Dynamic Overlay Node Activation Algorithms for Large-Scale Service Deployments

Jeroen Famaey*, Tim Wauters**, Filip De Turck**, Bart Dhoedt, and Piet Demeester

Department of Information Technology (INTEC), Ghent University - IBBT
Gaston Crommenlaan 8, bus 201, B-9050 Gent, Belgium
jeroen.famaey@intec.ugent.be

Abstract. Due to overprovisioning of infrastructure nodes in overlay networks, many nodes remain idle at times of low network load. Some of these nodes could be temporary removed from the overlay topology and could then be used for other purposes or alternatively be temporarily shut down, to save energy.

In this paper we present several algorithms to select the subset of overlay nodes that should be part of the overlay topology. This decision is made based on the current and (estimated) future network load and the locality of the clients and servers connected to the overlay network. As network load or conditions change, additional nodes can be dynamically (de)activated. Our algorithms can be used in conjunction with existing overlay topology construction protocols.

Through extensive simulations, we have evaluated and compared the performance of our algorithms.

1 Introduction

Recently, peer-to-peer overlay networks have been increasingly used to offer services on the Internet, such as Voice-over-IP (e.g. Skype), filesharing (e.g. Gnutella, BitTorrent), and Internet TV (e.g. Joost, Zattoo).

Although many protocols have been proposed to dynamically create and manage both structured and unstructured overlay topologies in a scalable manner, little attention has been paid to deciding which candidate overlay nodes to actually include in the overlay network. Existing protocols merely include every available candidate without any sort of node activation mechanism. As a consequence, many nodes in overprovisioned overlay networks remain idle. By adding a node activation phase to the overlay construction mechanism, the resources of these idle overlay nodes can be freed. They could be used for other purposes or be temporarily shut down to save energy. Additionally, if included in the overlay, these idle nodes take part in control and management protocols and algorithms

* Funded by the Institute for the Promotion of Innovation by Science and Technology Flanders (IWT-Vlaanderen).

** Funded by the Fund for Scientific Research Flanders (FWO-Vlaanderen).

F. De Turck, W. Kellerer, and G. Kormentzas (Eds.): DSOM 2008, LNCS 5273, pp. 14–27, 2008.

run on top of the overlay network, adding extra control-traffic overhead and negatively influencing overall performance and scalability [1].

In this paper, we propose two heuristics to select a suitable subset of the available overlay nodes, to be used in the overlay topology. The first heuristic is static and centralized, suitable for determining where to place the initial overlay nodes. The second heuristic is fully decentralized and thus suitable for managing large-scale topologies. Additionally, it is capable of dynamically adjusting the subset of activated overlay nodes. To compare the solution of these heuristics to the global optimal solution, we have also implemented an optimal algorithm to solve the problem, based on an Integer Linear Programming (ILP) formulation.

The two heuristics are specifically designed for service hosting overlays. In this case the overlay network consists not only of overlay nodes, which are responsible for routing client requests and selecting services, but also of clients and service hosts. As input, the activation algorithms take into account the demand generated by nearby clients, the demand satisfied by nearby service hosts, and the bandwidth load on incoming and outgoing overlay links. As a result the algorithms return which overlay nodes should be activated but also which overlay node to use as the gateway for each client and service host. The goal is to make sure there are enough resources available in the network to satisfy the client demand, while minimizing the amount of activated overlay nodes.

The rest of this paper is structured as follows. Related work is discussed in Section 2. The node activation and gateway selection problems are formally described in Section 3. Section 4 gives a description of the designed algorithms. An evaluation based on simulation results is presented in Section 5. And finally, a brief summary of our work is given in Section 6.

2 Related Work

In the context of overlay service hosting platforms, most research has been aimed towards service placement and server selection algorithms [2,3], while problems related to overlay construction have received little attention. In the context of general-purpose overlay networks, most research has been done towards constructing overlay topologies, both structured [4] and unstructured [5,6,7]. These protocols include every available overlay node in the topology, instead of selecting a subset of required nodes, based on the current network load. Our node activation algorithms can be used in conjunction with these overlay topology construction protocols, to improve their resource usage, and the performance of the resulting overlay network.

As mentioned in Section 1, the overlay node activation problem has received little attention so far. As far as we are aware, the problem has only been studied in the context of intrusion-tolerant networks [1]. The authors propose an algorithm that selects overlay nodes from a set of inactive ones to replace unresponsive or compromised nodes. They, however, focus on security and Byzantine fault tolerance. In contrast, our main goal is to maximize the satisfied client demand, while minimizing the amount of idle resources.

3 Problem Formulation

Given is an underlying network topology represented by a bidirectional graph $U(N, E)$, with N a set of network nodes (e.g. clients, servers or routers) and E a set of edges connecting these nodes. Each edge e has a distance d_e (e.g. one-way transmission delay) and available bandwidth b_e.

Also given is a set of candidate overlay nodes $O \subseteq N$, a set of clients $C \subseteq N$ and a set of service-hosts $S \subseteq N$. Any two overlay nodes, clients or service-hosts m and n are connected through a path of edges $p_{m,n} \subseteq E$ (called an overlay edge) in the underlaying network (in our implementation the shortest hop-count path). The distance $d_{m,n}$ and bandwidth $b_{m,n}$ between two overlay nodes m and n are respectively defined as $\sum_{e \in p_{m,n}} d_e$ and $\min_{e \in p_{m,n}} b_e$.

Every overlay node $o \in O$ has a limited amount of resources Ω_o and a set of nearby overlay nodes $N_o \subset O$ (the neighbours of o). The neighbourhood set N_o of $o \in O$ contains all overlay nodes $v \in O$ for which $d_{o,v} \leq D$ (with D a predefined distance bound).

Every client and service-host $u \in C \cup S$ has an amount of required resources ω_u (e.g. CPU, memory), which represent the resources needed by the gateway (the overlay node used to access the overlay network) to perform overlay-related tasks. For clients this would consist of service discovery, server selection and request routing. For service-hosts this would be service placement, service advertisement and reply routing.

The set M_o denotes all $u \in C \cup S$ that have $o \in O$ as nearest overlay node. This set contains all clients and service-hosts for which $d_{u,o} = \min_{q \in O} d_{u,q}$.

Additionally we define a set of available services T. Each client $c \in C$ has requests for a subset of these services $T_c \subseteq T$. The variable $r_{c,t}$ denotes the amount of requests per second of client $c \in C$ for service $t \in T_c$. Each service-host $s \in S$ runs a subset of all available services. Each service $t \in T$ has a required amount of bandwidth β_t^o per request and β_t^i per reply. The service-host that satisfies the demand of client $c \in C$ for service $t \in T_c$ is denoted by $s_{c,t}$.

The goal of overlay node activation is to select a subset $A \subseteq O$ of active overlay nodes. Additionally a gateway $o \in A$ should be selected for each client and service-host $u \in C \cup S$. Several constraints should be satisfied during the activation process. These can be summarized as follows

$$\forall o \in A : \sum_{u \in U_o} \omega_u \leq \Omega_o \ . \tag{1}$$

$$\forall o \in A, \forall u \in U_o : d_{u,o} \leq K \ . \tag{2}$$

$$\forall e \in E : \sum_{c \in C} \sum_{t \in T_c} \left(\sum_{p \in \pi^o_{c,t} \wedge e \in p} r_{c,t} \cdot \beta_t^o + \sum_{p \in \pi^i_{c,t} \wedge e \in p} r_{c,t} \times \beta_t^i \right) \leq b_e \ . \tag{3}$$

With U_o the set of clients and service-hosts using $o \in A$ as gateway, K the predefined maximum allowed distance between a client or service-host and their gateway, and $\pi^o_{c,t}$ and $\pi^i_{c,t}$ respectively the set of overlay edges in the overlay path from the client c to the service-host $s_{c,t}$ and back. Eq. 1 dictates that the

resources used by all clients and service-hosts using an overlay node as a gateway should not exceed the available resources of that overlay node. Eq. 2 states that the distance between a client or service-host and its gateway should not exceed a predefined distance bound K. And finally Eq. 3 stipulates that the bandwidth used on each underlay edge to send requests and replies for all clients should not exceed the available bandwidth of that edge.

The optimization goal consists of three subgoals, which can be described as follows

1. Satisfied client demand should be maximized. This means that enough bandwidth should be available in the overlay network to send as many service requests and replies between clients and service-hosts as possible.
2. A gateway should be provided for as many clients and service-hosts as possible.
3. Enough overlay nodes should be activated in order to maximize the first two goals, but not more. This goal minimizes the amount of idle overlay nodes.

4 Algorithms Description

4.1 Integer Linear Programming Approach (ILP)

Based on the formal problem description given in Section 3 we can define an ILP formulation for the problem. This formulation can be used to find the optimal solution for the overlay node activation problem. Although this approach scales very poorly, it is still useful for comparing the solution of the heuristics to the optimal.

We will start by defining several decision variables used by the constraints and objective function

- a_o is 1 if $o \in A$ (o is activated) and 0 otherwise
- $g_{u,o}$ is 1 if $u \in U_o$ ($o \in A$ is the gateway of $u \in C \cup S$) and 0 otherwise
- $q^{o/i}_{c,t,m,n}$ is 1 if $p_{m,n} \in \pi^{o/i}_{c,t}$ (with $m, n \in O \cup C \cup S$) and 0 otherwise
- $r^{o/i}_{c,t,m,n}$ is the amount of satisfied requests of $c \in C$ for $t \in T_c$ on $p_{m,n} \in \pi^{o/i}_{c,t}$

The ILP formulation consists of several constraints, which limit the possible values of the decision variables to valid solutions only. The first three constraints are the same as the ones given in Section 3 (though formulated using the decision variables). Additionally, there are several other constraints. These include constraints that make sure every client has only 1 gateway and that all gateways and overlay nodes used for routing client requests or replies are activated. Finally, there are also several flow conservation constraints.

The optimization goal of maximizing the total satisfied demand and number of clients and service-hosts with a gateway while minimizing the total number of activated overlay nodes, can be translated into the following ILP objective function

$$\max \sum_{o \in O} \left(\sum_{u \in C \cup S} 2 \cdot \omega_u \cdot g_{u,o} \right) + \left(\sum_{c \in C} \sum_{t \in T_c} \left(\beta^o_t + \beta^i_t \right) \cdot r^o_{c,t,c,o} \right) - a_o \ .$$

Algorithm 1. The static node activation heuristic (STANA)

procedure SelectGateways()

1: **for all** $o \in O$ **do**
2: **for all** $c \in C_o$ **do**
3: $n \leftarrow$ o.GetMostSatisfyingGateway(c); $C_o \leftarrow C_o \setminus \{c\}$; $C_n \leftarrow C_n \cup \{c\}$
4: **for all** $o \in O$ **do**
5: **for all** $s \in S_o$ **do**
6: $n \leftarrow$ o.GetMostSatisfyingGateway(s); $S_o \leftarrow S_o \setminus \{s\}$; $S_n \leftarrow S_n \cup \{s\}$

$b \in O$ **function** o.GetMostSatisfyingGateway(u)

1: $b \leftarrow o$; $r \leftarrow$ o.GetSatisfiableDemand(u)
2: **for all** $n \in O$ **do**
3: **if** $d_{u,n} \leq K$ **and** $\omega_u + \sum_{v \in C_n \cup S_n} \omega_v \leq \Omega_n$ **then**
4: **if** $\sum_{v \in C_o \cup S_o} \omega_v < \sum_{v \in C_n \cup S_n} \omega_v$ **or** $\sum_{v \in C_o \cup S_o} \omega_v > \Omega_o$ **then**
5: **if** n.GetSatisfiableDemand(u) $> r$ **then**
6: $b \leftarrow n$; $r \leftarrow$ n.GetSatisfiableDemand(u)
7: **return** b

An algorithm to solve the ILP formulation was implemented using ILOG CPLEX [8], which solves ILP problems using the simplex and interior point methods.

4.2 Static Node Activation (STANA)

The pseudocode of the heuristic is shown in Algorithm 1. The heuristic is initiated by calling the `SelectGateways()` procedure. First it attempts to find the best gateway for each client (lines 1-3). Then it attempts to find the best gateway for the service-hosts (lines 4-6). C_o and S_o represent the clients and service-hosts that have o as their gateway. They are initialized to the clients and service-hosts in M_o.

The `GetMostSatisfyingGateway()` function returns an overlay node b for the given client or service-host u. The number of requests of u that can be satisfied is maximal if b is the gateway of u. The function iterates over all overlay nodes (line 2). First it checks if the overlay node n meets the requirements to be the gateway of u (line 3). The distance from u to n should be less than K and n should have enough free resources (e.g. CPU). Subsequently the function checks if n has a higher load than the current gateway o of u or if o is overloaded (line 4). The reason an overlay node will only transfer clients and service-hosts to neighbours with higher load (unless it is overloaded) is based on the observation that it would be easier to transfer all clients and service-hosts away from a node with only few of them, than from a node with many. This allows the heuristic to empty as many nodes with low load as possible and minimize the number of overlay nodes that should be activated. Finally the function checks if n would be a better gateway than the current best gateway b and replaces it if necessary (lines 5-6).

After calling the procedure all overlay nodes o for which $C_o \cup S_o \neq \emptyset$ will be activated. All overlay nodes on the path from a client c to one of its service-hosts $s_{c,t}$ or back will also be activated.

4.3 Dynamic Node Activation (DYNNA)

Although the static heuristic is interesting from a theoretical point of view and for initial network dimensioning, it cannot adapt to changing conditions (such as changes in client demand, underlying network conditions, or node churn). To be able to quickly adapt to these changes we have devised a dynamic heuristic. Additionally, this heuristic runs fully distributed and overlay nodes use only measurable information from their direct neighbours (current resource load, overlay edge transmission delay and overlay edge bandwidth usage). In contrast, the static heuristic requires a view on the entire overlay network and needs information on expected future bandwidth usage of each client and which service-hosts each client will use.

Pseudocode for the dynamic heuristic is shown in Algorithm 2. Several new variables are introduced: specifically Ω_{min}, Ω_{max}, β_{min}, β_{max}, $b^u_{m,n}$, b^{out}_u and b^{in}_u. Ω_{min} and Ω_{max} define the minimum and maximum resource load percentage of an overlay nodes before it should attempt to transfer clients and service-hosts to other overlay nodes. β_{min} and β_{max} have simular meanings, but for the overlay edge bandwidth. $b^u_{m,n}$ denotes the used bandwidth on the overlay edge between any two overlay nodes, clients, or service-hosts m and n. Finally b^{out}_u and b^{in}_u denote the bandwidth currently provided to client u, respectively to and from its gateway.

A node initiates the algorithm by calling the `ReDistribute()` procedure. This could be done at fixed intervals or when the node detects a significant change in resource load or bandwidth usage of its overlay edges. The heuristic consists of 5 phases. The first phase (lines 2-8) is only initiated if the resource load percentage of the overlay node is greater than Ω_{max}, meaning it is either overloaded or nearly overloaded. It will first attempt to transfer clients and service-hosts to its active neighbours (lines 3-4). If this fails it will activate additional neighbours and try to transfer clients and service-hosts to them (lines 5-8). The second phase (lines 9-11) is only initiated if the overlay node has a resource load of Ω_{min} or less and the bandwidth usage of its outgoing links is less than β_{min}. This means the node has very low load, and only few service requests or replies pass by it. The node will attempt to transfer all its clients and service-hosts to active neighbours, so it can deactivate itself. In the third phase (lines 12-16) the node checks the available bandwidth on the links to and from its connected clients and service-hosts (lines 12-14). All clients and service-hosts that might require more bandwidth are, when possible, transferred to other nodes with more available bandwidth (lines 15-16). In the fourth phase (lines 17-18) the node checks if the average bandwidth usage on its outgoing overlay edges exceeds β_{max}. If it does, the node attempts to increase the available bandwidth in the overlay by activating extra neighbours. In the fifth and final phase (lines 19-20) the overlay

Algorithm 2. The dynamic node activation heuristic (DYNNA)

procedure o.ReDistribute()

1: $T \leftarrow \{\}$; $A \leftarrow$ GetActive(N_o) ; $I \leftarrow N_o \setminus A$; $U_o \leftarrow C_o \cup S_o$
2: **if** $\sum_{u \in U_o} \omega_u > \Omega_{max} \cdot \Omega_o$ **then**
3: **while** HasNext(A) **and** $\sum_{u \in U_o} \omega_u > \Omega_{max} \cdot \Omega_o$ **do**
4: $s \leftarrow$ GetNext(A) ; $U_o \leftarrow U_o \setminus$ s.TransferNodes(U_o, $\{o\}$)
5: **while** HasNext(I) **and** $\sum_{u \in U_o} \omega_u > \Omega_{max} \cdot \Omega_o$ **do**
6: $s \leftarrow$ GetNext(I) ; s.Activate() ; $A \leftarrow A \cup \{s\}$; $I \leftarrow I \setminus \{s\}$;
7: $U_o \leftarrow U_o \setminus$ s.TransferNodes(U_o, $\{o\}$)
8: $U_o \leftarrow U_o \setminus$ o.GetOverloadingNodes(U_o)
9: **if** $\sum_{u \in U_o} \omega_u < \Omega_{min} \cdot \Omega_o$ **and** $\text{avg}_{s \in A} \frac{b^u_{s,o}}{b_{s,o}} < \beta_{min}$ **then**
10: **while** HasNext(A) **and** $|U_o| > 0$ **do**
11: $s \leftarrow$ GetNext(A) ; $U_o \leftarrow U_o \setminus$ s.TransferNodes(U_o, $\{o\}$)
12: **for all** $u \in C_o \cup S_o$ **do**
13: **if** $b^u_{u,o} > \beta_{max} \cdot b_{u,o}$ **or** $b^u_{o,u} > \beta_{max} \cdot b_{o,u}$ **then**
14: $T \leftarrow T \cup \{u\}$
15: **for all** $s \in A$ **do**
16: $U_o \leftarrow U_o \setminus$ s.TransferNodes($T \cap U_o$, $\{o\}$)
17: **while** HasNext(I) **and** $\text{avg}_{s \in A} \frac{b^u_{o,s}}{b_{o,s}} > \beta_{max}$ **do**
18: $s \leftarrow$ GetNext(I) ; s.Activate() ; $A \leftarrow A \cup \{s\}$; $I \leftarrow I \setminus \{s\}$
19: **if** $|U_o| = 0$ **and** $\text{avg}_{s \in A} \frac{b^u_{s,o}}{b_{s,o}} < \beta_{min}$ **then**
20: o.DeActivate()

$T \subseteq U$ **function** o.TransferNodes(U, V)

1: $T \leftarrow \{\}$; $A \leftarrow$ GetActive(N_o) ; $W \leftarrow$ o.GetNodesInReach(U) ; $V \leftarrow V \cup \{o\}$
2: **if** $|W| > 0$ **then**
3: **for all** $u \in W$ **do**
4: **if** $\omega_u + \sum_{v \in U_o} \omega_v < \Omega_{max} \cdot \Omega_o$ **then**
5: **if** $b^{out}_u \leq \beta_{max} \cdot b_{u,o}$ **and** $b^{in}_u \leq \beta_{max} \cdot b_{o,u}$ **then**
6: $U_o \leftarrow U_o \cup \{u\}$; $T \leftarrow T \cup \{u\}$
7: **for all** $s \in A \setminus V$ **do**
8: $T \leftarrow T \cup$ s.TransferNodes($W \setminus T$, V)
9: **return** T

node deactivates itself if no clients or service-hosts use it as gateway and the incoming bandwidth on its overlay edges is below β_{min}.

The `TransferNodes()` function recursively attempts to find a new gateway for the clients and service-hosts in U. It returns a set T of all elements of U for which a new gateway was found. A node first checks if it can act as a gateway for any of the nodes in U (lines 3-6). Then it sends the remaining nodes to its active neighbours (lines 7-8) who recursively do the same. The extra V parameter is used to prevent loops. Nodes already visited in a recursive call will be stored in V and are not visited again. `GetNodesInReach()` merely returns all clients and service-hosts $u \in U$ for which $d_{u,o} \leq K$.

Note that the algorithm disconnects clients and service-hosts if a node is overloaded and no replacement gateway is found (line 8 of `ReDistribute()`).

Additionally, the algorithm does not check if the distance to the current gateway has become larger than K (it will however check this when looking for a new gateway). If a client or service-host has been disconnected or the distance to its gateway has changed it is itself responsible to find a new gateway (by reconnecting to another overlay node).

The join procedure for clients and service-hosts is described as follows. First it contacts a random overlay node (most likely selected from a set of static bootstrap servers) and requests a set of nearby overlay nodes. It then contacts the nearest from this set of nearby overlay nodes. If that node is active and it has enough free resources it will become the new gateway of the client (or service-host). If it is inactive or it does not have enough free resources it will attempt to find another gateway using the `TransferNodes()` function. If this is unsuccesful the node will activate itself if it was inactive or disconnect the client if it was already active.

5 Evaluation Results

In this section the performance, scalability and adaptability of the heuristics is discussed based on simulation results. Performance is measured by comparing the percentage of satisfied demand and activated overlay nodes to the optimal solution (calculated using the ILP algorithm) for growing bandwidth load. Scalability is evaluated by comparing results for a growing amount of available overlay nodes. Finally, adaptability is studied by dynamically changing the demand pattern. Satisfied demand is measured as the percentage of requests from clients that can be processed by service-hosts.

For reference purposes, we also implemented an underlay algorithm (UND). This algorithm routes data from the sender to the receiver directly via the shortest-hop-count path in the underlay network.

Statistical analysis was used to interpret simulation results. A one-way ANOVA [9] was used to compare several levels of a single factor. If an effect of the factor was detected, a Tukey test [9] was performed to determine which averages actually differed significantly. The 'homogeneity of variance' prerequisite for ANOVA was checked using a modified Levene test [9]. All statistics were performed using a 5% signficance level.

5.1 Simulation Setup

Random underlay networks were generated using the BRITE topology generator [10] with the hierarchical top-down model and Waxman's algorithm. The bandwidth of the links between underlay routers was chosen according to a uniform distribution in the interval $\{5, 15\}$. Each router had on average an outdegree of 2. All routers were placed on a 200×200 grid and the link delay was chosen directly proportional to the euclidean distance in this grid. A subset of all available routers was selected as access routers, all others are considered core routers. Clients and service-hosts were connected to a randomly selected access router by

a fast and high-bandwidth link (0 ms delay and 100 Mbps bandwidth). Overlay nodes were directly connected to the access routers and high-degree core routers, also by fast, high-bandwidth links. The CPU resources of each overlay node were chosen uniformly from the interval $\{6, 8\}$. As the resource requirements of each client and service-host were set to 1, this means every overlay node could be the gateway of at most 6 to 8 devices.

5.2 Performance Results

The goal of this simulation was to evaluate the performance of the heuristics in terms of satisfied demand and amount of activated overlay nodes for different bandwidth loads (Mbps). The bandwidth load is measured as the total bandwidth needed by all client requests.

Because of the exponential time-complexity of the ILP algorithm, this simulation was performed only on small networks. Each network contained 20 routers and 12 overlay nodes.

Fig. 1 shows the simulation results as a function of bandwidth load (Mbps) and for different values of K (maximum distance to the gateway). As expected, the satisfied demand is inversely proportional and the number of activated overlay nodes is directly proportional to the bandwidth load.

Statisticaly analysis showed that, in terms of satisfied demand, ILP did not perform significantly better than DYNNA and only significantly better than STANA for $K = 150$ at 20 Mbps and higher. For $K = 50$, ILP performed at most respectively 7 and 9% better than DYNNA and STANA. On the other hand, both heuristics performed up to 25% better than UND.

In terms of number of activated overlay nodes, ILP performed significantly better than both heuristics (respectively at most about 15% for $K = 50$ and 25% for $K = 150$). STANA performed significantly better than DYNNA only for $K = 150$, from 10 to 20 Mbps.

In summary, we can remark that the heuristics perform significantly better than classic underlay-based routing. There is however little difference between both heuristics. Additionally, for the heuristics there is little difference between results for different values of K.

5.3 Scalability Results

In this section, we study the scalability of the heuristics in terms of satisfied demand and amount of activated overlay nodes, for a growing number of candidate overlay nodes. As the ILP algorithm was not used for this simulation, larger test networks were possible. Tests were performed for networks with 2500 and 10000 underlay routers. The number of candidate overlay nodes was varied between 0 and 250.

Fig. 2 shows the simulation results as a function of available overlay nodes. As more overlay nodes become available, the amount of satisfied demand and number of used overlay nodes grows proportionally. It is striking that only a portion of available overlay nodes is used, even if only few are available. This is

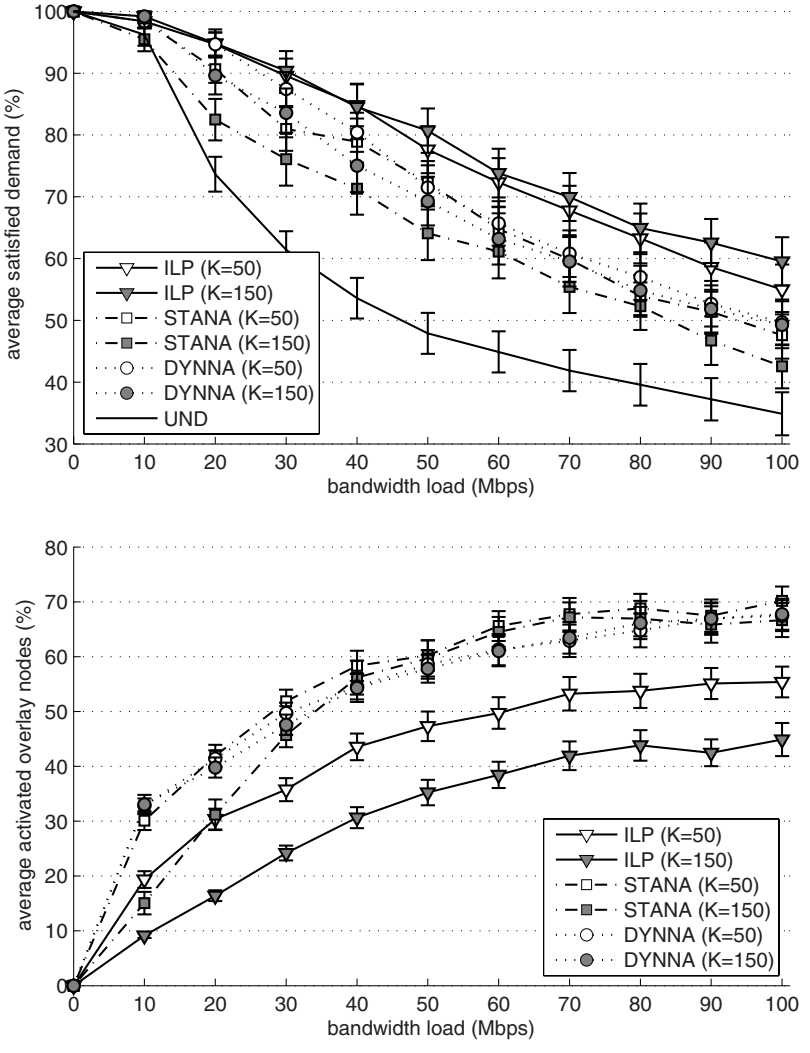

Fig. 1. The performance of the two heuristics (STANA, DYNNA) compared to the optimal algorithm (ILP) and underlay routing (UND) in terms of satisfied demand (%) (left) and activated overlay nodes (%) (right) for varying bandwidth load (Mbps) and maximum gateway distance K (averaged over 30 iterations - the error bars represent the standard error)

because the underlay paths corresponding to the overlay links of different overlay nodes might overlap. One of those nodes might cause the bandwidth in these links to become saturated. The algorithms will detect that there is no bandwidth available in the incoming or outgoing links of the other overlay node and will thus not use it.

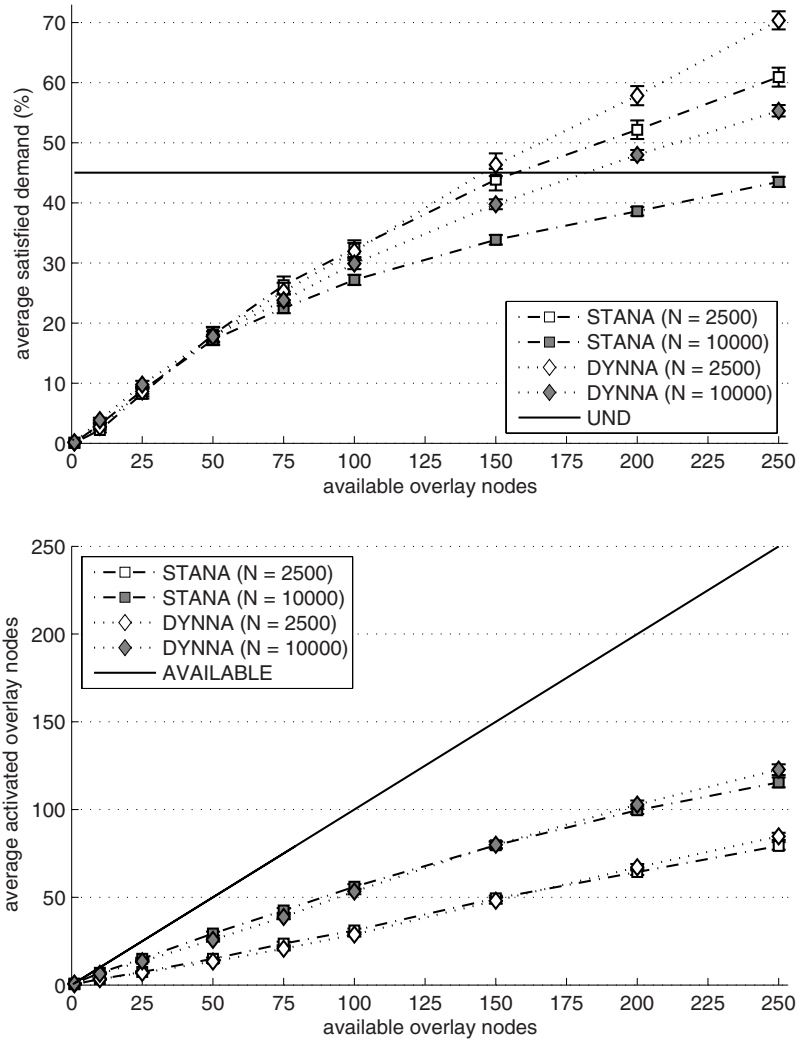

Fig. 2. The scalability of the two heuristics (STANA, DYNNA) compared to underlay routing (UND) in terms of satisfied demand (%) (left) and activated overlay nodes (%) (right) for varying number of available overlay nodes and for networks with 2500 and 10000 underlay routers (averaged over 39 iterations - the error bars represent the standard error)

Statistical analysis shows that DYNNA performs significantly better than STANA in terms of satisfied demand from 100 available overlay nodes and onwards when using 10000 routers, but only from 200 onwards when using 2500 routers. Additionally, no significant differences were found between both heuristics in terms of number of activated overlay nodes (for 2500 and 10000 routers).

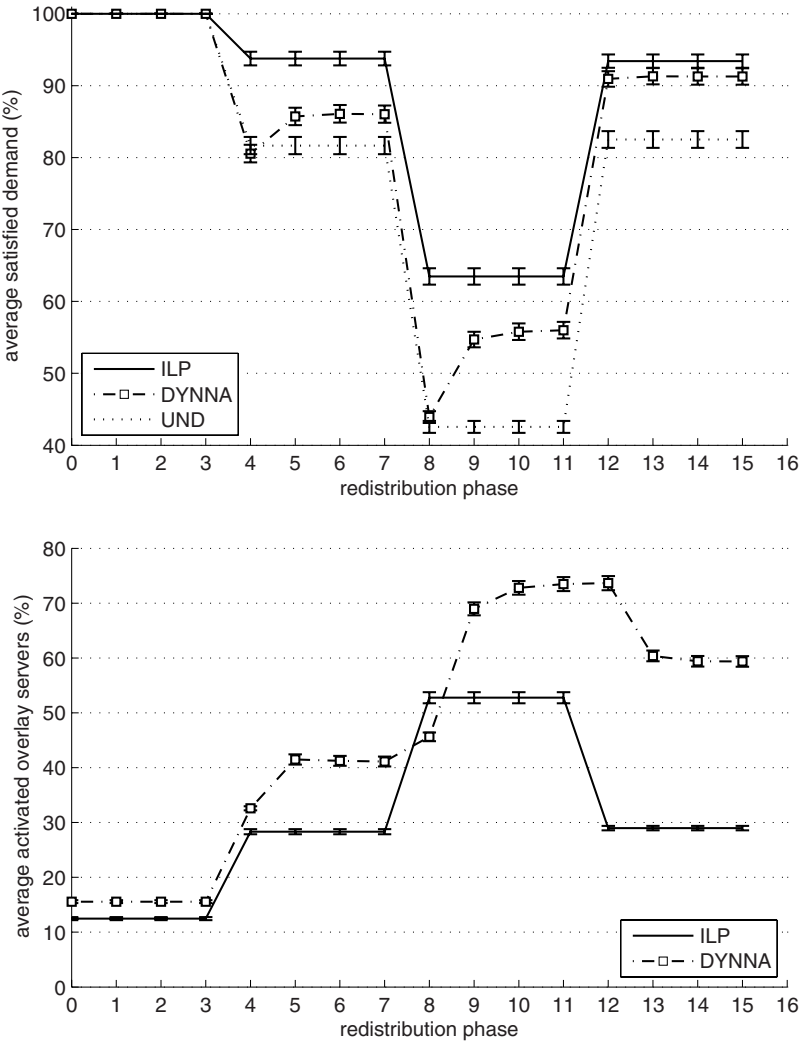

Fig. 3. The adaptability of DYNNA to dynamically changing demand compared to ILP and UND, in terms of satisfied demand (%) (left) and activated overlay nodes (%) (right) (averaged over 174 iterations - the error bars represent the standard error)

In summary, we have shown that, in line with the first experiment, there is little difference between both heuristics in terms of the amount of overlay nodes they activate. Additionally these simulations show that even when placing overlay nodes near hotspots (routers with high link degree) not all overlay nodes can be efficiently used, because of bandwidth constraints. Therefore, the presented node activation heuristics could prove to be very useful for deriving suitable locations for overlay infrastructure nodes.

5.4 Dynamic Demand Results

In this section, DYNNA is evaluated in face of dynamically changing demand patterns. Tests were performed on networks with 40 routers and 20 overlay nodes. In our example scenario, the demand was initialized to 0 Mbps and subsequently changed to 50, 200 and back to 50 Mbps. After each change in demand the dynamic algorithm's `ReDistribute()` was called 4 times on each node (called redistribution phases), to make sure the solution stabilized before the demand changed again. Consequently, demand increases occurred after phase 3 and 7 and a decrease after phase 11.

Fig. 3 shows the simulation results after subsequent redistribution phases. The figure shows that in terms of satisfied demand, one redistribution phase is needed for the results to stabilize after an increase in demand, and none after a decrease. On the other hand, the number of activated overlay nodes needs one redistribution phase to stabilize, both after an increase and decrease in demand. Additionally, it should be noted that after a decrease in demand (phase 11) the number of activated overlay nodes remains higher than it was before the identical increase (phase 7). As an interesting side effect, satisfied demand remains closer to optimal as well.

6 Summary

In this paper, we presented an optimal algorithm and two heuristics to solve the node activation problem in the context of service hosting platforms. The optimal algorithm (ILP) and the first heuristic (STANA) solve only a static version of the problem. This means that the request pattern for all clients and the entire network topology is known (or estimated) in advance. The second heuristic (DYNNA) is capable of dynamically (de)activating overlay nodes to cope with changes in demand patterns and client or service-host churn. To improve scalability it can also be run in a distributed way, as each overlay node only requires knowledge on its direct overlay neighbours and the links between them.

Using extensive simulations, we showed that the dynamic (distributed) heuristic performs as good, if not better, than the static heuristic. Additionally, we showed that when bandwidth is plentiful, the heuristics do not perform significantly worse than the optimal algorithm.

Overlay node activation mechanisms provide several advantages to any overlay-based platform. First, dynamic node activation allows idle resources to be temporarily freed and used for other purposes. Intelligent algorithms can reclaim or free nodes to cope with fluctuations in overlay-resource demand. Second, idle nodes no longer take part in overlay control and management protocols, improving overall performance and scalability. And finally, static node activation can be used to find suitable locations for infrastructure nodes.

References

1. Obelheiro, R.R., Fraga, J.S.: Overlay network topology reconfiguration in byzantine settings. In: IEEE Pacific Rim Dependable Computing Conference (PRDC 2007), pp. 155–162 (2007)
2. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: Proceedings of the 15th International Conference on World Wide Web (WWW 2006), pp. 595–604 (2006)
3. Adam, C., Stadler, R., Tang, C., Steinder, M., Spreitzer, M.: A service middleware that scales in system size and applications. In: 10th IFIP/IEEE International Symposium on Integrated Management (IM 2007), pp. 70–79 (2007)
4. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001), pp. 149–160 (2001)
5. Tang, C., Chang, R.N., Ward, C.: Gocast: Gossip-enhanced overlay multicast for fast and dependable group communication. In: Conference on Dependable Systems and Networks (DSN 2005), pp. 140–149 (2005)
6. Voulgaris, S., Gavidia, D., van Steen, M.: CYCLON: Inexpensive membership management for unstructured P2P overlays. Journal of Network and Systems Management 13(2), 197–217 (2005)
7. Ganesh, A.J., Kermarrec, A.M., Massoulie, L.: Peer-to-peer membership management for gossip-based protocols. Transactions on Computers 52(2), 139–149 (2003)
8. ILOG: CPLEX 10.0 User's Manual. ILOG Inc., Mountain View, CA (2006)
9. Hill, T., Lewicki, P.: Statistics: Methods and Applications. StatSoft, Inc. (2006)
10. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: An approach to universal topology generation. In: Proceedings of the Internation Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2001) (2001)