The Fourth Advanced International Conference on Telecommunications

# Towards Efficient Service Placement and Server Selection for Large-Scale Deployments

Jeroen Famaey Tim Wauters Filip De Turck Bart Dhoedt
Piet Demeester
Department of Information Technology (INTEC), Ghent University - IMEC - IBBT,
Gaston Crommenlaan 8, bus 201 , B-9050 Ghent, Belgium
jeroen.famaey@intec.ugent.be

## Abstract

*Currently many service providers offer their services using a private and proprietary hard- and software infrastructure. These infrastructures often share many similarities. Hence we believe a generic service delivery architecture, that allows service providers to offer a large array of different services on a shared infrastructure, would provide many advantages over current silo-based approaches.*

*In this paper we propose the first step towards such an architecture, namely several algorithms for dynamically allocating server and network resources to a set of services and selecting a suitable service instance for each client. Service instances are placed on a set of servers, taking into account network resources (available bandwidth), server resources (CPU and memory) and service Quality of Service (QoS) demands (maximum transmission delay and bandwidth requirements). The optimization goal is to maximize the percentage of satisfied demand (answered requests) and minimize the total number of used servers for service hosting. Each service has a relative priority, which can be manually adjusted to influence the placement scheme.*

## 1 Introduction

The exponential growth of the Internet allows an ever-growing number of service providers to reach more and more users. Many services require a similar hard- and soft-ware infrastructure that offers scalability, resilience, QoS, multicast, lifecycle management, etc. Currently most providers maintain a private and proprietary infrastructure specifically designed for a single service. As an alternative we propose a generic service delivery architecture. Such an architecture would allow many service providers to offer their services on a single, shared infrastructure. This approach has several advantages for both service providers and users. First, the investment threshold to start offering services will be much lower for both new providers and private users. Second, providers will be able to focus their efforts on designing and implementing the service logic instead of the service delivery platform. And finally, a distributed and shared infrastructure would span a much larger area, allowing service providers to offer their services closer to their users, thus improving transmission delays.

In this paper we discuss the first step towards such an architecture, namely offline, centralized algorithms for service placement and server selection. The algorithms take into account two server resources, one load-dependent (in this case CPU) and one load-independent (in this case memory). We treat memory as being independent of the current load because many applications consume a conciderable amount of their memory even if they are not processing requests. Also current memory usage may depend on past load because of caching.

Unlike many existing service placement algorithms, ours also use network-related factors when making service placement decisions. These factors include available edge bandwidth and transmission delay.

Currently many networks offer no end-to-end support for QoS demands, resilience and multicasting (e.g. the Internet). They can nevertheless be offered by defining and adjusting routing paths. On a network such as the Internet routing cannot be directly controlled, but limited control can be obtained by using an overlay network [4]. The overlay network consists of a set of overlay servers (OS) which are linked together in a topology consisting of virtual overlay edges. Each such edge corresponds to the path in the underlying physical network. Although the mapping between overlay edges and underlying paths is determined by the routing policies of the traversed autonomous systems (AS), the routing path on the overlay layer can be controlled. In the past this technique has already been used to offer re-

978-0-7695-3162-5/08 $25.00 © 2008 IEEE
DOI 10.1109/AICT.2008.46

silience [3], improved QoS [7] and overlay multicasting [5].

In Section 5 we study the effect of randomly generated overlay topologies of different sizes on the algorithms. We will use these results as a basis for intelligent overlay topology construction and adaptation algorithms in future work. It should be noted that many current overlay-based approaches lack scalability, and use a static topology. Therefore much work remains to be done in this area.

The rest of this paper is structured as follows. We discuss related work in Section 2. The service placement and server selection problems are formally described in Section 3. Section 4 gives a summary of the designed algorithms. They are evaluated using simulation results in Section 5. And finally, conclusions are drawn and future work is briefly discussed in Section 6.

## 2 Related Work

Karve et al. [6] designed a centralized application placement middleware. Their heuristic maximizes satisfied demand and minimizes total number of placement changes compared to a previous placement scheme. They only take into account server resources, namely CPU and memory.

Adam et al. argued that any centralized service placement scheme has a limited scalability and created a decentralized variant of [6]. In [1] they propose an application placement middleware that constructs overlays, places services, selects service instances for clients and routes the client requests to the correct instance. Their service placement heuristic maximizes satisfied demand, also taking into account only CPU and memory constraints. In [2] the scalability of the design in [1] was improved. Here every node requires knowledge on only a limit number of other nodes, instead of global knowledge on all nodes in the network.

Although these designs take into account several server resources, none of them consider network related resources and limitations. We argue that in this age, where many applications require large amounts of bandwidth (e.g. video streaming, IPTV) and guarantees concerning transmission delay (e.g. online gaming, video/audio conferencing) these are just as important as server resources. Therefore our algorithms also concider bandwidth and transmission delay.

## 3 Problem Formulation

The service placement and server selection problems can be described as follows. Given a directed graph $G(N, E)$ denoting the overlay topology, with $N$ a set of overlay nodes (both servers and clients) and $E$ a set of directed overlay edges. $M$ denotes the set of available services. Every service $m \in M$ has a set $N_m \subseteq N$ of nodes which have requests for it (clients). The two dimensional matrix $R$ is called the request-matrix, $R(m, c)$ gives the number of requests per second of client $c \in N_m$ for service $m \in M$.

Every node $n \in N$ has a memory capacity $\Gamma_n$ and a CPU capacity $\Omega_n$. Every edge $e \in E$ has a bandwidth capacity $B_e$ and transmission delay $\Delta_e$. Every service $m \in M$ has a memory requirement $\gamma_m$ per instance, CPU requirement $\omega_m$ per request, bandwidth requirement $\beta^i_m$ per request and $\beta^o_m$ per reply, a maximum allowed transmission delay $\delta_m$, and a priority $\pi_m$.

In a heterogenous environment, such as the Internet, not every server is able to run every service. Individual services might have extra requirements, such as a certain operating system, or software library. These extra requirements are modelled using a capabilities-matrix $C$. $C(m, n) = 1$ if node $n \in N$ is capable of running service $m \in M$.

Service placement entails deciding on which nodes to place instances of which services, while satisfying several constraints. Server selection comes down to selecting which instance of a service to use to answer a given client's requests, while satisfying a number of constraints.

Before listing the constraints we define a few extra decision variables. $s_{m,n} = 1$ if node $n \in N$ hosts an instance of service $m \in M$, 0 otherwise. $a_{m,n,c}$ defines whether or not the instance of service $m \in M$ on node $n \in N$ answers the requests of client $c \in N_m$. $p^i_{m,n,e,c} = 1$ if edge $e \in E$ is part of the routing path from client $c \in N_m$ to server $n \in N$ for service $m \in M$. $p^o_{m,n,e,c}$ has the same meaning, but for the path from the server to the client. $u_n = 1$ if node $n \in N$ hosts 1 or more services, 0 otherwise. Using these decision variables the constraints are

$$\forall e \in E : \sum_{m \in M} \sum_{n \in N} \sum_{c \in N_m} (p^i_{m,n,e,c} \cdot \beta^i_m + p^o_{m,n,e,c} \cdot \beta^o_m) \cdot R(m, c) \leq B_e \quad (1)$$

$$\forall m \in M, \forall n \in N, \forall c \in N_m : \sum_{e \in E} (p^i_{m,n,e,c} + p^o_{m,n,e,c}) \cdot \Delta_e \leq \delta_m \quad (2)$$

$$\forall n \in N : \sum_{m \in M} \sum_{c \in N_m} a_{m,n,c} \cdot \omega_m \cdot R(m, c) \leq \Omega_n \quad (3)$$

$$\forall n \in N : \sum_{m \in M} s_{m,n} \cdot \gamma_m \leq \Gamma_n \quad (4)$$

$$\forall m \in M, \forall n \in N : s_{m,n} \leq C(m, n) \quad (5)$$

Equation 1 stipulates that the total used bandwidth on every edge cannot exceed the available bandwidth on that edge. Equation 2 stipulates that the delay of the path from a client to the server hosting its instance of a specific service and back cannot exceed the delay bound of that service. Equation 3 denotes that the total CPU used to answer

all requests for the service instances on a node, cannot exceed the available CPU on that node. Equation 4 denotes the same for the memory used by all service instances on a node. Finally Equation 5 stipulates that a node can only host services if it meets all requirements of that service.

The objective function, which the algorithms attempt to optimize is given as follows

$$\max \sum_{n \in N} ((\sum_{m \in M} \sum_{c \in N_m} \alpha_1 \cdot \pi_m \cdot a_{m,n,c} \cdot R(m,c)) - \alpha_2 \cdot u_n)$$

The $\alpha_1$ and $\alpha_2$ parameters denote the relative importance of both objectives. The first objective, which is maximized, equals the total satisfied demand, multiplied by the priority of each service. This effectively prioritizes services with a higher priority. The second objective, which is minimized, equals the total number of servers used to host services. We believe it is important to use as few resources (in this case servers) as possible, while still optimizing the main objective. This allows us to use these nodes for other purposes, or switch them off.

# 4 Service Placement Algorithms

We have designed 3 algorithms that solve the problem described in Section 3. The first algorithm is based on the Integer Linear Programming (ILP) [10] formulation and finds the optimal solution. The problem can be formulated as a variant of the Class Constrained Multiple-Knapsack Problem, which has been shown to be NP-hard [8]. Therefore this algorithm scales very poorly in terms of number of servers, users and services. To improve scalability we devised two heuristics, which find a sub-optimal solution in polynomial time.

## 4.1 ILP Algorithm ($ILP$)

The ILP formulation is based on the formal description of the problem given in Section 3. The algorithm maximizes the given objective function by selecting suitable values for the decision variables, using branch and bound techniques. For the implementation we used the ILOG CPLEX 10.0 software package. Note that not all constraints needed for a correct ILP formulation are given in the problem description. Due to space limitations, constraints defining the relationship between different decision variables have been omitted.

## 4.2 Greedy Heuristic ($Gr$)

As its name implies this heuristic is based on the greedy principle. It is greedy in the way that it will seperately place instances of each service, without taking into account services that have not been placed yet (it does however take into account resources used by already placed services).

Before executing the algorithm, the services should be sorted from small to large according to a certain metric. Experimentation showed that sorting services from most to least requests multiplied by priority ($\pi_m \cdot \sum_{c \in N_m} R(m,c)$), performed well. The algorithm can be described as follows

1. Select the next service $m$ in the sorted list of services.
2. For each client $c \in N_m$ create a list of possible candidate servers to host an instance of $m$ for it. Only include servers that are capable of running service $m$, have enough available CPU and memory, and have an overlay path to and from $c$ with enough available bandwidth and a low enough delay (do not take into account resources potentially used by other clients of $m$).
3. Sort the servers from most to least possible clients (the server that occurs on the most candidate lists comes first). Also include clients of previously placed services in this count. Sort the clients from least to most candidate servers (this will also give clients with only few possible candidates a chance).
4. Select the next server $s$ in the sorted list.
5. For each client $c$ in the sorted list for which no candidate has been selected yet. Check if $s$ has enough remaining memory and CPU and if a path can be found from $c$ to $s$ and back with a low enough delay and enough remaining bandwidth (take into account resources used by previously placed clients for service $m$ now). If any clients remain without candidate and not all servers have been checked yet, return to step 4.
6. If any services remain, goto step 1, otherwise finish.

It can be easily shown that this algorithm has a worst-case time complexity of $O(m \cdot c \cdot n^3)$. With $m$ the number of services, $c$ the maximum number of clients for a single service, and $n$ the number of overlay nodes.

## 4.3 Selective Greedy Heuristic ($SGr$)

Because of its design, it would be very difficult to use the standard greedy heuristic in a decentralized architecture. To solve this problem we have designed another heuristic, also based on the greedy principles. Before running the algorithm, services are sorted using the same metric as used for $Gr$. The algorithm can be described as follows

1. Select the next service $m$ in the sorted list of services.
2. Select the next client $c \in N_m$.
3. Select the $K$ servers with shortest delay to $c$. If one or more of these servers are already running an instance of $m$, then prioritize them over the others. Select the server which is the candidate for most clients

already and has enough CPU and memory resources. Make sure the path from $c$ to the server and back has enough available bandwidth and low enough delay. If any clients for service $m$ remain, goto step 2.

4. If any services remain goto step 1, otherwise finish.

This algorithm has two advantages, which make it easier to convert to a decentralized form. First, for each client it checks only those servers nearest to it. And second, when selecting a candidate for a client, only past placement information is used. $Gr$ uses information about all clients of a service, even when their candidate has not yet been chosen.

It can easily be shown that $SGr$ has a worst-case time complexity of $O(m \cdot c \cdot K \cdot n^2)$. This is better than the complexity of $Gr$ when $K < n$.

# 5 Evaluation

In this section we evaluate the performance and scalability of the heuristics. Performance is measured by comparing the percentage of satisfied demand and number of used servers to the optimal (calculated using the ILP algorithm). Scalability is evaluated by comparing results for different sized networks and service sets. Additionally we have performed a third simulation to examine the influence of the overlay topology on the solution.

When relevant we used the S-PLUS 8.0 software package to statistically interpret simulation results. All statistical tests were performed using a 5% significance level. We used the one-way analysis of variance (ANOVA) test to compare several levels of a single factor. If an effect of the factor was found, we used a Tukey test to detect differences between individual averages.

## 5.1 Simulation Setup

The tests were performed on randomly generated network topologies. The physical underlays were created using Waxman's algorithm [9]. For the first 2 tests no overlay network was needed, here the overlay topology was identical to the generated underlay. For the third test an overlay network was generated on top of the underlay. The overlay nodes were randomly selected among the underlying network nodes. Overlay edges were then generated to connect these nodes, until a predefined average node degree was reached. For each overlay edge the corresponding underlying path was set to the shortest hop-count path. Clients were selected at random out of the set of overlay nodes.

## 5.2 Performance of the Heuristics

The goal of this test is to compare the solution of the heuristics to the global optimum. This optimum was calculated using the ILP algorithm. The test was performed on networks with 30 nodes (of which 10 acted as clients) with an average in- and outdegree of 3. The clients had requests for 3 different randomly generated services.

Fig. 1 shows the results as a function of available CPU (%) and service delay bound (db). Each client is connected to a single server via a zero-delay edge, all other edges have delay 1. By limiting the delay bound to 0, clients can thus only be served by the server they are connected to directly.

As expected the available CPU is directly proportional to the satisfied demand (%) (measured as percentage of answered requests). Up to about 40% available CPU also influences the number of used servers to host instances. This is most likely because at lower values, some servers do not have enough CPU available to serve even a single client.

The service delay bound is inversely proportional to the number of used servers. As the figure shows, delay bound has only a limited influence on the percentage of satisfied demand. Statistical analysis showed that there is no significant difference between satisfied demands for delay bounds 2 and 4, for any available CPU percentage, except for 40 and 80%. Satisfied demand for delay bound 0, on the other hand, is significantly different from results for delay bounds 2 and 4 (for all algorithms and any available CPU percentage, except 100%).

Statistical analysis showed that there are no significant differences between the 3 algorithms, in terms of number of used servers, for any available CPU percentage or any service delay bound. For satisfied demand there are only significant differences between $SGr$ and $ILP$ for available CPU percentage 90% (delay bound 4) and 100% (delay bounds 2 and 4). Between $Gr$ and $ILP$ there is only a significant difference for 100% available CPU (delay bound 2).

## 5.3 Scalability

In this section we study the scalability of the heuristics, both in terms of satisfied demand and execution time. Tests were performed using different sizes networks. For networks with $x$ nodes, $\frac{2x}{3}$ of them acted as clients and $\frac{x}{6}$ services were randomly generated. We also varied the maximum number of candidate servers $K$ of the $SGr$ algorithm. This allows us to study the trade-off between solution quality and execution time, for varying $K$. From here on we denote the $K$ value used for $SGr$ between brackets. So $SGr(n)$ means $SGr$ with $K = n$.

Fig. 2 shows the results as a function of network (and service set) size. Fig. 2(a) shows clearly that the satisfied demand (%) of $Gr$, $SGr(n)$, and $SGr(2log(n))$ (n being the network size), degenerates only slowly. This is not the case for $SGr(log(n))$. Statistical analysis showed that results for $SGr(log(n))$ are significantly different from the other algorithms starting at 120 nodes. There are no sig-

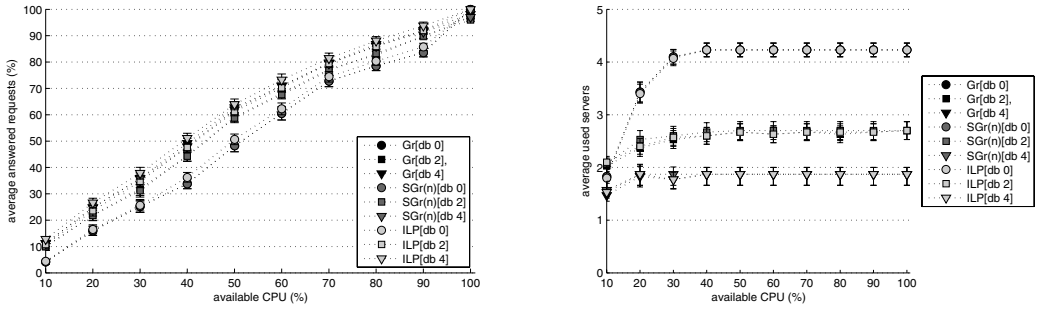


**Figure 1: Performance of the heuristics compared to the optimal algorithm as a function of available CPU (%) and service delay bound, in terms of satisfied demand (%) (left graph) and servers used to host services (right graph)**

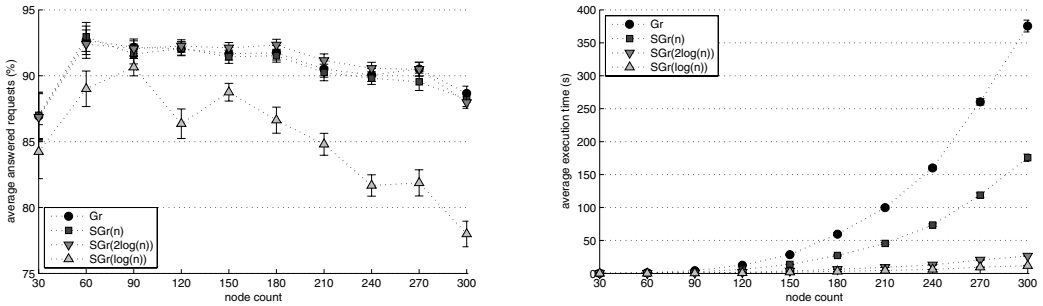


**Figure 2: Scalability of the heuristics as a function of the overlay network size, in terms of satisfied demand (%) (left graph) and execution time (s) (right graph)**

nificant differences between results for node counts 90 to 270 for $Gr$, $SGr(2log(n))$, and $SGr(n)$, and results for node count 300 are only significantly different from those for node counts 30 to 180.

Fig. 2(b) shows scalability in terms of execution time. It is obvious that $Gr$ and $SGr(n)$ scale poorly compared to $SGr(log(n))$ and $SGr(2log(n))$. This is in accordance to the theoretical time complexity, which is respectively proportional to $O(m \cdot c \cdot n^3)$ for the first two algorithms and $O(m \cdot c \cdot log(n) \cdot n^2)$ for the other two.

Looking at both figures $SGr(2log(n))$ seems to be the best choice in terms of scalability. Its solution quality (in terms of satisfied demand) is not significantly different from that of $Gr$ and $SGr(n)$, while it is almost 15 times faster than $Gr$ (in terms of execution time) for larger networks.

## 5.4 Overlay-awareness

As stated in Section 1 overlay networks have several useful advantages for a generic service delivery architecture. In this section we will study the effect of random overlay topologies on the service placement heuristics. The generated physical networks contain 100 nodes. On top of these networks overlay topologies of different sizes were created by selecting from 10 up to 100 nodes at random and adding them to the overlay.

Fig. 3 shows the results. Fig. 3(a) shows the satisfied demand (%) for both heuristics as a function of amount of used overlay servers. We define an overlay server as being used if it is used on the path from 1 or more clients to their candidate server or back for 1 or more services, that does not equal the shortest hopcount path. We use this definition because if the shortest hopcount path is used, this is most likely equal to the routing path that would be used if no overlay was present. As expected, satisfied demand is directly proportional to the size of the overlay network. This is because a larger overlay means more available server and network resources. What is interesting is that the effect of the overlay size on satisfied demand is only minor. Statistical analysis also showed that there are no significant differences between the algorithms for any overlay size. Also there are no significant differences between results for overlay sizes 40 to 100.

Fig. 3(b) shows the number of used overlay servers as a function of available overlay servers. The figure shows several interesting things. First, $SGr$ uses less overlay servers than $Gr$. This is because $SGr$ will search for its $K$ possible server candidates via the shortest delay paths starting

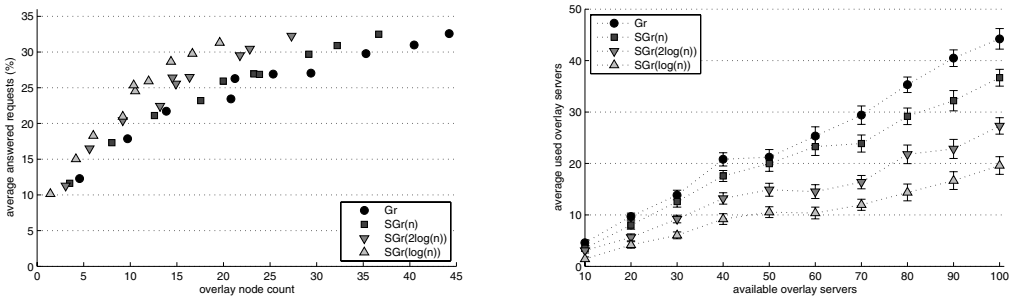


**Figure 3: Performance of the heuristics using a 100 node underlay, the left graph shows satisfied demand (%) as a function of used overlay servers, and the right graph shows used overlay servers as a function of available overlay servers**

from the client. Therefore it will more likely select the underlay path. $Gr$ will potentially select a server far away, lessening the chances it will select the shortest hopcount path. Second, even when 100 overlay servers are available all algorithms will use on average less than 50.

Statistical analysis showed that in terms of used overlay servers $SGr(log(n))$ performs significantly better than $Gr$ and $SGr(n)$, for all overlay sizes. $SGr(2log(n))$ performs significantly better aswell, but only for overlay sizes 40 and upward. Starting at overlay size 70, $SGr(n)$ performs significantly better than $Gr$.

From these results we can conclude that even when all servers are available to be used in the overlay network only a portion of them is actually used. Intelligently selecting these overlay servers would allow us to satisfy more demand with only a portion of all available overlay servers.

## 6 Discussion and Future Work

In this paper we proposed an optimal algorithm and two heuristics to solve the service placement and server selection problems. These form the first step towards an overlay based generic service delivery architecture. Such an architecture could provide several advantages to current and future service providers and end-users compared to existing practices where the hard- and software infrastructure is private and mostly used to offer only a single or a few services.

We showed that in many situations the optimal algorithm does not perform significantly better than our heuristics. Additionally our results show that intelligent overlay topology construction could prove benificial.

In future work we are planning to design an online, decentralized variant of the selective greedy heuristic to improve scalability and reliability. We will also incorporate intelligent overlay topology construction into our design.

## Acknowledgements

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders.

## References

[1] C. Adam, G. Pacifici, M. Spreitzer, R. Stadler, M. Steinder, and C. Tang. A decentralized application placement controller for web applications. Technical report, Royal Institute of Technology (KTH), 2006.

[2] C. Adam, R. Stadler, C. Tang, M. Steinder, and M. Spreitzer. A service middleware that scales in system size and applications. *The 2007 IFIP/IEEE Symposium on Integrated Network Management*, pages 70–79, 2007.

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. *18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, 2001.

[4] D. Clark, B. Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclawski. Overlay networks and the future of the internet. *Communications & Strategies*, 63(3):109–129, 2006.

[5] B. De Vleeschauwer, F. De Turck, B. Dhoedt, and P. Demeester. Online management of QoS enabled overlay multicast services. *Proceedings of IEEE GLOBECOM 2006, The Global Telecommunications Conference*, pages 1–6, 2006.

[6] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. *Proceedings of the 15th International Conference on World Wide Web*, pages 595–604, 2006.

[7] Z. Li and P. Mohapatra. QRON: QoS-aware routing in overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):29–40, 2004.

[8] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.

[9] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communication*, 9(6):1617–1622, 1988.

[10] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.