The Optimal Admission & Adaptation of Service Level Agreements In Packet Networks :

Applying the Utility Model

By

Robert Watson

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical & Computer Engineering

University of Victoria

We accept this thesis as conforming
to the required standard

Prof. Dr. Eric G. Manning, P.Eng., ISP, Supervisor (Departments of ECE & C S)

Dr Kin F. Li, P.Eng., Departmental Member (Department of ECE)

Dr. G. Shoja, P.Eng., Outside Member (Department of Computer Science)

Mr. Michael Reece, External Member (Nortel Networks Inc.)

University of Victoria

Supervisor: Prof. Dr. Eric G. Manning, P.Eng., ISP

ABSTRACT

Khan considered the problem of optimal allocation of the resources of a single server, while meeting the Quality of Service (QoS) requirements of users' multimedia sessions. He showed how the problem could be mapped onto a variant of the combinatorial Knapsack Problem, with server utility (e.g. revenue) as the quantity to be optimized and with the user QoS requirements expressed as constraints on the resource allocation. He gave both optimal (algorithmic) and fast but suboptimal (heuristic) methods to solve the resulting Multidimensional Multiconstraint Knapsack Problem (MMKP).

In this thesis we apply Khan's general approach to the problem of optimal allocation of the resources of a packet network to requests for network service (called Service Level Agreements or SLAs) to support multimedia sessions - subject to session QoS constraints, expressed as constraints on the data rates and latencies experienced by sessions.

New problems arising from a number of interesting differences between Khan's server problem and our packet network problem are articulated and dealt with. The resulting heuristic solution technique has been implemented as a Java-based program for the optimal admission and QoS adaptation of Service Level Agreements in a packet network. Preliminary performance data and suggestions for further work are given.

[redacted]

Prof. Dr. Eric G. Manning, P.Eng., Supervisor (Departments of ECE & CS)

[redacted]

Dr Kin F. Li, P.Eng., Departmental Member (Department of ECE)

[redacted]

Dr. G. Shoja, P.Eng., Outside Member (Department of Computer Science)

[redacted]

Mr. Michael Reece, External Member (Nortel Networks Inc.)

Table of Contents

List of Figures

Acknowledgments

Many thanks to Dr. Eric Manning, my supervisor, for insight, encouragement, and support throughout.

Thank you also to my committee members, Dr. Shoja, Dr. Li, and Mr. Reece for their questions, suggestions, and help, to Dr. Khan, whose research prompted this work, and to the members of the PANDA group.

1. Introduction

In this Chapter we describe the problem to be solved and mention a few key related ideas.

1.1 The Problem

1.1.1 Multimedia traffic

Over the last ten years, there has been an incredible increase in the amount of traffic on both the Internet and on various internets. In addition, an increasing amount of this traffic is multimedia, and therefore comes with specific performance or Quality of Service (QoS) requirements, high bandwidth and low latency being chief among them. Examples of such applications are: streaming audio and video generated by films and television programming, video conferencing, large file transfers (such as site backups, music and video), and games.

1.1.2 Quality of Service (QoS)

In addition to these challenges, an Internet Service Provider or ISP [1] must deal with rapidly changing patterns of use within a network. The ISP must be able to react to such changes in traffic quickly, in order to minimize his costs while at the same time being able to honour quality of service guarantees made to his clients. He must be able to avoid the overbooking of his resources, which may break QoS guarantees, but must not purchase excessive amounts of these resources from the underlying *facilities - based carrier* who owns them, or he may go bankrupt in today's highly competitive marketplace.

1.1.3 Binding Time

Current IP-datagram based networks bind network resources to traffic flows too late to allow QoS guarantees to be made and honoured. Packets are treated independently, and link capacity and switch buffers are only bound (allocated) to a packet when it arrives at the link or switch. Hence it is impossible to predict or guarantee in advance the level of service which the packets of a traffic flow will experience. Finally, there is no fixed, pre-planned route through the network, which is obviously a prerequisite to the binding of resources to a flow.

---

[1] By ISP we mean a network Service Provider (such as UUNet) as opposed to a local ISP.

In order to achieve QoS guarantees, packet networks can use some form of virtual circuit (for example, ATM virtual channels and virtual paths) [STAL97]. It then becomes possible, by adopting a fixed route for all of the packets of a flow (virtual circuit) to identify and thus pre-allocate resources (link timeslots, switch buffers) to the flow, or more accurately to the virtual circuit, and thus guarantee a level of QoS.

1.1.4 The Problem

Our problem is to select, from a set of Service Level Agreements (SLAs – the agreement for service between an ISP and its customer) requesting admission to a packet net, a subset which will result in maximum utility (revenue) for the network operator, while fully respecting the allocations of resources necessary to honour QoS guarantees, made both to the newly admitted SLAs and to all other currently active SLAs.

1.2 Related concepts

1.2.1 Network Reconfiguration

Traditionally, network ISPs reconfigured their networks about once or twice a year. In addition the SLA has been inflexible.

Now, however, ISPs need to reconfigure daily or even hourly, in order to meet the tighter QoS requirements of customers while minimizing their costs. And, customers for multimedia services such as videoconferencing or movies would like more flexible SLAs, with features such as the ability to

- make bids & adjust them if necessary (auctions – a form of resource allocation much favoured by economists for its efficiencies)
- reserve capacity for a defined time period
- ask for several QoS levels at several prices

However, one factor works in our favour. Often, a customer will book a reservation for traffic several days (or weeks) in advance, or an event that will require large amounts of resources may occur periodically . (An example of this is a scheduled videoconference held, say, every Monday morning.) In these cases we have substantial time to do admission control and can therefore undertake lengthy computations in support of it.

2. Previous Work

In this Chapter we review previous work related to our problem.

2.1 Containers and Leaky Bucket Algorithms

Perhaps the earliest work on admission control was by Donald Davies. He proposed we bound the total number of packets by the introduction of packet "containers" which circulated in the net. A customer had to obtain a container before being allowed to insert a packet [DAVI73]. A later attempt involved cells in ATM networks. The leaky-bucket or token bucket algorithm [STAL99] attempted to model circuit capacity, yet allow for bursty traffic. More recently there has been a host of heuristics that rely on simulation to demonstrate their plausibility. Prior to this thesis, to the best of our knowledge no-one has given a revenue-optimizing admission control algorithm which also guarantees all QoS constraints.

2.2 The Utility Model

The purpose of the Utility Model [KHAN98] is to manage the consumption of resources in a multimedia server, in order to maximize the revenue generated by the server (or more generally, its *utility*), while keeping the quality of the services (QoS) provided to clients at guaranteed levels. Examples of the resources to be managed include CPU time and graphics coprocessor time, memory and video bus bandwidths, and primary and secondary memory. The limited quantities available of these system resources constrain the use of the server. In traditional environments, such as a classical interactive compute server or the current Internet, we simply take on all customers (*promiscuous admission*) regardless of ability to serve them, and therefore can honour no guarantees whatsoever about QoS. The best we can achieve is some notion of fairness, where all customers receive equally bad service, or of priority, where the service provided to customers of high priority may be less terrible than the service provided to the others.

In a multimedia environment with paying customers and legally enforceable SLAs, however, we **must** observe QoS requirements, and hence we must guarantee the availability of the necessary resources to each customer. Hence, we need some notion of *admission control.*

A *session* is the entity which requires the resources, at the level of granularity at which the Utility Model, or UM, operates. Each session has a number of possible QoS levels . We use the generic terms *Bronze, Silver, and Gold* without loss of generality.

Each QoS level for each session then implies a set of resource requirements needed for that QoS level, and a value, or *utility*, assigned to each such set. The utility can be

viewed as the amount of money that the session's owner – the customer - is willing to contribute to server revenue if it is granted admission at that QoS level. We note that, by convention,

QoS level 0 ($V_0(S)$)

means no QoS (and hence no resource usage). The session is rejected.

For each session i, a QoS level Q_l must be selected. This level then implies the *session utility* $u_l(Q_l)$, and the *session resource usage* $r(Q_l)$.

In order to guarantee service at the level of quality Q_i selected, we must *bind* the necessary resources to the session before it begins, and for the duration of its existence. The *system utility, U,* is some suitable function, often simply the arithmetic sum, of all session utilities. Figure 2-1 shows the relations between system and session utilities, and between resource mappings and constraints, established via the choice of session QoS..

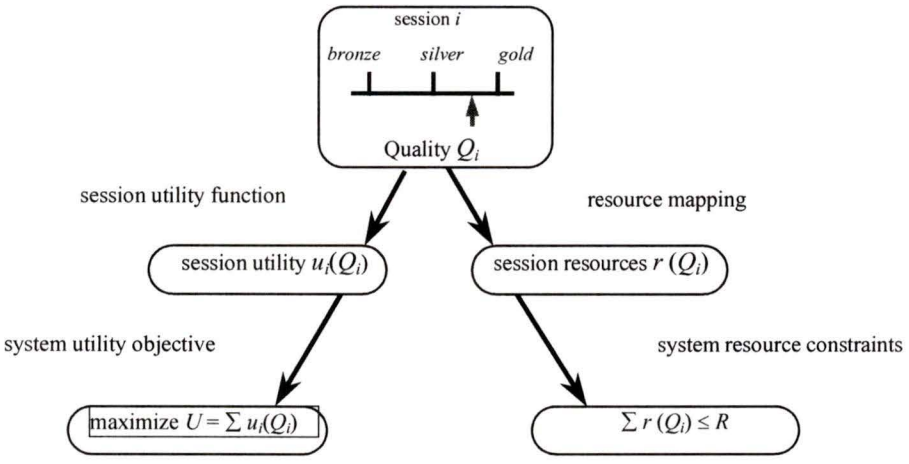


Figure 2-1. Relations among qualities, utility and resources

Our goal, then, is to maximize the revenue U while respecting the system resource constraints R. This problem is called the Adaptive Multimedia Problem, or AMP.

2.2.1 Knapsack Problems

A simple *0-1 knapsack* problem can be visualized as shown in Figure 2-1: Given a knapsack with capacity C, and a set of stones S, each with volume $v(S)$ and value $V(S)$, choose those stones S' from S such that

***v(S') < C* and *V(S')* is maximized:**

that is, pick the most valuable set of stones which will fit in the knapsack. In the example, the value V is represented as a *utility*, u, and the volume v as a resource constraint p.

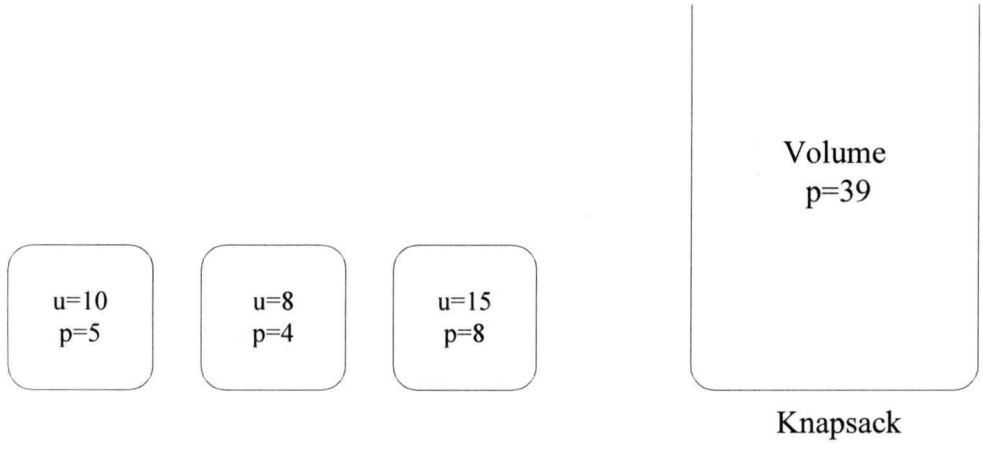


Figure 2-1. 0-1 Knapsack Problem

The Adaptive Multimedia Problem can be formulated as a Multi-choice Multi-dimension 0-1 knapsack problem (MMKP), where we have multiple sets of stones

$$S_1, S_2, S_3, \ldots, S_n.$$

and each stone has multiple volumes

$$v_1(S), v_2(S), \ldots v_3(S).$$

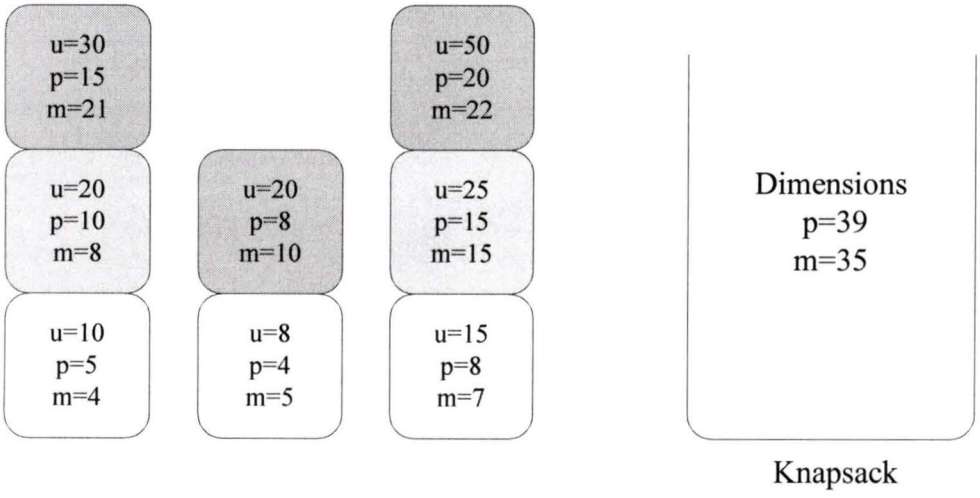
In this case, the weights represent the utilization of each resource for a given stone, and the each pile of stones represents the possible QoS, levels for the session S_n. We then wish to select one QoS level S'_m from each session, such that we maximize the revenue

$$\Sigma\, V(S'_m)$$

while ensuring that none of the resources w_n are overused; i.e.

$$\Sigma\, v_n(S'_m) < R_n \text{ for all } n.$$

Figure 2-2 illustrates this selection. The simple knapsack problem involves picking stones to pack a knapsack so to maximize the total weight of stones chosen, while respecting the knapsack constraint of total volume. For an MMKP, we generalize the stone and knapsack [CHEN98]. Each session, in this case, pile of stones, contains several QoS choices, which are the stones of a pile, and each choice (stone) specifies a set of requirements (resources) and a utility. We wish to select at most one stone from each pile (session and its QoS level) to place in the knapsack, maximizing the total utility of the sessions (stones) chosen while respecting the resource constraints of the knapsack.



Pick at most one item from each stack in order to maximize U=$\sum u$,
subject to resource constraints: $\sum p \leq 39$,
$\sum m \leq 35$

Figure 2-2. Multi-choice Multi-dimensional 0-1 knapsack problem (MMKP)

Khan presents two solutions to this problem: the branch-and-bound algorithm BBLP using linear programming for bound calculation, which achieves the optimal solution, and the heuristic HEU whose result often comes reasonably close to the optimal solution, but takes much less time. We will base our work upon this heuristic, although we will modify it considerably, so we now briefly describe it.

We assume an empty system, i.e. no previously- admitted sessions. HEU operates by first ordering the QoS levels for each candidate session according to revenue $V(S_m)$. It then selects the **least profitable** (smallest value of v) of the feasible QoS levels for each session as its initial solution. Note that this includes denial of service, if no level is feasible. This is *admission control,* and is treated here as a particular level (level 0) of QoS.

HEU then considers, in a loop, the possible *upgrades* for each session. An *upgrade* is a move to a higher level of QoS for a session. HEU calculates the net change in *aggregate resource consumption*[2] for each session's feasible upgrades. Toyoda [TOYO75] penalizes those upgrades that consume heavily used resources. If any upgrades are found to decrease the system aggregate resource consumption[3], the upgrade which yields the greatest decrease in aggregate resource consumption is selected. Otherwise, the upgrade which provides the greatest value gained per extra unit of aggregate resource

$(\Delta v / \Delta r)$

– a kind of figure of merit – is selected.

The loop then returns to considering possible upgrades for each session. The heuristic terminates when no more *feasible* upgrades can be made; that is, when all further upgrades cause us to consume more resources than we have, i.e. *violates the resource constraints.*

In the case of a multimedia server, Khan was able to provide a well-defined ordering of our QoS levels, as he only wished to consider upgrades, and not *downgrades*[4] which will result in lower revenue. Note that this ability to fully order QoS levels is important for the efficient operation of HEU. It will become a problem when we apply the UM to packet networks, for three reasons:

1. The existence of multiple paths for a flow, leading to multiple resource-sets per QoS level, and hence an inconvenient loss of uniqueness
2. a multi-dimensional QoS criterion (path bandwidth plus latency), leading to a *partial* ordering, not a *full or linear* ordering

3. our use of *net* rather than *gross* revenue as our QoS measure

For these reasons we will develop a new heuristic, based on HEU and called *Network HEU* or N-HEU, incorporating solutions to the problems sketched above.

---

[2] See section 2.2.2 Aggregate Resource Consumption

[3] While this may appear counter-intuitive, we can imagine that if the resources used by the current QoS level are in heavy use, and there exists an upgrade whose resources are in light use, then this upgrade may decrease the system aggregate resource consumption.

[4] These are defined as a move to a lower level of QoS for a session. There are two obvious methods of QoS ordering; by utility (revenue) and by resource usage; please see section 3.4.1. The Utility Model takes the former approach.

Pseudocode for HEU is given in Figure 2-3:

```
for (i = 1, ..., n) ρ[i] = 1;                    /* initialize with smallest QoS levels */
C = Σ r[i][ρ[i]];                                /* calculate resource usage C */
while (1) {
        Δr_max = 0, Δp_max = 0;
        for (i = 1, ..., n; j = ρ[i] + 1, ..., l_i) {
                if (∃k : k = 1, ..., m, C[k] - r[i][ρ[i]][k] + r[i][j][k] > R[k]) continue;
                Δr = ((r[i][ρ[i]] + r[i][j]) • C) / C;      /* calculate aggregate resource */
                if (Δr > Δr_max)                 /* upgrade with max resource saving */
                        Δr_max = Δr, i' = i, j' = j;
                if (Δr_max ≤ 0) {
                        Δp = (v[i][ρ[i]] - v[i][j]) / Δr
                        if (Δp > Δp_max)         /* upgrade with max value /aggr. resource */
                                Δp_max = Δp, i' = i, j' = j;
                }
        }
        if (Δr_max ≤ 0 and Δp_max ≤ 0) return ρ;
        C = C – r[i'][ρ[i']] + r[i'][j'];        /* update C and return */
        ρ[i'] = j';
}
```

Figure 2-3. Pseudocode for HEU

2.2.2 Aggregate Resource Consumption

Here we define *aggregate resource consumption*.

Toyoda (op cit), who calls it *effective gradient*, introduces the idea as a *penalty vector*, in order to obtain a one-dimensional resource consumption index based on the consumption of multiple limited resources. As an example, suppose we have two limited resources r_1 and r_2, and that the current consumption of these resources is

$$\mathbf{C} = (c_1, c_2) = (0.5, 0.2)$$

Given two sessions S_1 and S_2 with required resource vectors (r_1, r_2)

$$\boldsymbol{R_{S1}} = (0.1, 0.4)$$
$$\boldsymbol{R_{S2}} = (0.4, 0.1)$$

we prefer S_1 to S_2, as it requires less of r_1, which is currently in heavier use than r_2. $\boldsymbol{C}$ is used as a *penalty vector*, and we calculate the penalties as:

$$S_1: \quad \boldsymbol{R}_{S1} \bullet \boldsymbol{C} = 0.1 \bullet 0.5 + 0.4 \bullet 0.2 = 1.3$$
$$S_2: \quad \boldsymbol{R}_{S2} \bullet \boldsymbol{C} = 0.4 \bullet 0.5 + 0.1 \bullet 0.2 = 2.2$$

Geometrically, the effective gradient is the projection of the required resource vector onto the current consumption vector C. Put another way, $\boldsymbol{R} \bullet \boldsymbol{C}$ measures S's demands on resources weighted by their scarcity.

We can then normalize the effective gradient by the magnitude of $\boldsymbol{C}$ (0.54) to obtain aggregate resource consumptions for S_1 and S_2 as 2.4 and 4.1 respectively. In N-HEU, of course, we are more interested in the *change* in aggregate resource consumption when a session is upgraded; please see Section 0 below.

2.2.3 Applications of the Utility Model

Khan (op. cit.) applied the Utility Model to solve the AMP . Specifically, each session corresponds to a pile of stones and each stone to one level of QOS (Gold, Silver and Bronze) of one session. The objective is to select at most one stone from each pile so as to maximize utility without overfilling the bag.

Lei Chen [CHEN98] then applied the Utility Model to layer coded sources[5], and constructed a prototype Linux implementation, incorporating simple vidoconferencing and file-transfer applications. Since the Linux OS does not provide a resource reservation mechanism, and only provides a very simple resource monitoring mechanism through the */proc* file system call, Chen also developed a resource contention strategy (please see Section 3.5).

2.3 The Service Level Agreement

As defined in Chapter 1, a Service Level Agreement or SLA is a contract between an internet Service Provider or ISP, and a customer. It simply represents an agreement to provide a certain level of service, for a certain time, and at a given price. However, we reflect the new reality by replacing the traditional SLA with a much more flexible one, which offers, for example:

- Time-dependent service (e.g. 2Gb/s next Tuesday from 2 to 6 pm, for a videoconference)

---

[5] i.e. where a source provides a *base layer* plus some number of optional layers in order to allow for differing resource availability situations.

- Repetitive service (the above, but *every* Tuesday)
- Multiple service/price options (e.g. 1Mb/s @ $10/hour **or** 2 Mb/s @$15/hr)
- Service auctions and bidding (If a customer can't get the service that she wants at $10/hr, she may choose to raise her bid to $12).

In this thesis, the customer session is characterized by two scalars – peak data rate and maximum acceptable latency – neither of which is stochastic in nature. Probabilistic considerations can be added to the model, as a subject for further work.

The carrier, on the other hand, wants to do *admission control*, i.e. to admit the subset of the SLAs on offer at QoS levels which maximize revenue, while fully respecting all terms and conditions of both the newly-accepted and all previously-accepted SLAs (QoS guarantees)[6].

2.4 Other Related Work

Chang & Zakhor [CHAN96] have looked at costs for Variable Bit Rate (VBR) video servers. Chen & Chen [CHEN96] have studied simple heuristics for admission of sessions to a server. Davies et al [DAVI94] looked at the needs of adaptive services in mobile environments. Tokuda et al [TOKU94] studied operating system tactics for reserving CPU capacity for multimedia applications.

Kamath, Plotkin et. al. [KAMA98] have developed online competitive routing and admission control strategies for both throughput maximization and congestion minimization models, motivated by competitive analysis [BORO98] .

The throughput maximization strategy involves defining, for each link in the network, a *length* c_e which is exponential with regard to the current congestion of the link. Then, when a request for bandwidth between nodes s and t arrives, the flow is routed if there exists a path $P(s,t)$ such that the flow is profitable with respect to this length c_e. If it is not, the flow is rejected.

Similarly, the congestion-minimization model (Kamath, op. cit.) also defines the length c_e of the link as exponential in the current congestion of the link. However, unlike the throughput-maximization case, no rejections are allowed. Instead a rejection edge *rej* is created; a flow is routed along the shortest path with respect to the length c_e. The flow is

---

[6] In fact, given finite carrier resources and potentially unbounded service requests, the carrier *must* do admission control, if any credible attempt to guarantee QoS is to be made.

then considered rejected if it is routed along *rej*, or if the path selected has insufficient capacity. Otherwise, it is considered accepted.

Biddiscombe, Midwinter and Sabean [BIDD99] developed a resource control algorithm for networks to allocate resources (in this case, bandwidth) in a provably fair manner. Users express a willingness to pay (WtP) for the resources, which are then divided amongst the users in a proportion to their WtP. Proportional fairness assumes that the price per unit of any given resource is identical for all users.

Biddiscombe et. al. proposed that the utility U obtained by a user for bandwidth b at cost c be given by:

$$\mathbf{U = c \ln b}$$

and presented an algorithm to maximize the aggregate utility of all users' sessions.

Following Biddiscombe et. al. , we wish to maximize the aggregate utility of all users' sessions. For this, we need two things:

1] a mapping of QoS levels (as perceived by the user) to resource requirements and utility, *and*

2] an optimizing method for selecting among these mappings.

The purpose of the Utility Model is to solve the second problem; it assumes the existence of quality-resource mappings.

Finally, Kirkby & Kadengal [KIRK99] have proposed a link congestion measure called *congestion price*, which can be used to determine link weights for route determination in the model developed in this thesis.

3. Applying the Utility Model to Packet Networks

3.1 The Problem

We will show that the Utility Model maps neatly to a simple model of a packet-switched network. In this case, the relevant resources are no longer the CPU cycles, memory bytes and bus bandwidths of Khan's server, but instead the bandwidths and latencies[7] of the links and switches of a packet network. The novel features of a packet switched network – primarily the existence of multiple paths and hence multiple resource-sets for a given flow, cause the convenient linear ordering of possibilities to be destroyed. We will have to invent ways to recover linear order, and we embody these methods in the heuristic HEU. This gives rise to a new heuristic which is applicable to packet networks: N-HEU.

Our problem becomes to decide whether a new or changed traffic flow (that is, the traffic flow that will be associated with a Service Level Agreement, which one can view as similar to an ATM virtual path connection or an MPLS path) should be admitted, and if so, which path from source to destination it should use.[8]

The problem may readily be split into two distinct sub-problems:

1. the admission of a new SLA into the network

2. the modification, upwards or downwards, of the QoS level provided to an SLA which has been previously admitted to the network.

These subproblems are very similar, and we approach them in much the same way.
In order to apply the Utility Model to our situation, we must map the SLA's requirements to the resource requirements needed for the UM. We formulate the SLA's requirements as levels of Quality of Service (QoS), mapping each into a set of resource requirements for the network.

---

[7] We can also use other measures, such as packet loss rates and delay variation; many of these concepts are also present in the ATM specification (ITU-T I.150)

[8] Here is a major difference between our work and Khan's treatment of a server: In practical networks there are multiple routes between a given source S and destination D. Hence the set of resources needed to provide a given QoS level for a flow from S to D is no longer unique.

For our simple network model, we restrict these requirements to be the bandwidth and latency of the connection. As in Khan, for each level, we also specify a value of utility, to express how much the SLA's owner - the customer - is willing to pay.

Latency as a resource is unlike bandwidth: While a connection with a certain amount of latency may be required, it is not *consumed* in the same way as bandwidth. Thus latency resembles a constraint more than a consumable resource. Of course, given queuing for a link (which we ignore in this work, as our methods deal exclusively with resource reservation *before* any traffic flows), link latency does depend on the amount of traffic on the link. However, latency can be approximated by the path length of a connection. Moreover, lower latency connections are generally more desirable and thus might command higher utility. In our discussion, then, we do not treat latency as a resource for the utility model, but instead as a criterion for path selection.

A path meets a certain QoS level Q_i iff

1] the path's latency l_p is less than or equal to the QoS level's required latency l_Q, and

2] each link in the path has uncommitted bandwidth b greater than or equal to the QoS level's required bandwidth b_Q.

Finally note that path latency l_p is defined here as

$$\mathbf{l_p} = \Sigma \mathbf{l_l}$$

- the sum over the links of the path of link latencies.

As an example, consider an SLA needing a 500MBps connection, with latency to be no greater than 1.7 milliseconds, for which the customer is willing to pay $40. However, the customer will also accept other, lower grades of service, with lower utilities (offered prices). We can express this as a session profile for the Utility Model, in Figure 3-1:

	Level 1 (Bronze)	Level 2 (Silver)	Level 3 (Gold)
bandwidth (Mb/s	250	300	500
end-to-end latency (s)	2.5	2.0	1.7
utility value ($)	20	30	40

Figure 3-1. Example Session Profile

(Here we use the terms "Bronze", "Silver", and "Gold" to describe the QoS levels, but we can have any number of levels and they need not be named.)

In the multimedia server example presented earlier, each QoS level implies exactly one configuration of resources required. However, in an IP net, there generally will be multiple paths from the source of the flow to its destination (i.e., graph connectivity greater than one, for reliability and routing reasons). Therefore, more than one of these paths may satisfy the requirements of the circuit at various QoS levels and hence there is usually more than one configuration of resources meeting the SLA's needs.

Uniqueness in one sense, of a single feasible configuration of resources required per QoS level, is therefore lost. In addition, performing just a simple shortest path (i.e. by speed, or latency) discovery algorithm for each flow may not make best use of the network, as particular fast links may get overburdened. Furthermore, the resources in the multimedia server example are chosen from a small set as mentioned above - CPU cycles, memory bytes, etc. In the present case, however, *each link* in a network has at least one resource that must be managed – in our example, capacity – so the resource set becomes much larger, on the order of a few hundred links for an Enterprise Network of 100 nodes.

3.2 Assumptions

To apply the Utility Model to an ISP network, we make several assumptions about the ISP network and about the facilities supplied to it by the underlying facilities-based carrier. (ISP network operators typically do not operate transmission systems, but lease capacity, perhaps in the form of ATM VPs, from a facilities-based carrier who owns and operates them. There may in addition be cascaded vendors of optical channels [λ-vendors], fibre-vendors, and conduit-vendors.)

3.2.1 Assumptions about network links

1. Link latencies are fixed.
We will ignore variations in queuing delay, which are incurred in switches. This should

not be a problem provided the distribution of traffic is stable[9]; if it does not we may see a problem similar to that experienced by the second-generation ARPANET algorithm [MCQU80] - traffic oscillation under heavy loads [STAL97]. However, we are establishing paths, instead of routing individual packets. Once a path is established, a change in delay due to queuing will not force the circuit to be re-routed. This should dampen the oscillation experienced by the ARPANET algorithm. In addition, we know the requested bandwidth and so may be able to estimate the new delay values; this is especially valuable in the case of SLA admission. Finally, we reserve the *peak* bandwidth required; this is conservative but allows us to guarantee QoS. Recovery of some of the wasted bandwidth associated with bursty traffic is an important topic for further research; this subsumes knowledge of bandwidth consumed at *traffic time* as opposed to bandwidth requested at *reservation time*. Our research is entirely concerned with the latter time.

We note that latency under these assumptions can be reduced to the hop count from source to destination by assuming unit delay for all links. Delays dependent on queue length are a topic for future work.

2. Link costs are fixed.
Costs reflecting congestion-based pricing (Kadengal, op.cit.) could be easily added, incorporating their results into our model.

3. Equally utilized links are desirable
This assumption, if respected, should reduce the mean link queue length and might also help avoid thrashing as described above.

4. Call setup and teardown times for the flows are negligible.
We allow the SLA owner to make decisions based on whether or not she wishes to allow her SLA to be re-routed; if this is allowed, then we assume that the cost of re-routing is negligible. This may incur a momentary drop in quality; however, this is assumed to be acceptable to the SLA owner[10].

5.The network is either small or sparse (topologically, not geographically) and hence exploration of all possibilities is feasible, **or it is not,** in which case we will cap the number of possibilities explored by some rule. In this case our solution becomes heuristic rather than algorithmic, i.e. not guaranteed to find the global, truly optimal solution.

6.We do not split a session's traffic over multiple paths.
We assume that the session's bandwidth requirement is small compared to the capacities

---

[9] That is, changes in traffic through a switch causing excessive variations in queuing delay for other traffic.

[10] In many cases (non time-dependant flows) this can be circumvented by caching.

of the links (for example, megabits as opposed to terabits) and hence splitting a session over paths is often unnecessary.

3.2.2 Assumptions about Service Level Agreements

As mentioned above, a *Service Level Agreement* (SLA) is a contract between an ISP and a customer, for the carrying of traffic from one point in a network to another. In addition, we make the following assumptions about service level agreements:

1. A Service Level Agreement is created for a source node and a sink node; however, for our model we consider paths to be bi-directional and symmetric in capacity, so there is no effective distinction made between the source and sink nodes. However, N-HEU could be used to accommodate asymmetric SLAs by using unidirectional links.

2. There may be multiple service levels specified in an SLA request. For example, a customer may be willing to pay n dollars for m units of bandwidth, and $4n$ dollars for $2m$ units of bandwidth, if available. These are considered service levels, or grades of service, and they correspond to the QoS levels Q_n of the Utility Model.

3. An SLA's associated session will run for a specific length of time, specified when the SLA is created. It may recur, however. For example, a customer may want 500Mb/s between Chicago and Los Angeles every Tuesday between 11h00 and 12h00, for a weekly videoconference.

4. An SLA will be created and submitted long before the bandwidth is required (long lead time). However, occasionally this may not be the case, and we may need to deal with an SLA on short notice. The notion of SLA merges with the notion of call setup as the lead time approaches zero.

5.The session parameters are deterministic not stochastic.

6.Once an SLA is admitted, its associated session may not be canceled (i.e. removed) unless the customer consents. In addition, its service level should not be changed without the customer's agreement. Finally, the customer should be allowed to fix the routing of the SLA if this is desired. (Otherwise, we may change the routing assigned to an SLA before its traffic starts to flow, if we are able to accept more SLAs and thus increase revenue by doing so. However, in no case does the routing of traffic change after it has begun to flow – i.e. it is admitted and active. These restrictions are ignored only under network failures.)

7. An SLA is considered *active* once its traffic begins to flow, i.e. when its associated call or session begins to use its reserved network resources. Before this time it is considered *inactive*. While an admitted-but-inactive SLA may be freely re-routed, the session of an active SLA is *never* re-routed (fixed routing in the usual sense).

We will use the Utility Model to admit an SLA into a network, and an associated routing algorithm to decide the **fixed route** that all datagrams of an admitted SLA will use. (Establishing and policing such a route is the responsibility of the network; MPLS would be a suitable method to use.) Hence the network must be able to

- calculate a fixed route from source to destination(s) for a session
- reserve resources associated with the route (i.e. link bandwidth) as required for the SLA, and
- route all packets of the SLA along the route.

3.2.3 Other Assumptions

Finally, we assume that we are given:

- **a network *N***, consisting of a set of nodes and a set of links .
- **an SLA matrix *T***, whose rows correspond to SLAs at levels of QoS, columns to peak bandwidths, latencies, and offered prices, consisting of those SLAs currently active, with their QoS levels and routes; and
- **a routing algorithm *R***

and that all of these are feasible for the current state of N.

3.2.4 Establishing Fixed Routed SLAs

One possibility is to view SLAs as ATM virtual path connections (VPCs). When an SLA is admitted, or moved, we create an ATM VPC along the given route. The ISP (or the customer) can then route several user calls as VCCs within such an established VPC. In such a case, our resources, instead of bandwidth, may correspond to the ATM traffic parameters: peak cell rate, cell delay variation (for all connections), and sustainable cell

rate and burst tolerance (for VBR). The traffic parameters (together with the cell-rate algorithm defined in ITU-T I.371) would allow the network to monitor traffic and ensure that the parameters are not violated. Another, more contemporary, possibility is to use Multi Path Label Switching (MPLS) to define paths.

3.3 Routing Algorithm

In the multimedia server case, the Quality of Service mapping provides an explicit and unique list of the session's resource requirements. However, in our network model, we have seen that there may be a number of feasible sets of resources for each QoS level; each set represents a path, or route, and hence the mapping from QoS level to associated resource-set is no longer unique.

How to determine which routes to consider? We base our solution on two assumptions:

- Shorter paths (i.e. those using fewer links) will cost less, and will congest the network less (and hence allow a larger number of sessions), as they mean less total bandwidth used.

- Such paths will also allow greater revenue, as they will tend to have lower total delay (and hence will be feasible for higher QoS levels).

To address this problem, we apply a *k-th shortest* (elementary) *path* discovery algorithm to the network, in an effort to discover all, or a large number of, paths whose latencies fit the requirements for the Bronze (or lowest) QoS level for the flow. There has been a fair amount of work on k-th shortest path problems; for a good overview see [EPPS94 Of the several known algorithms for k-th shortest path discovery; we have selected a path-deviation style algorithm due to Martins (with work from Eppstein) [MART97]. These shortest paths then become candidates for the flow. Once these have been discovered, we can apply the Utility Model to the problem, as described below.

If the resource requirements of the flow cannot be met, we will then consider adding additional bandwidth to the network, by leasing it from the underlying facilities-based carrier. Since the UM will determine when the addition of a flow is infeasible in the current network, we can then consider leasing additional bandwidth for certain links. For reasons of cost and complexity, of course, this expensive option is considered last.

3.4 New Problems Arising in Switched Networks

Switched data networks have some interesting features that make the application of the Utility Model, originally developed for servers, less than completely straightforward. A discussion of some of these features follows.

3.4.1 The Crossgrade Problem

The heuristic N-HEU for solving the MMKP requires that the possible solutions be given in some order of desirability (Khan, op. cit.) in order to make a proper selection among them; that is, that $p_1, p_2, p_3 \ldots p_n$ be given in some fixed order. Specifically, the assumed order is by increasing value of utility U. In Khan's implementation of HEU, this is relatively straightforward: we have a specific order in which upgrades and downgrades take place, and, as he notes, if this ordering property does not hold for a given instance of the MMKP, it may be achieved by preprocessing.

In the case of networks, however, this is no longer true. Instead, we have *two* different ways that an SLA's net revenue may change. The first is by the use of upgrade paths – that is, paths that will yield increased revenue because they support a new, higher, SLA Qo level.

The second way is by use of crossgrade *paths*. These are the set of possible paths under consideration within a *single* SLA Qo level. These will not increase gross revenue, as they are all at the same Qo level and hence attract the same gross revenue. They may, however, affect *net* revenue (gross revenue less costs) because of differing costs to the carrier of providing different paths. The difference is illustrated below. Crossgrades did not exist in Khan's study because there was only one, unique set of resources per QoS level, and because Khan considered gross, not net, revenues.

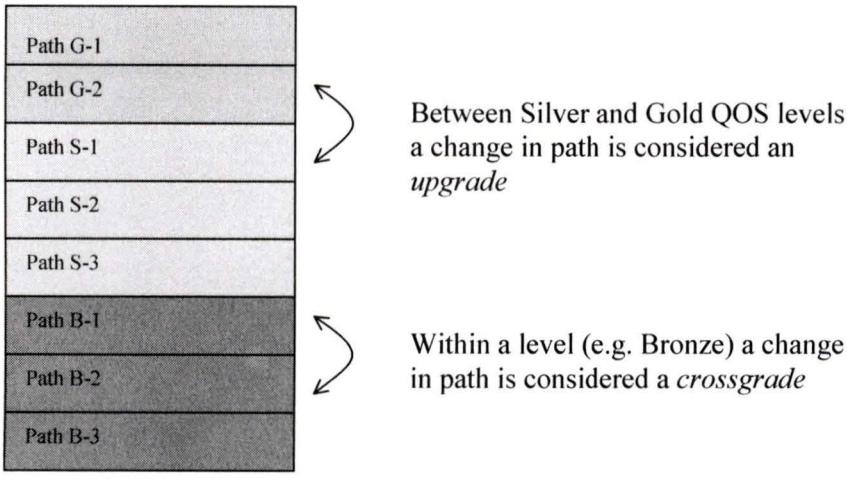


Figure 3-1. Upgrades and Crossgrades

3.4.2 Full Ordering of Paths

An upgrade or crossgrade may prove to be invalid, and unlike Khan's server problem, it is possible that a more desirable upgrade or crossgrade is valid although a less desirable one is not. For example, an upgrade from Bronze to certain Silver-level paths may be possible, although certain others may not be. An upgrade to Gold may be valid although all Silver upgrades are not. To further complicate matters, certain crossgrades within the Bronze level may not be possible, while others are. Finally, due to increased costs, it may even be the case that an upgrade results in lower net revenue. (Khan used gross revenue as his utility measure; we use net revenue.)

Here, then, our algorithm must deviate substantially from HEU. HEU can consider, in order, each possible QoS level within a QoS profile. As an example, if the Bronze level is feasible, the QMgr (KHA98, Ch 5) can then consider the Silver and Gold levels. If the Bronze level is infeasible, the search can stop at once: the higher levels cannot be feasible as the requirement for each resource is assumed to be monotone increasing with QoS level. In our network model, however, there may be many paths corresponding to the Bronze QoS level, and many paths corresponding to Silver, as illustrated above. We wish to refine the *partial* ordering of paths imposed by QoS level to yield a *full* or *linear* ordering of paths.

Further, we would like to preserve the *monotonic feasibility* property, which specifies that

$Q(P_2) > Q(P_1)$, and

P_2 feasible, =>

P_1 feasible.

(Monotonic feasibility is handy because we can stop searching for feasible upgrades or crossgrades as soon as we find the first infeasible one, if it holds.)

We see two possible ways of approaching the problem.

The first is to assign to each possible path a QoS level. Within QoS levels, the paths could be sorted by some criterion.The algorithm could then use this full ordering of paths as its list of "new" QoS levels to consider. Viewed another way, in this manner we expand the notion of QoS to include enough additional criteria, to refine the partial ordering imposed on paths by QoS level to a linear or full ordering.

There are several possible criteria. The possible crossgrade paths within a single QoS level could be ordered by total **latency**. Thus, we would first consider first the shortest paths and then the longer ones, a valuable feature.

A shorter path contains fewer links by definition, and therefore is likely to cost less. This leads us to a second possible ordering: ordering by **net cost within each QoS level.** Within QoS levels, we consider first those paths having the lowest net cost (cost less revenue derived), followed by those having higher costs.

This was the first approach implemented in the development of the SLAOpt prototype. However, this approach leads to problems. Among them are the following:

- Although the paths are listed by this approach in an order corresponding to their *desirability* as solutions, there is no direct correspondence to *feasibility*. That is, monotone feasibility is denied. Hence there is no guarantee that

QoS level Q_n feasible => QoS level Q_m feasible, $m < n$.

- It is also possible that a given path may be feasible for multiple QoS levels. For example, suppose there exists path P with available bandwidth B and latency l. This path is feasible under all QoS levels with

  bandwidth requirement $\boldsymbol{B_n \leq B}$ and

  latency requirement $\boldsymbol{l_n \geq l}$

and should therefore be considered at each of these QoS levels.

Instead, then, we apply a different approach. We do not restrict a path P to a single QoS level Q_n, but instead allow it to be considered as an upgrade at all QoS levels for which it is feasible. This is done as follows:

We can calculate the aggregate resource consumption for each QoS level for a given path P quickly, as the current resource usage does not change, and the resource usage for a QoS level will be the same for each component of the path. For example, given:

path P, consisting of links $p_1, p_2, \ldots p_n$ and

current resource usage $C = c_1 + c_2 + \ldots + c_n$ where

$c_i = u_i / p_i$

u_i is the current link usage and

p_i is the link capacity

Recall that we need to obtain the normalized aggregate resource consumption, c, as the heuristic N-HEU looks for the QoS upgrade for each SLA that minimizes aggregate resource consumption. We can obtain the normalized resource consumption a_0, as

$a_0 = W / |C|$, where:

$W = w_1 + w_2 + \ldots + w_n$ is the *weighted resource usage*

$w_i = c_i / p_i$

The aggregate resource consumption for the QoS level Q_m can then be given in terms of a_0 and the resource requirement r_m:

$$a_m = a_0 \, r_m$$

which gives

$$a_m = \frac{r_m W}{|C|}$$
$$= \frac{r_m \sum \frac{c_i}{p_i}}{|C|}$$
$$= \frac{\sum \frac{r_m}{p_i} c_i}{\sqrt{\sum c_n^{\,2}}}$$

We can generalize this for multiple resource constraints at each link (for example, bandwidth and switch CPU time) r_{jn} by calculating

$$a_{j0} = W_j / |C|$$

$$a_m = \Sigma \, a_{j0} \, r_{jm}$$

where C_j is the sum of the current resource usage of resource j. The coefficients a_0 need only be calculated once for each path, and we can quickly obtain the aggregate resource utilization for each possible upgrade. The Utility Model is then invoked to choose which SLA to upgrade.

This allows us to overcome the second problem described above, that of feasibility at different QoS levels, because a path is available for consideration at a QoS level iff it is feasible. It also solves the first problem (monotonic feasibility), *if* the QoS levels are ordered such that all requirements at QoS level Q_n are more restrictive than those at all QoS levels Q_m, $m < n$.

> For, if QoS level Q_n is feasible, then there exists a path P which has available resources to satisfy Q_n. Since QoS level Q_m , $m < n$, has resource requirements that are less restrictive than those of Q_n, then path P is also feasible under Q_m. Thus there exists then at least one feasible path for Q_m, and Q_m is thus feasible. QED.

In the more general case, where the restriction above does not apply, it is of course possible that a lower QoS level may not be feasible, but a higher QoS level is feasible.

To see this, imagine an SLA with two QoS levels. Q_1 has resource requirements

$B_1 = 100\text{MBps}$ and $l_1 = 30\text{ms}$,

and QoS level Q_2 has resource requirements

$B_2 = 250\text{MBps}$ and $l_2 = 35\text{ms}$.

In this case, the higher QoS level is willing to settle for longer latency if it can get higher bandwidth, and the QoS levels cannot be linearly ordered by simply linearly ordering their components. (This did not arise in Khan's work as he only considered single-component QoS criteria.)

Suppose there is exactly one path P_1 for this SLA, having

$B_{P1} = 300\text{MBps}$ and $l_{P1} = 32\text{ms}$.

In this case, Q_1 is infeasible (as P_1 has latency which is too great), but Q_2 is actually feasible, since its latency requirement is less restrictive than that of Q_1. The heuristic will then upgrade the solution directly from Q_0 to Q_2, because Q_1 is infeasible.
If no upgrade decreases the net aggregate resource utilization (Δa is positive for all upgrades), N-HEU selects the upgrade which maximizes the utility gain per extra unit of aggregate resource Δp.

When allowing crossgrade paths to be considered as upgrades, we must take care that we do not get caught in a *crossgrade loop*, where session S crossgrades from path P_1 at QoS level Q_n to path P_2 at the same QoS level; then, via some set of crossgrades, back to path P_1. It is easy to demonstrate that this is possible.

Imagine two paths P_1 and P_2 between a given source and destination $[S,D]$ where the QoS characteristics of each path are equal, and one SLA. Then, suppose that the algorithm eventually selects path P_1 at QoS level n, where Q_n is the highest level of QoS that is actually feasible for the system.

We know that any QoS level $m > n$ will not be selected due to infeasibility. However, to the algorithm, path P_2 looks attractive; because P_1 is in heavy use but P_2 is not.

Since r_2 (the resource used by P_2) has 0 utilization it carries no penalty; then P_2 has a change of aggregate resource. Hence, the algorithm will move the SLA to P_2. Now, of course, P_2 is in heavy use and P_1 is not, and P_1 once again looks attractive to the algorithm and will be selected. At this point, we are stuck in the loop.

In order to avoid this situation, we modify Khan's implementation of the effective gradient. As our objective when considering an upgrade path is really to decide whether the new path P' represents a more effective use of resources than the current path P, we compute the effective gradient for P and the effective gradient for P' separately. When doing so, *we do not consider the resources currently used by P.* We consider that this gives a better representation of the difference in effective gradient between P and P', because it simply represents a choice between two alternatives given an equal starting point[11]. It also helps to avoid simple crossgrade loops; in our example above, P_1 and P_2 look equally attractive.

3.5 Resource Contention

Resource contention exists when there is more demand than supply of some resource, so users must contend for it. In our situation, resource contention occurs when we have allocated more of a given resource than is available to us. QoS guarantees are then broken, and customers may consult their lawyers.

There are several situations where this might occur, including:

- Hardware malfunction of a link or switch
- Reconfiguration of the network by adding or removing resources such as links or switches
- Seizing of resources by an entity outside of the control of the Utility Model

The first two cases will occur, even if the Utility Model controls all resource consumption. The last arises in a shared environment, such as a corporate network attempting to offer QoS guarantees, embedded in a larger network without admission controls. Here, the Utility Model does not control all resource consumption. [CHEN98]

---

[11] What it does not do, however, is take into account the situation where we may wish to penalize a new path P' because it involves resource reallocation.

In this situation, when resource contention occurs, we must attempt to adapt to the new state of the network, while attempting to preserve the QoS contracts established with the active sessions in the network.

3.6 Minimum QOS Guarantees

A related problem to the above is best demonstrated by looking at the feasibility of a set of SLAs when N-HEU is executed on them. Before N-HEU runs, any SLA that has not yet been admitted is feasible (as the non-admitted QoS level Q_0 has no resource requirements). In addition, SLAs that have already been admitted will be feasible. However, when N-HEU begins, it initially *downgrades* any SLAs to the lowest level. Khan *(op. cit.)* assumes that this solution is feasible; certainly, any non-admitted SLAs are still feasible (as Q_0 is the lowest QoS level). However, an SLA which has already been admitted will be downgraded in one of two fashions:

1] If it has chosen to restrict its QoS levels, and the N-HEU execution is not ignoring such restrictions (such as when a link is over capacity) it will be downgraded to its lowest acceptable QoS level.

2] If it has not chosen to restrict QoS levels, or N-HEU is ignoring such restrictions, it will be downgraded to QoS level Q_1.

However, if the QoS level which the SLA has been downgraded to is not feasible, then Khan's assumption is not valid. In general, we have found that N-HEU will tend to upgrade an SLA out of a non-feasible routing; however, such action is not guaranteed by the heuristic, especially if no such routing is available.

Therefore, the following precautions apply to SLAs:

1. An SLA is suggested (but not required) to set Q_1, its minimum QoS level, to be the least restrictive QoS level for all resources. In this case, if QoS level Q_n is feasible for some path P, Q_1 is also feasible.

2. If an SLA does not follow the above guideline, a situation may arise where the resource availability on an SLA's routing does not meet its resource requirements. It is left up to the individual implementation whether to allow the SLA to proceed (but with reduced resources) or to terminate the SLA, possibly attempting to readmit it. In our prototype either is allowed.

3.7 Request Batches and Online Problems: Quantizing arrival times

Khan (op. cit.) deals with the Utility Model as a strictly *online problem* [BORO98]; that is, as each request or batch of requests comes in, the algorithm is used to decide whether or not to admit the SLA. Advanced knowledge of session requests is unavailable. As well, the level of QoS which it, and the other sessions in the system, should be assigned to maximize total revenue is decided.

Although in our network model we allow an SLA to begin at time t_1 and end at time t_2, we do not allow out-of-order admission of SLAs. That is, we will admit an SLA S which is to be activated at time t iff there does not exist an admitted SLA S' which is to be activated at time $t' > t$.

If such admission were allowed, it would introduce another dimension to the problem. Currently, when we make an admission decision, we need only consider the state of the network at time of admission t, which is easily computed. If we were to allow out-of-order admission, this would no longer be sufficient. For example, if we have an SLA S which we wish to admit at time t_1, but there exists an SLA S' which has been admitted for time $t_2 > t_1$, then we must ensure that the system is both feasible at time t_1 (when S is admitted) and at time t_2 (when S' becomes active). This can be seen in the following figure:

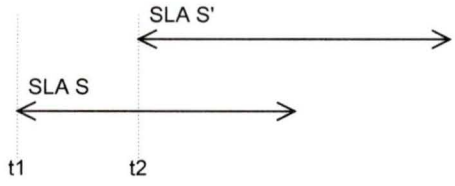


Figure 3-1. Out of Order SLAs

For a problem involving large numbers of SLAs, this becomes impractical.
Instead, we can allow requests to be *batched* over a period of time t_b. We schedule an interval of time t_s, called an *admission epoch* or *epoch*, before the desired admission time t_a (which may vary depending on the number of requests that are batched) during which we will attempt to admit the requests batched. During this time no further requests for admission at t_a are accepted. Once all requests are either admitted or rejected, we return to accepting requests for the next batch.

A possibility for future extensions would be to look at time-slicing the period $t_1 - t_2$ in order to reduce the complexity involved. See also section 7.2.3.

4. Solution, by an example

Consider the simple network shown below, and focus attention on the marked path from source S to destination D. Assume that an SLA has been submitted for service from S to D.

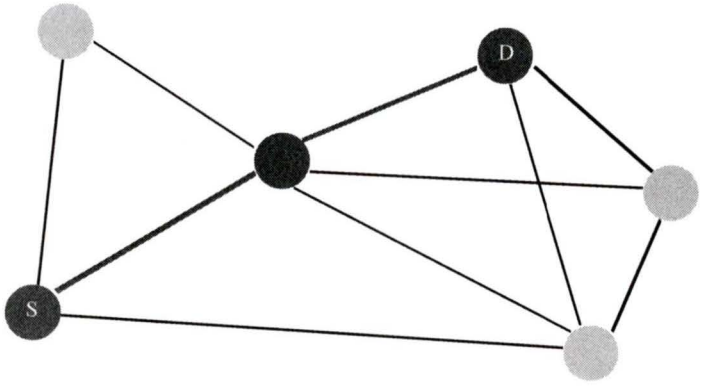


Figure 4-1. Example Network N

4.1 Admission

Suppose that SLA (S, D) is currently inactive. We need to find the following information:

- A routing for SLA (S, D), and its QoS level Q_n
- The routing for any flows that may have been changed while routing (S,D).
- Any new link capacities

These, taken together comprise the new state of the network *N*.

Before proceeding we must define two concepts absent in Khan's formulation of the Utility Model: *nearness*, and *net cost*.

4.1.1 Nearby Paths

In the multimedia server example, the list of limited resources considered is small, and, more importantly, all sessions compete for the same resources. In our example, neither of these is the case. There are many network resources that need be considered, and any two given sessions will not necessarily require the same resources in the network. We assume that paths that are near the path under consideration are more likely to be consuming resources that could be used by (S,D). We define a *nearby path* as one which has nodes in common with (S,D) – more specifically, in order:

1. Those paths which have startpoint S and endpoint D, or vice-versa.
2. Those paths which have S or D as an endpoint or startpoint
3. Those paths which contain both S and D
4. Those paths containing either S or D

(Although 3 is more likely to be near to (S,D) than 2, the latter is much easier to compute.)

4.1.2 Net revenue

The Utility Model selects QoS levels for SLAs based on their *utility* and their resource usage. In Khan's implementation, gross revenue was taken as the measure of utility, as Khan assumed that the resources of the multimedia server were paid for, and so did not have *costs*.

In our work, we associate a *cost* with resource usage, and we consider

$$\text{net revenue} = (\text{gross revenue}) - (\text{cost})$$

as our measure of utility.

Cost can reflect many things: Some possibilities are:

- costs of link bandwidth and switch utilization, which may be important on a leased line,
- rerouting charges, or

- link delay; as with the ARPANET routing algorithm [STAL97]

These costs may be fixed or may vary with the amount of resource utilization.
In addition, we note that Khan's measure of utility is in fact a *rate*; a revenue per unit time. In order to properly assess the revenue of any up-front or one-time charge[12], we need to amortize this charge over the duration of the session.

For our implementation we will use a simple cost model of a fixed rate for link usage. However, other, more realistic models exist. For example, given

an SLA S with duration d,

a path P and

a QoS level Q, where Q specifies a certain bandwidth b,

we can define a *utility rate* u as

$$u = r_Q - (f_P/d + c_P + v_P b)$$

where:

r_Q denotes the rate that the customer is willing to pay for Q,

f_P denotes any up-front charges for the use of P,

c_p denotes any fixed rate for P, and

v_p denotes variable (resource usage) based charges for P.

Of course, the customer's rate r_Q could also be made to include up-front or variable charges as well.

4.2 Example

As described above, in order to activate a session (S,D) requested by an SLA, we first use a path discovery algorithm to find the possible routings for (S,D). These paths are then sorted by aggregate resource usage.

---

[12] Or, for that matter, a one-time payment

We then attempt to find the best feasible path available to (S,D). If such a path exists, (S,D) is routed along it, and the admission algorithm terminates. If, however, no such path exists, then we accept *nearby* paths as candidates for re-routing, and then apply the Utility Model.

We add the SLAs using each of a group of nearby paths, in order, to the set S' of possible SLAs to be re-routed, invoking the Utility Model after each group is added, until a solution is found. If no solution is found before a time limit expires or before the algorithm for finding nearby paths terminates, we consider adding resources to N . (N is assumed to be owned by a network ISP who leases resources from an underlying facilities-based carrier.) Otherwise, the SLA (S,D) is considered to be *infeasible* for the current state of the network N.

4.3 Changes in SLA requirements

In the event that SLA (S,D) wishes to increase its bandwidth by some amount Δc, we need to find the following:

1. the new routing for SLA (S,D)

2. the new routing for any other flows that may have been changed

3. any new link capacities

4. the new state of the network N

We use a perturbation-based approach, changing as little as we possibly can. We believe that this will be fast, and will also lead to a more stable system. The chief foreseeable disadvantage is that this approach will not necessarily give an optimal solution, even using BBLP. This is because the selection of possible paths must be heuristic (in this case by shortest path).

To accommodate an increase in bandwidth of a session, we apply the following four techniques – the phases of the algorithm - in order, until a solution is found:

1. Do not change the routing of (S,D); try to find the additional bandwidth on its existing path.

2. Reroute (S,D), changing nothing else

3. Reroute (S,D) and other sessions (flows). (This is problematic: please see below.)

4. Increase link capacities.

Here is a more detailed description of the phases:

1. We simply check whether the current routing for (S,D) has enough surplus capacity to accommodate the increased flow. If this is the case, then we do not need to change any routings. We simply increase the bandwidth allocation of flow (S,D) and decrease the free bandwidth of all links along the path to reflect the increased allocation to (S,D).

2. If the current route for (S,D) does not have enough free bandwidth, we next attempt to locate some route for (S,D) that does have sufficient free bandwidth, while changing nothing else. This involves first finding the shortest possible paths with sufficient bandwidth and acceptable latency (or other properties implied by the definition adopted of QoS), and second, choosing the best of these paths.

As mentioned above, to discover acceptable paths for this flow, we apply a *kth shortest elementary paths* algorithm to source S and destination D of N. If the network is small (as in the base UUNet network, for example, or in enterprise networks) we may be able to allow the algorithm to run to completion (unless the latency, or the net cost, becomes excessive)[13]. If the network is large, we may be forced to reduce the number of paths considered by using some heuristic (i.e. just a strict cap on the number of paths, or, to reduce complexity, by running the algorithm on some subset of N). In any case, the algorithm will return some subset P of the paths in N.

We then invoke the Utility Model on the subset P; this will select the path

$$\boldsymbol{p} \in \boldsymbol{P}$$

that will return the greatest net revenue, and that has sufficient capacity to accommodate the increase in bandwidth.

It is possible, of course, that simply changing the routing for SLA (S,D) will not allow the SLA to be feasible; this will be the case if none of the possible routes for (S,D) discovered by the kth shortest path algorithm has excess capacity of at least the original SLA bandwidth plus $\Delta c_{(S,D)}$.

We note that we either know (cached from step 2), or can generate reasonably quickly, the kth shortest paths for our SLA (S,D) and all other SLAs in the network. However, in large networks we do not wish to invoke the Utility Model on the entire graph. There are

---

[13] See section 6.1 for more details on selecting a criterion for this.

two obvious reasons for this: First, there is the issue of complexity; and second, in keeping with our perturbation-based approach, we do not wish to arbitrarily re-route the flows of a large number of existing agreements. We choose, then, to reduce complexity by only looking at the flows of SLAs using *nearby paths* (nearby SLAs for short). Again, we apply the definition given above for *nearby* SLAs (see section 4.1.1.) While in general, we wish to find

$$(S',D') \mid (S',D') \cap (S,D) \text{ is maximal,}$$

this requires a lot of computation and our intention is that the rules as given above (i.e. select those SLAs which share a maximal number of endpoints with the given SLA) will give a reasonable approximation. We wish to select some number n of nearby SLAs; these SLAs S' become candidates for re-routing. Of course, an active SLA will only be rerouted, or have its quality level changed, if the customer[14] has indicated that she wishes to allow this.

We apply the Utility Model to our set of routes S' to attempt to discover if S' contains a routing which the UM finds to be *feasible*[15] (and near optimal) for these SLAs. If so, then we have placed $\Delta_{(S,D)}$ and are finished. If not, we can introduce some nearby SLAs

$$s \notin S'$$

and re-evaluate. If sufficient time is available, and there are no objections to re-routing SLAs, we can continue evaluating the Utility Model with additional SLAs even after a solution is discovered. In this way, if there exists some set of routings R' that raise revenue but requires additional re-routing, it may be found. In either case, we continue evaluating until an adequate solution is found, or time expires.

3. In this case, we attempt to re-route existing flows by choosing new routes for them, so as to open up a path of adequate capacity and latency. This problem has been studied for switching matrices for circuit switching by Benes [BENE65] and is known to be hard[16].

4. If no solution has been found after the previous step, we consider that there is insufficient capacity in the current network N. We then examine the possibility of requesting additional capacity on one or more links in the network from the underlying facilities-based carriers. For convenience we will assume the

---

[14] Or, more likely, their service provider.

[15] We use *feasible* in the sense of i) respecting resource constraints and ii) increasing net revenue.

[16] An alternative way to view this is as a *set-scheduling* problem

homogeneous case – that is, the cost of adding extra capacity Δc to all links is constant.

Again, in an attempt to avoid perturbing the system excessively, our first attempt is to determine whether we can add the required additional capacity to (S,D) (and thus place the additional $\Delta c_{(S,D)}$) without reducing net revenue. If we cannot, we apply the Utility Model.

In order to do this, we will extend the idea of net cost as presented above. Recall that given a path P and an SLA S with QoS level Q, we defined utility rate u as

$$u = r_Q - (f_P/d + c_P + v_P b)$$

Of course, a path is a set of links, so u can be expressed as

$$u = r_Q - \Sigma^{l \in P} C_l(d,b)$$

where C_l is some link cost rate function dependent on the duration and required bandwidth.

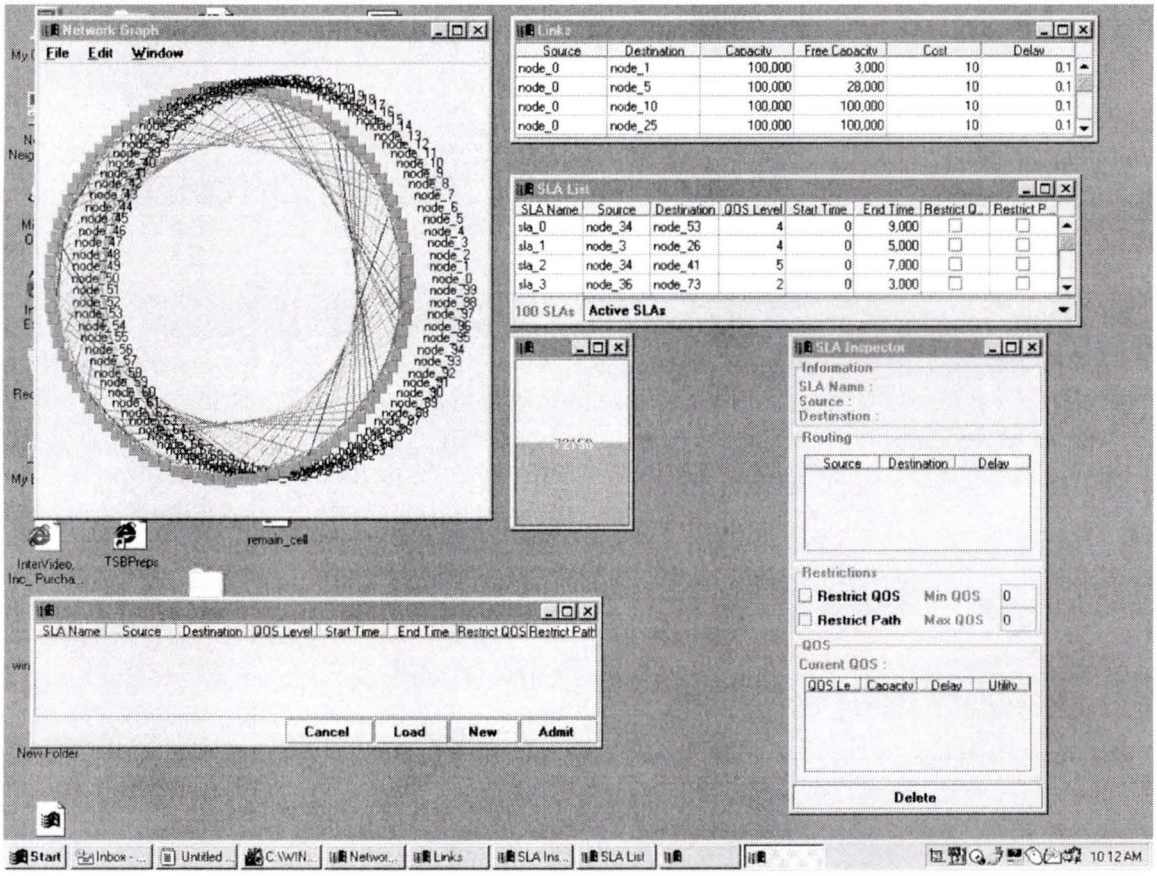
Now, suppose that for any path, there exist some links whose free capacity $c < b$. We simply extend the cost rate function C_l to include the required charges for adding capacity to the link if b is greater than the free capacity c. If b is less than or equal to the free capacity, then this function $C_{l,new}(b)$ will be equal to $C_l(b)$; if adding the required capacity is not possible, then $C_{l,new}(b)$ will be infinite, and the path is infeasible. In this way we deal neatly with requests to the underlying facilities-based carrier for more link capacity. Although we could in principle consider this step at all times, we assume that adding bandwidth is not only expensive in dollars but is also generally undesirable, unless no other choice exists.

5. Implementation

In the previous Sections, we have described the Utility Model as formulated by Khan, and we have described the Service Level Agreement, routing paths, Quality of Service levels and mappings, and our notion of monotonic feasibility. In this chapter, we will present implementation details of the Service Level Agreement Optimizer SLAOpt, a prototype Java implementation of the Utility Model, including the inevitable differences between the implementation and the model.

5.1 SLAOpt

Figure 5-1 shows a snapshot of the SLAOpt prototype running in Java[17].



[17] In this case, under Windows98. We chose Java for portability and ease of coding, but not for efficiency. Re-implementation in a compiled language such as C might decrease runtimes significantly

Figure 5-1. SLAOpt Prototype in Action

The platform used is Sun's JDK 1.1.8 with Swing (Java's windowing toolkit), version 1.1.1. In addition, we used Sun's Java XML parser, JAXP, for message parsing.

SLAOpt consists of several components:

- The *Network Model* represents the network under analysis, and stores current capacities, cached paths, and node information. As it is responsible for routing, in our implementation it also implements the *k*th shortest path algorithm for the network.
- The *SLA Directory* stores SLA profiles, including their QoS mappings.
- The *QoS Management Engine* implements the N-HEU heuristic[18] to select QoS levels for the SLAs.

In turn, the components must communicate with the user of SLAOpt and with the customers (the people who request network services and pay for them, the owners of SLAs). This is done via two interfaces:

- The *Graphical User Interface* shows the system's state to the SLAOpt user and allows her to modify the system.
- The *SLA Interface* allows customers to request SLA admission. It is implemented via XML text messages over sockets. This interface also allows the network manager to modify the network parameters.

5.2 The Network Model

To SLAOpt, a *network* consists of a set of nodes and a set of edges or links. The network under consideration is read from two files when SLAOpt is started. The *node file* contains a list of all the nodes of the network. Each node is given a name and a location, (generally in degrees of latitude and longitude, but any representation may be used as long as all nodes follow this representation.). The following is an example of a node:

node name	*LAT*	*LONG*
Vancouver	49.16	-123.70

Figure 5-1. Node Format

The location data is used only for display of the network by the GUI.

---

[18] We use HEU as opposed to BBLP for reasons of speed and simplicity.

The second file read by the network model is the *link* file, which contains a list of all links in the net. Each link has two endpoints (which must be node names), and several parameters: initial capacity, initial cost, and delay. The following gives an example of a link:

source-name	*dest-name*	*capacity*	*cost*	*delay*
Vancouver	Los Angeles	100e3	1	100e-3

Figure 5-2. Link Format

In this case, we create a link from Vancouver to Los Angeles with initial capacity of $1.0x10^5$, cost of 1, and delay of 0.1. The values are unit-less; capacity and cost are integers, while delay is a float.

We assume that links are bi-directional and symmetric.
From this information, the program creates a network model. This model has two principal functions: first, keep the state of the network, and second, provide possible routing paths to the QoS Management Engine. The model can be displayed to the user through the Network Canvas component of the GUI.

5.2.1 Network State

As described above, each link in the network has several parameters that affect the QoS Management Engine (QME - see 5.4. below); these are *cost*, *delay*, and *total* and *used* bandwidths. The nodes do not have parameters (although it would be possible to introduce them.) The network model includes methods for modifying each of these parameters.

The current *cost model* is implemented as a constant, time-invariant cost for any SLA wishing to use this link as part of a path. While this is unrealistic, there are several possible alternative cost models, including cost per unit of capacity, and a flat cost plus a cost per unit . (This may be changed if desired.) More information about the net cost model may be found in the description of the QME in section 5.4 .

The value of the *delay* parameter is used for path length when computing paths for routing SLAs. In accordance with our assumptions in 3.2.1 this is not currently changeable by the user, although it is taken into consideration when determining the feasibility of a path.

The total and used bandwidth represent, respectively, the capacity of the link, and the amount of this capacity which is currently in use; this is the main resource requirement considered by the UM. A link is considered *overbooked* when the used measure exceeds the total measure. The *free* bandwidth is simply calculated by taking the difference.

We next implement the *path* data structure, which consists of a vector of the links comprising the path; because the nodes do not have any requirements, they are not included. This structure is used for routing path discovery, described below.

5.2.2 Paths for Routing

In order to determine a set of possible routes for an SLA, we require the Network Model to provide a list of possible paths or candidate routes between the source and destination of the SLA. As described above, we implement a *kth shortest path* algorithm, under the assumption that shorter paths (measured by total delay) are more desirable than longer ones, other things being equal.. The algorithm implemented by SLAOpt is a deviation style algorithm[19], and is briefly described below; the algorithm is fully detailed in Martins (op. cit.)

.

Given two nodes *src* and *dest* we must first compute the shortest path tree rooted at *dest*; that is, we must obtain the shortest paths to all nodes from node *dest*. For our model this is done by invoking Dijkstra's algorithm[20] [DIJK59]. This algorithm also provides us with the initial shortest path. This path is placed into an empty set of paths under consideration. We then loop as follows:

> We retrieve the shortest path in the set of paths under consideration, which becomes the k^{th} shortest path. We increment k; then, for the current path, we create a set of *deviation paths*. These are created by considering the shortest deviation from each node of the path; that is, from a node v_i of the path p_k we find the arc a_n such that:
>
> i) a_n does not form a cycle with p_k
>
> ii) a_n originates at v_i
>
> iii) a_n is the shortest such arc that is greater than the current arc a_h
>
> We create a new path from the head of the original path, the deviation, and the shortest path from the deviated node to the destination. If any deviation path creates a cycle, it is rejected. These paths are then added to the list of paths under consideration and the loop is restarted.

---

[19] The algorithm was selected most notably because it is an elementary path algorithm; that is, it generates the k^{th} shortest *loop-free* paths between source and destination.

[20] Chosen for simplicity

In order to quickly determine the arc a_n we store the arcs such that all arcs are grouped first by tail node, and then sorted by length within their groupings. Finding a_n is then simply a matter of incrementing a pointer from the current node a_h.

We terminate the loop, either when the set of paths under consideration is empty, or when either k or the total delay exceeds some threshold that we provide.

In the case of SLAOpt, it is not strictly necessary to continue executing, once we have exceeded the delay requirement of the lowest QoS level[21]. The counter-argument is that we may wish to cache the shortest paths between a given source and destination, for the use of other SLAs, and these SLAs may have vastly different delay requirements. The solution used by the QME is to retrieve the full list of paths, and then clone the list, removing those paths that exceed the lowest delay requirement. This allows us both to cache the results from the k^{th} shortest path algorithm, and to avoid considering paths that will be automatically rejected by the QME.

The vector of paths created by the algorithm is then returned to the QME, which can then use these paths as the possible routes for the SLA.

5.3 The SLA Directory

The SLA Directory comprises three lists of SLAs: the *admitted* SLA list, the *rejected* SLA list, and the *completed* SLA list. Each is implemented as an array of SLA data structures, which are described below.

The admitted SLA list details those SLAs which are currently *admitted SLAs* in the network; that is, they have a QoS level > Q_0 and are currently candidates for consuming network resources. The rejected SLA list contains those SLAs whose admission bids were rejected by the QoS Management Engine. Finally, the completed SLA list contains those SLAs which were removed from the network either through action of the customer, or by the QME as a result of an unforeseen network event (such as the loss or downgrade of a link).

[21] In addition, doing so simply gives the QME invalid paths that must be considered (and rejected.)

5.3.1 The SLA in SLAOpt

SLAOpt represents each SLA as a Java object. Each SLA must be given a name (which should be unique), plus a source node and destination node; these nodes must exist within the network[22].

In order to be accepted by the QME, an SLA must also have at least one QoS level. These QoS levels are stored in an array, and are referenced by index. Because Java arrays are 0-based, we use QoS levels 0-(*n-1*) for valid QoS levels, and QoS level –1 for the rejected or non-admitted QoS level. When an SLA is created in SLAOpt by the GUI user, it does not have any QoS levels, and the current QoS level is set to –1, The structure for the QoS levels is described below. The SLA Directory does not enforce ordering of QoS levels, but customer changes to the lowest QoS level are not allowed; this would allow the customer to defeat admission by redefining the accepted SLA's lowest QOS level to be more restrictive, i.e. providing a higher level of service.[23]. As detailed in section 3.6, it is recommended (but not enforced) that the lowest QoS level remain the least restrictive QoS level for all resources.

The SLA structure also stores a current routing (which is initially null) and a list of possible paths (which is generated by the Network Model described above, and is also intially null).

As mentioned in section 3.2.2, we wish to allow the SLA customer to restrict both the routing, and the possible QoS levels of an SLA, once it has been accepted by SLAOpt. We provide two flags (the *Path Restriction* flag and the *QoS Level Restriction* flag) for this purpose. When the Path Restriction flag is set, SLA instructs the QME to only consider the current path when adaptation decisions are made. This has the effect of preventing the QME from re-routing the SLA during normal operation. Similarly, the QoS Level Restriction flag instructs the QME to only consider a restricted set of QoS levels; the SLA customer provides the SLA Directory with a range of desired QoS levels. Both flags can be ignored by the QME under some circumstances.

5.3.2 The QoS Level Representation

SLAOpt represents each QoS level by a simple Java object that contains the required bandwidth for this QoS level, the maximum allowable total delay, and the utility (the net revenue that the SLA customer is willing to pay, less costs) for this QoS level.

---

[22] The GUI restricts the user to selecting nodes that exist in the network; however the SLA Interface does not enforce any such restriction.

[23] See Chen [CHEN98] section 4.1.2

5.4 The QOS Management Engine

The main task of the *QOS Management Engine*, or QME, is to implement the N-HEU heuristic described in section 4 for SLAOpt. A call to the heuristic is called an *adaptation* and is done at several distinct times:

- When a set of SLAs is proposed to be added
- When the provisioning (e.g. link or switch capacities, due to failure or reconfiguration) of the network changes
- When an SLA's QoS parameters change
- When a set of SLAs is removed from the system.

The first scenario corresponds to admission (section 4.1); the second to failure or reconfiguration of the network; the third to changes in an SLA's requirements (section 4.3); and the fourth to the end of a user session or call.

While performing an adaptation, the QME may choose whether or not to respect the SLA's path or QoS level restrictions. Generally, restrictions are only ignored in emergency situations (such as the unexpected removal or loss of a link).

5.4.1 Modifications to the HEU heuristic

The implementation of the N-HEU heuristic by the QME is mostly identical to that proposed by Khan (op. cit.) A major difference is that proposed in section 3.4.1 to deal with the crossgrade problem.

When the session upgrades are considered, *the aggregate resource consumption per unit,* a_0 , for each of the possible paths P, is calculated. From this the change in aggregate resource Δr is calculated for all QoS levels

Q_n, where $n >$ current QoS level

Following the N-HEU heuristic, we then select the path/QoS level combination with the greatest negative change in aggregate resource, or, if no such combination exists, that combination which provides the greatest value gained per extra unit of aggregate resource.

The only other major change to N-HEU is the minor change in how Δr is calculated, as detailed in section 6. Briefly, when considering a move from one path to another, the current path's resource usage is not considered when computing Δr.

5.4.2 Net Cost

Unlike previous incarnations of the Utility Model [KHAN9, CHEN98], we chose here to reflect the fact that certain network resources (links) will carry a greater cost for use than others. This is reflected by assigning a *cost* to each link of the network. SLAOpt uses a simple cost model, where there is a flat fee for *each* use of a link. Other possibilities for cost models include, but are not limited to

- A flat fee for only the *first* use of a link; that is, the fee is charged if *any* SLA is using the link, and is not charged if no SLA is using the link.
- A fee based on utilization; an SLA is charged a fee for using a link based on the amount of bandwidth it wishes to consume.
- Some combination of either the flat-fee scheme or the consumption based scheme.

5.4.3 Nearby Paths

We defined nearby paths in Section 4.1.1. SLAOpt uses a similar, but slightly simpler definition of nearness.

Two SLAs are considered *near* in the following order:

1. The SLAs share both source and destination nodes, or one SLA's source is the other's destination, and vice-versa.

2. The SLAs share either source or destination nodes, or one SLA's source is the other's destination.

3. Neither source nor destination nodes match.

SLAOpt considers SLAs to be *near* if the either of the first two conditions hold. However, the user may change this setting.

5.4.4 Admission

SLAs are normally added to the Optimizer in batches; this gives NHEU more flexibility, especially when currently active SLAs wish to restrict their routing and/or QoS levels[24]. Similar to the Utility Model, the QME takes a perturbation approach to admission, and makes two attempts to admit SLAs:

[24] It is of course possible to create a batch of one.

- It first simply performs an adaptation using only the batch of SLAs that are currently candidates for admission[25]. All SLAs that are admitted are added to the active SLA list; those that are not admitted are sent to the second step.

- Secondly, it considers those SLAs that were rejected by the first step. The QME discovers admitted SLAs that are considered *near* to the rejected SLAs.[26] It then performs an adaptation using the SLAs rejected in step 1 and the near SLAs discovered in this step.

The QME respects SLA restrictions (path restrictions and QoS level restrictions) at all times during admission.

Any SLAs that have not been admitted after both steps have completed are deemed *rejected,* and are added to the rejected SLA list. If the customer wishes to persist in requesting admission for these SLAs, she should increase her bid and resubmit the SLA for admission. (Economists tell us that bidding and auctions are an effective and appealing way to allocate resources.)

5.4.5 Changes in operating conditions

Changes in operating conditions in SLAOpt occur when the capacity of a link is changed, or when the cost of a link is changed. The change may be a small modification to a link's capacity, or may involve the removal of the link[27]. In either case, several SLAs may be affected.

The QME first determines if the network is in a critical situation; that is, if the affected link is overbooked. If the link is *not* overbooked, the QME determines those SLAs that are currently using the affected link[28]. The QME then simply performs an adaptation on these SLAs, while respecting any SLA restrictions. However, if the link *is* overbooked, the QME is in a *resource contention* situation (see Section 3.5) In this case, we cannot guarantee that all admissions will be respected. The QME first downgrades all affected SLAs to QoS level Q_1, and performs an adaptation on these SLAs.

---

[25] These SLAs all begin at QoS level Q_0.

[26] These near SLAs may, of course, include SLAs admitted in the previous step.

[27] SLAOpt does not actually allow *removal* of links, but setting the capacity of a link to 0 may simulate this.

[28] It may also optionally find SLAs which are considered *near* to these.

During this adaptation, the QME ignores all SLA restrictions on affected SLAs. If the adaptation is successful (that is, the affected link is no longer overbooked), the QME has finished. However, if the link remains overbooked, the QME tries the following strategies, in order:

- It first finds nearby SLAs and performs the same adaptation, ignoring SLA restrictions.
- If this fails, it downgrades the affected SLAs (and neighbors) to QoS level Q_0[29] and attempts to re-admit them.

Any SLAs which are left at QoS level Q_0 after the second step are considered rejected, due to network interference; they are added to the *completed* SLA list and a message is logged to the console.

5.4.6 Changes in SLA requirements

A change in SLA requirements will occur when an SLA's QoS parameters change: in SLAOpt, this can be either the bandwidth requirement, the delay requirement, or the utility. Requirements can also change when a customer adds a new QoS level to an existing SLA. The procedure followed in either case is similar.
When a QoS level is added to an existing SLA, the QME first ensures that the user is not trying to add a QoS level below QoS level Q_1. It then discovers any nearby SLAs, and performs an adaptation on these. (That is, it considers only the subgraph of N's graph defined by the paths of the SLA and its nearby neighbours, and attempts to get the necessary additional resources for this SLA by trying the four tactics, namely

- look at the existing path for free bandwidth,
- reroute this SLA only,
- reroute this and nearby SLAs, and
- ask for more network resources from the underlying facilities-based carrier.

The QME respects SLA restrictions during this procedure.

There are several possible reasons for a change in an SLA's QoS parameters. Two of the more important are:

- An SLA's owner desires more bandwidth on the current QoS level[30], and

---

[29] This has the effect of revoking the admission of the SLAs

[30] Although it will also occur when an SLA desires *less* bandwidth. Presumably in either case there will be an appropriate modification to the utility.

- An SLA's owner wishes to raise its utility value for a higher QoS level, essentially "raising its bid" for that QoS level, presumably in an effort to get itself promoted to the higher level of QoS.

The QME first determines whether the QoS level being changed is the one currently in use; if this is the case, the QME must determine if the new bandwidth requirement of this QoS level is still feasible. If it is not feasible, the QME first downgrades the SLA to QoS level Q_1[31].

The QME then performs an adaptation on the system, using the affected SLA plus any neighbors. This allows the system to adjust to the changes made to the SLA's QoS parameters. Again, the QME respects all SLA restrictions during this adaptation; in this way, any SLA which does not wish to be re-routed or have its QoS level changed can preserve these settings.

5.4.7 Removal of SLAs

When an SLA has completed, it must be removed from the SLAOpt system, so that its resources can be freed for use by other SLAs. Like admission, removal is done by batch (although again, if only one SLA is to be removed, the batch may consist of just that SLA).

When the QME is requested to remove a set of SLAs, it first releases the resources being consumed by those SLAs, and moves them from the *active* SLA list to the *completed* SLA list. The QME then determines any neighbors to the SLAs being removed, and performs an adaptation using these SLAs. In this way, any SLAs close to those being removed have the opportunity to adjust to the removal.

5.5 The Graphical User Interface

SLAOpt provides a graphical user interface, or GUI, for interaction with the user. The GUI is written in Java using Sun's Swing 1.1.1 UI toolkit; Figure 5-1 in Section 5.1 shows the GUI. The GUI allows the user to control SLAOpt's behaviour, to observe the state of the network, admit, change, and remove SLAs from the system, and to modify network resources (such as link capacity).

The GUI comprises several parts; these are the *Network Canvas,* the *Link Table,* the *SLA Table* and *Inspector*, the *Admission Queue*, and the *Utility Monitor.*

The *Network Canvas* displays the topology and state of the network. It uses color-coded links to give a general impression of link usage – from green, to indicate a free link, to

---

[31] This will be feasible if the SLA has followed the suggestions outlined in section 3.6.

yellow, to red to indicate a very congested link. In addition, the Network Canvas highlights the current routing of the selected SLA.

The *Link Table* window shows all links in the current network, and allows the user to edit the capacity and cost of each of these links.

Similarly, the *SLA Table* shows the SLA Directory's *admitted* SLA list; a menu option allows the user to view the *completed* and *rejected* lists as well.

The *SLA Inspector* shows more detail about the selected SLA, including current routing, restriction information, and QoS levels. It also allows the user to edit the SLA's QoS requirements, and to remove the SLA from the system.

The *Admission Queue* allows the user to create a new batch of SLAs for admission. The user can create any number of SLAs before deciding to admit the batch. The admission queue can also load XML-formatted admission requests from a file (see section 5.6, below).

Finally, the *Utility Monitor* shows a simple thermometer-style gauge of total system utility.

5.6 The SLA Interface

The SLA interface is the interface through which remote SLA agents make requests for admission, SLA change and removal known to SLAOpt. It is implemented using XML-formatted text messages over standard UNIX sockets.

The SLA interface consists of the message *server*, and multiple message *handlers*. The message server accepts an incoming connection[32]. It then spawns a new message handling thread, to which it passes the connection, and returns to listening on the socket. The message handlers accept XML-formatted messages from the SLA agents[33]. These messages can be of four types: *admit, remove*, *changeQOS*, and *addQOS*. These correspond to admission requests, removal requests, SLA QoS level change requests and requests to add QoS levels to an SLA respectively. Each message can refer to multiple SLAs. The message handlers then invoke Sun's Java API for XML Parsing (JAXP) to read the XML document. After the document has been deciphered, the message handlers formulate the desired request (including creating or finding SLAs) and call the appropriate method of the QME.

---

[32] The server defaults to port 5060; this may be changed before starting SLAOpt.

[33] The DTD for the message document is given in Appendix A.

6. Some Results

6.1 Implementation Experiments

In this section we first describe several small tests performed to demonstrate that the implementation properly follows the algorithm description. We then give preliminary performance results. As for tests, we propose four simple experiments, each designed to demonstrate the execution of a different phase of the algorithm:

- First, we place a single SLA into a network where capacity is readily available; i.e. an empty network. We also show that in this case the algorithm will select the best possible route.
- Second we place a single SLA into an empty network. We then modify the constraints on the SLA's route (either by changing the SLA's QOS parameters or by modifying a link on the route) such that the SLA is no longer feasible along that route and confirm that SLAOpt returns this result.
- Third we attempt to place a single SLA into a populated (non-empty) network, chosen such that to place the SLA will require re-arrangement of the currently existing SLAs
- Finally we attempt to place a single SLA into a network where sufficient capacity is not available, and expect rejection of the SLA.

6.1.1 Network Description

For the tests described we use a simple nine-node network; this is loosely based on UUNet's backbone [HOBB00]. The network has nodes at Vancouver, San Francisco, Los Angeles, Chicago, Topeka, Amarillo, Boston, New York, and Washington DC. Each link is identical and is given a token capacity of 100,000 and a latency of 0.1.

The network as implemented in SLAOpt is shown below:

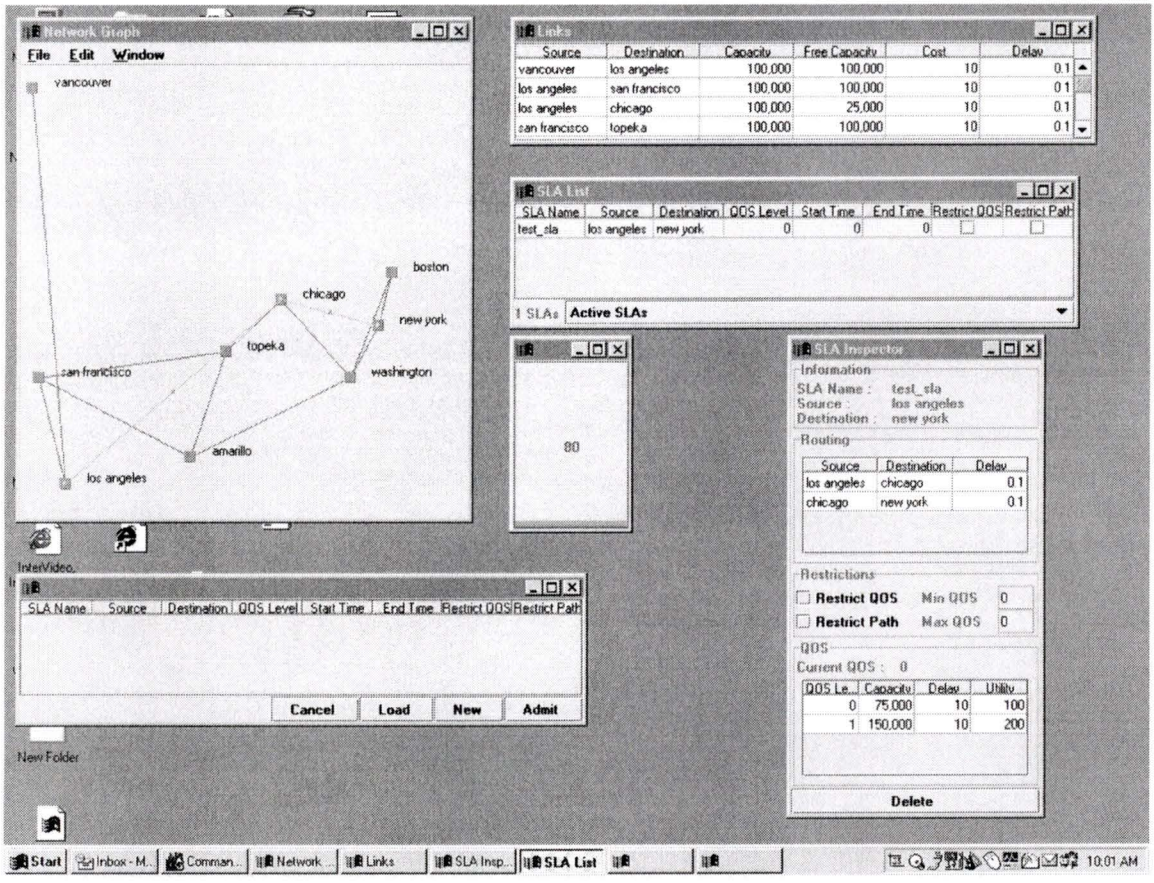


Figure 6-1. The Nine Node Network with One SLA

6.1.2 Simple SLA placement

We first place a single SLA into an empty network. For this degenerate case we select the following parameters:

Endpoints	Los Angeles, New York	
	Q_1	**Q_2**
Capacities	75,000	150,000
Latencies	1.0	1.0

We can readily see that this SLA should be feasible at its first or lowest QoS level (although not at its second, due to capacity constraints). It should also be feasible due to latency on any given path between Los Angeles and New York, since there are only nine nodes in the network, the latency of a Hamiltonian path (if one exists) would be 0.8.

Since QoS level Q_2 is infeasible, we can concentrate solely on Q_1. We know that given paths of length n and m, where there is no opportunity to reduce aggregate resource consumption, the selected path will be that path that minimizes aggregate resource usage per dollar of cost. With all links equal and with constant revenue we expect that this will be the shortest path, (Los Angeles -> Chicago -> New York) We expect then that NHEU will select this path.

Placing the SLA results in the program output shown above, confirming our expectations.

6.1.3 Modification of an SLA's parameters

In order to force the SLA allocated in the previous configuration to be re-routed, we now modify the network.

We modify the capacity of the Chicago <-> New York link to 0; the effect of this is to take down the link. Since our SLA is currently using this link it is affected; furthermore it is now infeasible. The model then performs re-adaptation of all infeasible SLAs (which in this case consists solely of our single SLA).

After re-adaptation, the SLA is now using an alternate path as shown below. Our revenue has dropped slightly due to the increased cost; however, our SLA's session is maintained.

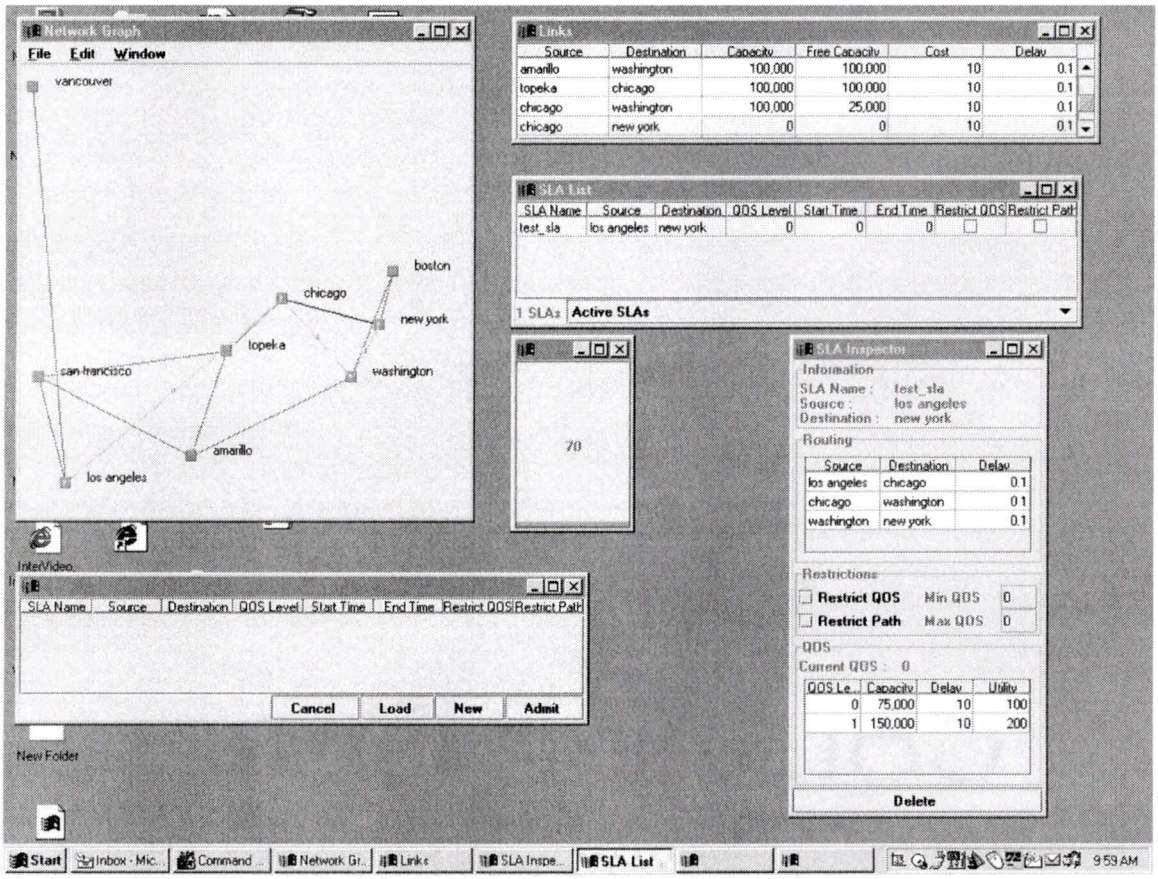


Figure 6-1. Nine Node Network – modified Parameters

6.2 Hundred Node Network

Each node is connected to its neighbours one, five, ten and twenty-five hops away to each side, so the degree of each node is 8. All links of the network are identical and have the following parameters:

- Capacity of 100,000 units
- Delay of 0.1
- Flat cost of 10 units per use

The network is shown in SLAOpt below:

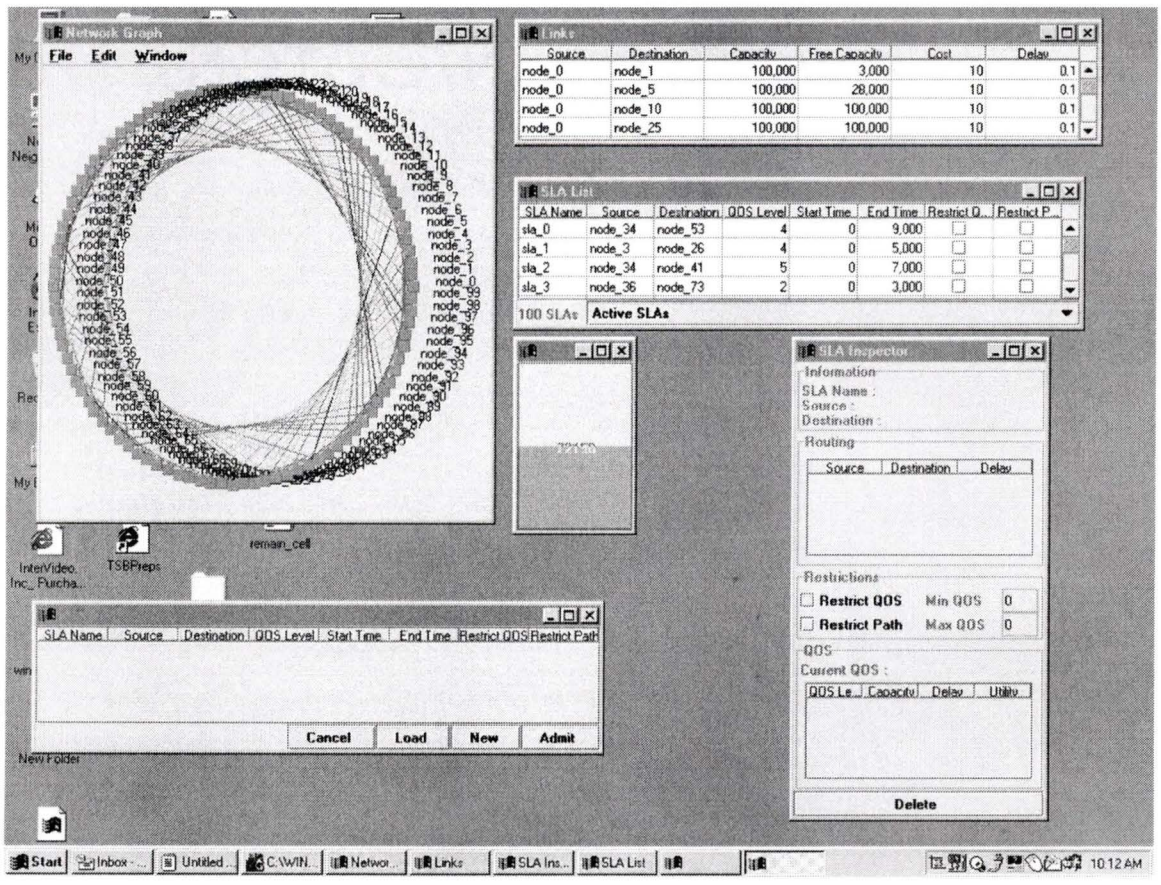


Figure 6-1. One-hundred Node Network

This network has an average path length of 3 hops (0.3 delay; cost of 30 units) for the shortest path between any two nodes. We created a simple Perl program to generate a batch of one hundred SLAs, with the following specifications:

Each SLA is for a random source-destination pair (with source ≠ destination). Each SLA has between 2 and 6 QoS upgrades, randomly determined. The QoS levels start with bandwidth requirements of between 1,000 and 50,000 units, delay requirements between 1.0 and 6.0, and utility values between 100 and 1000 units. For each higher QoS level:

- bandwidth is increased by between 1,000 and 10,000 units
- delay requirement is decreased by between 0.0 and 0.2, and
- utility is increased by either 50 or 100 units.

The expected revenue, if all SLAs achieve their highest QoS level, is thus 77,500 units[34]. Since the shortest paths have cost of 30 units each, we have an expected *maximum* profit of 74,500 units for this network, although this will vary with the particular SLAs chosen.

Since this network graph has very high connectivity, the selection of k for the maximum number of shortest paths is crucial, as it affects the timing of both the k^{th} shortest path algorithm and of NHEU. When k is unbounded, admitting 100 SLAs takes a sufficiently long time that it was not tested. On the test platform, with $k = 50$, SLAOpt will admit a batch of 100 SLAs in about 50 seconds; when k is decreased, this time decreases substantially. When $k = 5$, SLAOpt can place the same SLAs in about 6 seconds. All tests were performed on a Sony VAIO Pentium II, 275 MHz with 64 Mbytes of primary memory, running Windows 98SE with Microsoft's JVM version 5.

k	Elapsed Time	SLAs admitted	Total Utility	Notes
5	6 s	100	77110	
10	14 s	99	77370	SLA 31 rejected
15	19 s	100	77260	
20	21s	99	77380	SLA 23 rejected
25	24s	100	77440	
30	27s	100	77460	
35	36s	99	77340	SLA 23 rejected
40	41s	100	77510	
45	46s	100	77420	
50	48s	100	77490	

Figure 6-2. Results of admission of 100 SLAs into the 100-node network

[34] Given a highest expected QoS level of 4, the expected utility at this level per SLA will be 550 + 4 (75) = 850

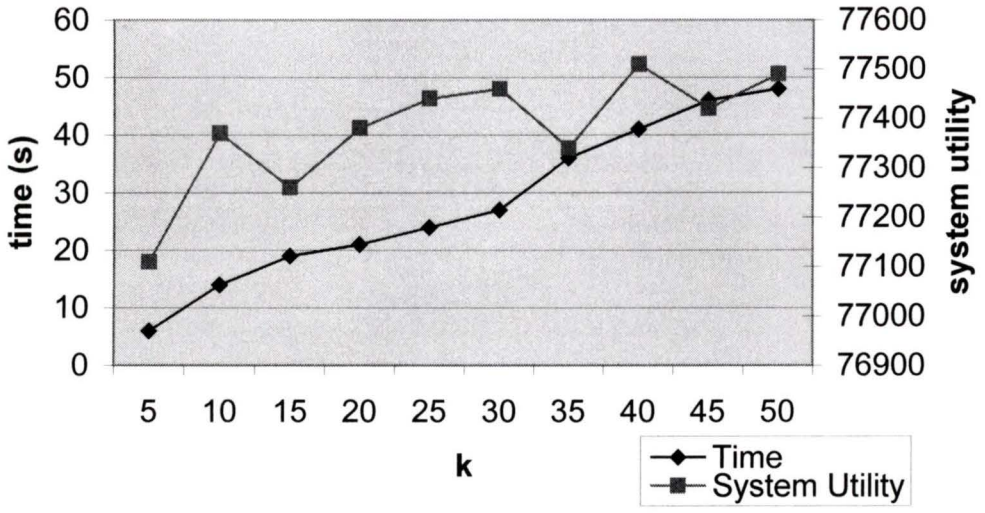


Figure 6-3. Time and System Utility vs. k *for admission of 100 SLAs*

We observe a small but noticeable upwards progression in total utility as k increases, although there are various local minima and maxima. In three cases, one SLA was rejected as unprofitable; examination of SLAs 23 and 31 shows them to have high bandwidth requirements (49,000 units and greater), small delay requirements (≥ 1.7) and small utility values ($100-200 for SLA 23, $100-150 for SLA 31). We note that in this case the selection of SLAs is such that the maximum revenue is actually higher than the expected value. This will be due to the paths being selected averaging less than the expected value of three hops.

We determine a random set of SLAs s with known maximum revenue $U = 75{,}800$ units, and hence maximum net revenue of 72,800 units. With $k = 20$, SLAOpt places all 100 SLAs and achieves net revenue of 72,150 units, or 99.1% of estimated optimal.

Repeating this procedure twenty times gives the following results:

Trial	Max. Revenue	Max. Net Revenue	SLAs placed	Utility	%
0	76400	73400	100	73110	99.6
1	75300	72300	100	71680	99.1
2	74500	71500	100	70660	98.8
3	81400	78400	100	77130	98.3
4	83750	80750	100	79890	98.9
5	78400	75400	100	74210	98.4
6	73850	70850	100	69890	98.6
7	76350	73350	100	73020	99.6
8	85150	82150	97	80330	97.8
9[35]	83250	80250	100	80280	100.0
10	78350	75350	100	74730	99.1
11	79350	76350	100	75540	98.9
12	81350	78350	99	77380	98.8
13	77300	74300	100	74160	99.8
14	77050	74050	100	73730	99.6
15	76700	73700	100	72850	98.8
16	75700	72700	100	71170	97.9
17	77550	74550	100	74170	99.5
18	80650	77650	100	77310	99.6
19	75800	72800	100	72150	99.1

Figure 6-4. Net revenue comparisons for twenty trials.

[35] SLAs arranged such that the average shortest path is less than 3.0 hops allow for the possibility of utility greater than expected maximum revenue, as with the above example.

7. Conclusions and Future Work

7.1 Conclusions

We have adapted Khan's algorithm for optimal admission of customers to a server while fully respecting QOS constraints, to the admission of flexible SLAs to a switched data network. We have dealt with several difficulties in doing so. We have implemented our result – algorithm N-HEU as the prototype optimal admission control SLAOpt for IP datagram networks. We have demonstrated the functioning of SLAOpt and have obtained a few preliminary performance numbers. We are quite satisfied with our preliminary results and feel that future work is warranted to better explore the model.

7.2 Further Work

7.2.1 Integration with an active network simulator

Currently SLA-OPT works in a rather static environment. The best example of this is the delay model, which assumes no variation in delay due to traffic. It would be useful to integrate SLA-OPT with a network simulator (such as OpNet or Nortel's Nine Node simulator) . This would allow us to admit SLAs, then, using traffic generators, generate appropriate traffic for these SLAs on the network, and obtain values of queuing delays by simulation of the traffic in the network. This, in turn, should allow us to make more realistic decisions on routing and admission of future SLAs, and on adaptation of SLAs currently in the network. We might also replace the conservative and expensive peak bandwidth reservation with more realistic estimators of actual bandwidth required. Here we have to consider just how hard we wish our hard guarantees of QoS to be.

We can also elegantly unify SLAOpt with recent work by Kadengal et al [KADE00] on path-by-path congestion control based on a congestion-based link price. Primarily, we have to allow modules implementing his algorithms to dynamically update the link price data used by SLAOpt.

7.2.2 Performance Studies

Extensive performance evaluation of SLAOpt including the effect of tuning the main parameters, need to be done.

7.2.3 Holding times of virtual circuits

Following Khan we do not concern ourselves with the holding times of sessions in particular, or with the dimension of time in general. For cases where the holding times of all sessions are similar, this may be realistic.

However, suppose there exist only two sessions S_1 and S_2, with holding times τ_1 and τ_2. If $\tau_1 << \tau_2$, then unless the net revenue p_1 is much greater than p_2, it is in our interest to accept p_2 over p_1. Over time, it will provide greater revenue.

Plotkin *et. al.* [AWER94] extend their work on routing permanent virtual circuits (with holding times of ∞) to those with predetermined holding times by "flattening" the time; that is, by duplicating the network for each time unit and maximizing (or minimizing) over the lifetime of the session.

We can also look at simpler cases, such as making resources before reservations available only to SLAs with holding times terminating before reservation begins.

7.2.4 Protection priorities during failures

We wish to extend the operation of the algorithm during times of failure to allow for the possibility of certain SLAs receiving priority over other SLAs. This may be viewed in terms of an additional service or resource requirement. A possible solution would be to downgrade in stages, beginning with those SLAs with the least protection priority, until the network is once again feasible.

References

[AWER93] B Awerbuch, Y. Azar, & S. Plotkin, "Throughput Competitive On-line Routing", Proc. 34th Annual Symp. on Foundations Comp. Sci., IEEE, Los Alamitos, CA, Nov 1993

[AWER94] B. Awerbuch, Y. Azar, S. Plotkin, O. Waarts, "Competitive Routing of Virtual Circuits with Unknown Duration", Proc. of the Fifth Annual SIAM Symposium on Discrete Algorithms, Arlington, VA, pp.321-327

[BENE65] V.E. Benes, "Mathematical Theory of Connecting Networks and Telephone Traffic", New York : Academic Press, 1965

[BIDD99] M.D. Biddiscombe, J.E. Midwinter, & S. Sabean, "Application of free-marked principles to telecoms resource allocation" Electronics Letters v35 n4 Feb. 1999, IEEE, Stevenage, pp, 264-266.

[BORO98] A. Borodin & R. El-Yaniv, "Online Computation & Competitive Analysis", Cambridge University Press, 1998, ISBN 0-521-56392-5

[CHAN96] E. Chang & A. Zakhor, "Cost Analysis for VBR video servers", IEEE Multimedia, Winter 1996, pp. 56-71

[CHEN98] L. Chen, "The Utility Model Applied to Layer Coded Sources", M.Sc Thesis, University of Victoria, Department of Computer Science, 1998

[CHEN99] L. Chen, K.F. Li, S. Khan, E. Manning, "Building an Adaptive Multimedia System Using the Utility Model." Parallel & Distributed Processing, Lecture Notes in Computer Science 1586, Springer Verlag, pp. 289-298, ISBN 3-540-65831-9

[DAVI94] N. Davies et. al. "Supporting Adaptive Service in a Heterogeneous Mobile Environment", 1994 Workshop on Mobile Computing, Santa Cruz

[DIJK59] E.. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, 1959, pp. 269-271

[EPPS94] D. Eppstein, "Finding the *k* Shortest Paths", 35th IEEE Symp. Foundations of Computer Science, IEEE, Santa Fe, pp. 154-165

[HOBB00] C. Hobbes, personal communication.

[ITUT95] ITU-T I.150 "B-ISDN Asynchronous Traffic Mode Functional Characteristics"

[ITUT96] ITU-T I.371 "Traffic control and congestion control in B-ISDN"

[KHAN97] S. Khan, "Quality Adaptation in a multisession multimedia system: Model, algorithms, and architecture", Ph.D.. Thesis, University of Victoria, Department of Electrical and Computer Engineering, 1998.

[KHAN98] S. Khan, E. Manning, & K. F. Li, "The Utility Model for Adaptive Multimedia Systems", Proc. Intl. Conf. Multimedia Modeling, Singapore, Jan. 1998, World Scientific Publishing Ltd., ISBN 981-02-3351-5, pp. 111-126

[KIRK99] Paul Kirkby & R. Kadengal, "Traffic Management & Control Using a Single 'Congestion Price'", Proc. IEEE Colloquium on Control of Next Generation Networks, IEEE, London, October 1999.

[KLEI99] J. Kleinburg, Y. Rabani, & E. Tardos, "Allocating Bandwith for Bursty Connections", Conf. Proc. of the Annual ACM Symposium on Theory of Computing, May 1997, El. Paso, TX, pp. 664-673.

[KUMA] V.P. Kumar, T.V. Lakshman, D. Stiliadis, "Beyond Best Effort : Router Architectures for the Differentiated Services of Tomorrow's Internet". Bell Laboratories (Lucent Technologies).

[MART97] E.Q.V. Martins, M. Pascoal, J.L.E. Santos, "A New Algorithm for Ranking Loopless Paths", Departamento de Matematica, Universidade de Coimbra, May 1997. http://www.mat.uc.pt/~eqvm

[MART96] E.Q.V. Martins, J.L.E. Santos, "A New Shortest Paths Ranking Algorithm". Departamento de Matematica, Universidade de Coimbra, 1996,. http://www.mat.uc.pt/~eqvm

[STAL94] W. Stallings, "Data and Computer Communications", Toronto : Macmillan 1994.

[TARD99] J. Kleinburg & E. Tardos, "Approximations for the Disjoint Paths Problem in High-Diameter Planar Networks", Journal of Computer and System Services v57 n1 Aug 1998, Orlando, pp. 61-73.

[TOBE98] Y. Tobe. & H. Tokuda, "Integrated QoS Management : Cooperation of Processor Capacity Reserves and Traffic Management", IEICE Transactions on Communications vE81-B n11 Nov 1998, Tokyo, pp. 1998-2006.

[TOKU94] C. W. Mercer, S. Savage, & H. Tokuda, "Processor Capacity Reserves", IEEE Conf. Multimedia Comp. & Sys., Boston, pp. 90-99.

[TOYO75] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one integer programming", Management Science 21, 1975 pp. 1417-1427.

[TSUI98] I.R. Chen & T.H Tsui, "Performance Analysis of Admission Control Algorithms based on Reward Optimization", Performance Evaluation, Vol. 33 No. 2, pp. 89-112

[VINA94] A. Vina et. al., "Real-time Multimedia Systems", 19th IEEE Symp. Mass Storage Sys., IEEE, 1994, pp. 77-83.

[YEN71] J.Y. Yen, "Finding the K shortest loopless paths in a network", Management Science 17, 1971, pp. 712-716

Vita

Surname: Watson Given Names: Robert

Place of Birth: Port Alberni, British Columbia

Educational Institutions Attended:

University of Victoria (1993-2000)

Degrees Awarded:

Bachelor of Engineering (University of Victoria, 1998)

Partial Copyright Licence

Title of Thesis:

The Optimal Admission & Adaptation of Service Level Agreements in Packet Networks :

Applying the Utility Model

Author: [redacted]

Robert Kristian Watson

September 27, 2001