pair. The plots of utility against the number of tasks are shown in Figure 6. The "$\eta_{th} = \infty$" case is shown by the bar graph instead of a line.

From the bar graph, we observe that we do not have any data beyond $N = 40$ for $\eta_{th} = \infty$. This is because for $N \geq 80$, the route discovery and the optimization processes become intractable. This is further confirmed by its steep rise in execution time as shown in Figure 7.

On average, the utility increases as η_{th} increases since it provides more alternative routes for each task. However, the difference between utilities at $\eta_{th} = 5$ and $\eta_{th} = \infty$ is statistically insignificant($< 0.09\%$), whereas the reduction in execution time for $\eta_{th} = 5$ is 93.6% (or, 15.6 times). Overall, we observe a 99.997% (or, 38239.4 times) reduction in execution time for $\eta_{th} = 5$ relative to $\eta_{th} = \infty$ when the number of tasks is 40. Even for $\eta_{th} = 2$, the reduction in utility is only 3.57% relative to $\eta_{th} = 80$ for 640 tasks, with a run-time reduction of 96.9%.

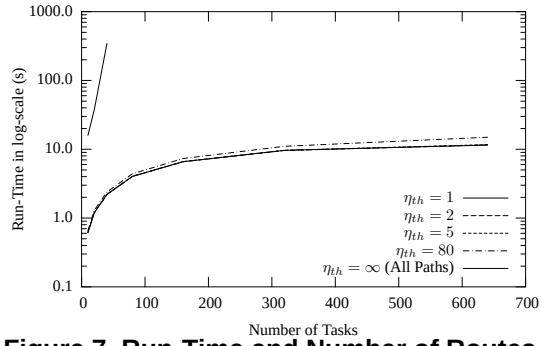

Figure 7. Run-Time and Number of Routes

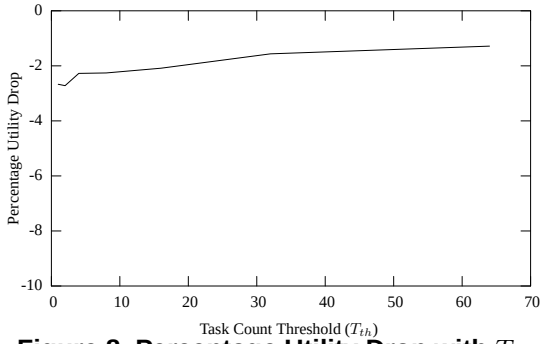

Figure 8. Percentage Utility Drop with T_{th}

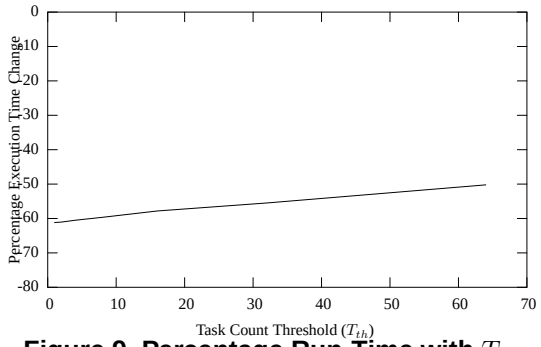

Figure 9. Percentage Run-Time with T_{th}

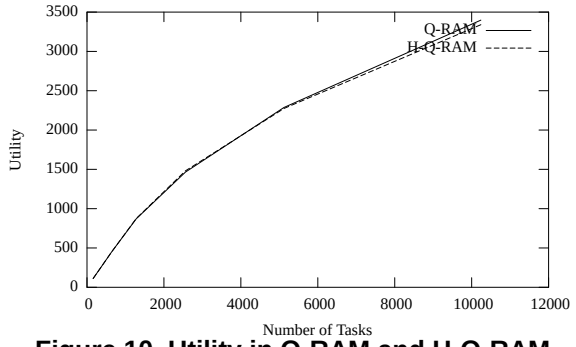

Figure 10. Utility in Q-RAM and H-Q-RAM

Net-ID	Sub-domains	Nodes	Links
1	5	100	207
2	8	160	334
3	15	450	930
4	20	600	1240

Table 3. Specifications of Networks

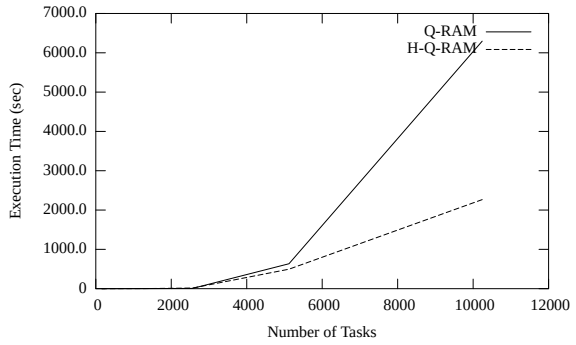

Figure 11. Run-Time in a Host for Q-RAM and H-Q-RAM

7.2.2. Results on Route Caching This experiment demonstrates how caching route information helps in reducing the execution time of the optimization. In this case, we fix the number of tasks N to 640 and vary the parameter Task Count Threshold T_{th}. Figure 8 shows the percentage drop in utility for different values of T_{th} compared to the same under no Route Caching, or $T_{th} = \infty$. The value of η_{th} is kept constant at 5.

We observe that even for $T_{th} = 1$, for example, we start exploiting route discovery information right after the first task's routes have been determined. The percentage loss of utility is less than 3%. On the other hand, we also observe a huge drop in execution time ($> 60\%$) as shown in Figure 9. Using the route caching technique, the route discovery time per task will reduce with time as nodes keep adding more entries to their routing tables. In other words, we can claim that in a dynamic scenario, in steady state, the optimization time *dominates* the route discovery time.

7.3. Performance Evaluation of Hierarchical Optimization

In this section, we evaluate the performance of Hierarchical QoS optimization. We use 2 levels of hierarchy for our experimental evaluation. We use the same specifications of tasks as mentioned in Table 1. In order to validate the usefulness of H-Q-RAM, we use larger networks, consisting of 5, 8, 15 and 20 sub-domains respectively. Their specifications are presented in Table 3, and their bandwidth distributions as specified in Table 2. For these large networks, we use $\eta_{th} = 2$, and $T_{th} = 1$, since these settings have provided reasonably good utility values($< 5\%$) with great reductions execution time for smaller networks.

In the first experiment, we use Network 3 from Table 3. In this case, we vary the number of tasks for optimization between 100 and 10240 in a geometric progression with a power of 2. Figure 10 shows the variation of utility between Q-RAM and H-Q-RAM against the number of tasks. Figure 11 shows the variation of execution time against the number of tasks.

We observe that H-Q-RAM reduces the optimization time for $10,240$ tasks by 64% while incurring a utility reduction of less than 2% utility than Q-RAM. From Figure 11, we also observe that the difference between Q-RAM and H-Q-RAM increases further with the increase in the number of tasks.

Implementation Considerations: As can be seen from Figure 11, the execution time of the optimization increases exponentially for a large number of tasks to be deployed in larger networks. This is because the simulation becomes memory-intensive under this situation and hence many page faults and swapping operations cause the non-linear (exponential) increase in the execution times. Consequently, it becomes difficult to *simulate* the hierarchical optimization of a very large network in a single host, as the memory requirement for the optimization process also increase. This effectively suggests the necessity of studying the performance improvement of distributed transaction-based optimization using H-Q-RAM. The execution time for H-Q-RAM will be reduced further if the optimization is distributed over multiple hosts. This will be the only option available, since run-

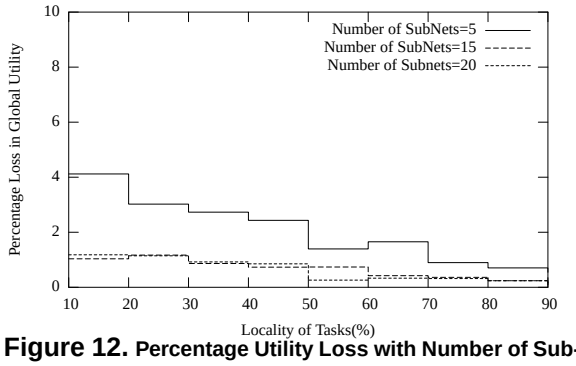

Figure 12. Percentage Utility Loss with Number of Sub-domains

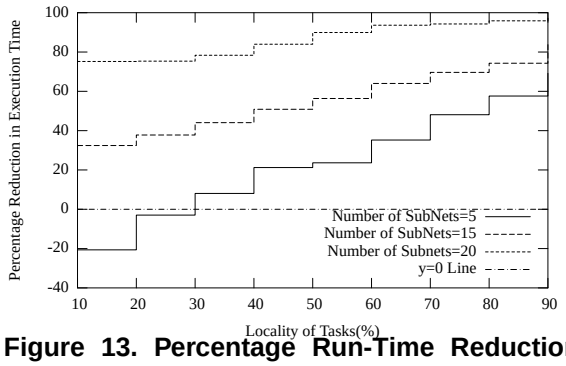

Figure 13. Percentage Run-Time Reduction with Number of Sub-domains

ning the Q-RAM optimization for all the tasks in a single host becomes intractable.

Next, we measure the performance of H-Q-RAM relative to the *locality* of tasks in different sub-domains. From our complexity analysis, we know that H-Q-RAM performs best when the source and the destination of a task are confined within a single domain, which in turn also eliminates transactions between sub-domains during the optimization step. In this experiment, we keep the number of tasks constant at 6400 and vary the locality of tasks between 0% and 96% and measure the performance of Q-RAM and H-Q-RAM. The results are taken for Networks 1, 3 and 4 from Table 3.

Figure 12 plots the percentage loss in utility under H-Q-RAM, which does not exceed 4.5%. In addition, the loss drops with the increase in the locality of the task and with the increase in the size of the network.

Figure 13 plots the percentage gain in execution time under H-Q-RAM. As seen from the figure, H-Q-RAM actually has 20% higher execution time under 0% task locality for the smallest network (Network 1 with 5 sub-domains). However, it increases with the size of the network as well as with the locality of the tasks. Moreover, the rate of increase in percentage gain decreases with the increase in the size of the network. In other words, for a very large network, H-Q-RAM performs better than Q-RAM and the significance of locality on this performance decreases.

The above experiment shows that H-Q-RAM provides a significant gain in performance when (a) the size of the network is large, and (b) the locality of the tasks is high. These results are in agreement the complexity analysis of H-Q-RAM.

In Figure 14, we also plot the number of transactions against the locality of the tasks. As expected, the number of transactions decreases with the increase in task locality. However, we observe a larger number of transactions with the increase in the size of the network. This affects the absolute execution time of H-Q-RAM in our simulation due to a large number of switching among optimization threads and the consequent page faults.

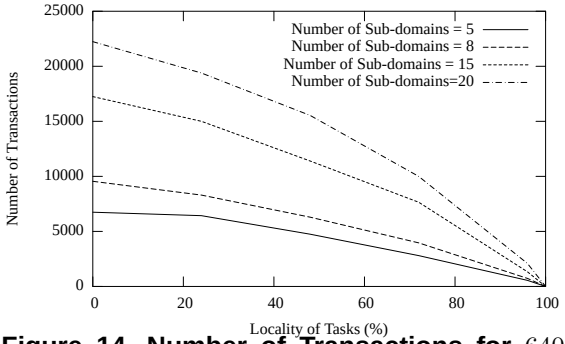

Figure 14. Number of Transactions for 6400 Tasks with the Number of Sub-domains

Based on the above results, we conclude that H-Q-RAM performs well for large networks compared to Q-RAM, which makes it feasible to employ QoS-based optimization in large networked environments. However, we also observe that the number of transactions also increases with the increase in the size of the network. Therefore, we would like to reduce the number of transactions for future implementations.

8. Concluding Remarks

In this paper, we have discussed a resource allocation scheme for a networked system based on Q-RAM. First, we proposed several pruning algorithms for smart route selections that makes the basic optimization more scalable without any significant loss in the optimality of the solution. Our main goal was to analyze the trade-off between optimality and the execution time of our QoS optimization. Although the specific values may vary depending on the topology, restricting the maximum number of routes only to 2 reduces the optimality only by 5%. In addition, exploiting the cached route information across the network becomes more useful as the size of the network increases.

Next, we presented a transaction-based hierarchical scheme (H-Q-RAM) that can make the problem more scalable by exploiting the presence of hierarchy in networks. The performance of H-Q-RAM improves with the increase in the size of the network and the locality of the tasks. We also observed that the simulation is memory intensive, and it becomes increasingly expensive in a single host with the increase in the size of the network. Therefore, a centralized scheme becomes infeasible for a network with the size of the Internet. Since H-Q-RAM can be executed concurrently on multiple machines using distributed transactions, it can be run in parallel to address large networks. In addition, we would also like to reduce the number of transactions which increases with the size of the network. This can be done if we can aggregate multiple tasks into a few "super-tasks" and perform transactions for "super-tasks". Hence our future work will investigate efficient methods of task aggregation.

References

[1] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[2] S. Chen and K. Nahrstedt. Distributed quality-of-service routing in high-speed networks based on selective probing. In *IEEE Annual Conference on Local Area Networks (LCN)*, pages 80–89, Oct 1998.

[3] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79, 1998.

[4] R. Cherukuri, D. Dykeman(eds.), and M. Gouguen(chair). Pnni draft specification, November 1995.

[5] Sourav Ghosh and Ragunathan Rajkumar. Practical management of end-to-end network bandwidth reservation. In *Proc. of Conference on Software in Telecommunications and Computer Networks (SOFTCOM)*, October 1999.

[6] Sourav Ghosh, Ragunathan (Raj) Rajkumar, Jeffery Hansen, and John Lehoczky. Scalable resource allocation for multi-processor qos optimization. In *23rd IEEE International Conference on Distributed Computing Systems (ICDCS 2003)*, May 2003.

[7] Roch Guerin and Ariel Orda. Qos-based routing in networks with inaccurate information: Theory and algorithms. *IEEE Transactions on Networking*, 1(3), June 1999.

[8] Jeffery P. Hansen, Haifeng Zhu, John Lehoczky, and Ragunathan Rajkumar. Quantized edf scheduling in a stochastic environment. In *Proc. of 10th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2002.

[9] Chen Lee. *On Quality of Service Management*. PhD thesis, Carnegie Mellon University, August 1999.

[10] King-Shan Lui, Klara Nahrstedt, and Shigang Chen. Hierarchical qos routing in delay-bandwidth sensitive networks. In *IEEE Local Computer Networks (LCN 2000)*, pages 579–588, 2000.

[11] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *IEEE International Conference on Network Protocols*, October 1997.

[12] R. Mangharam, M. Demirhan, R. Rajkumar, and D. Raychaudhuri. Size matters: Size-based scheduling for mpeg-4 over wireless channels. In *SPIE Conference on Multimedia Computing and Networking (MMCN)*, pages 110–122, 2004.

[13] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2001)*, August 2001.

[14] Abhay Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single node case. *IEEE/ACM Transactions on Networking*, pages 344–357, June 1993.

[15] Charles L. Hedrick Rutgers. Cisco white paper: An introduction to igrp.

[16] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of SIGCOMM'98*, 1998.

[17] L. Tauro, C. Palmer, G. Siganos, and M. Faloutsos. A simple conceptual model for the internet topology. In *6th IEEE Global Internet Symposium*, 2001.

[18] Lixia Zhang, Stephen Deering, and Deborah Estrin. RSVP: A new resource ReSerVation protocol. *IEEE network*, 7(5):8–18, September 1993.

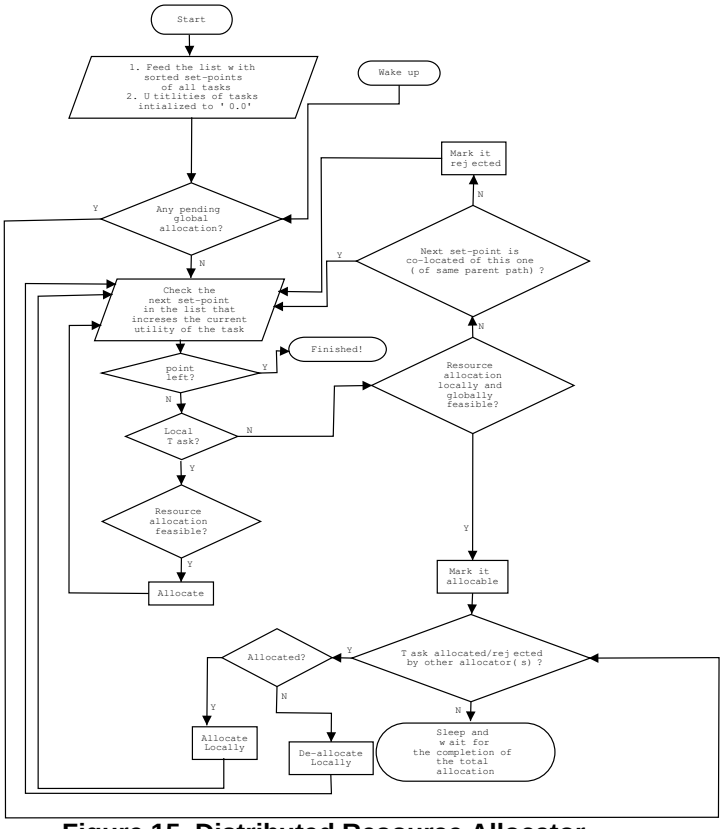

Figure 15. Distributed Resource Allocator