2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

QuLa: service selection and forwarding table population in Service-Centric Networking using real-life topologies

Piet Smet, Bart Dhoedt and Pieter Simoens
Department of Information Technology (INTEC)
Ghent University - iMinds
Ghent, Belgium

***Abstract*—The amount of services located in the network has drastically increased over the last decade which is why more and more datacenters are located at the network edge, closer to the users. In the current Internet it is up to the client to select a destination using a resolution service (Domain Name System, Content Delivery Networks ...). In the last few years, research on Information-Centric Networking (ICN) suggests to put this selection responsibility at the network components; routers find the closest copy of a content object using the content name as input.**

We extend the principle of ICN to services; service routers forward requests to service instances located in datacenters spread across the network edge. To solve this problem, we first present a service selection algorithm based on both server and network metrics. Next, we describe a method to reduce the state required in service routers while minimizing the performance loss caused by this data reduction. Simulation results based on real-life networks show that we are able to find a near-optimal load distribution with only minimal state required in the service routers.

***Keywords*-Service-Centric Networking, Information-Centric Networking, Latency-aware selection, name-based routing, QuLa**

I. Introduction

Over the last decade Internet usage has developed from data delivery between end-hosts to data distribution from one source to multiple destinations (e.g. many users watching the same video). To meet the requirements of the increasing user demand, Content Delivery Networks (CDNs) were developed to reduce network latency, bandwidth and congestion by caching content in the network edge and load-balancing requests over multiple replicas. While CDNs were originally developed for static content retrieval, certain implementations such as Akamai [1] also put a great deal of focus into support for data-processing applications. However, the long-term sustainability of CDNs is endangered by technology heterogeneity, poor reactivity, inefficient resource utilization and coarse granularity in management operations, as shown in recent research [2].

Another approach to optimize content delivery, Information-Centric Networking (ICN) [3] [4] [5] [6] [7] [8], allows users to send out an anycast-like message (i.e. one identifier can address multiple replicas) to search for data, using object names instead of IP addresses to identify the desired data. The responsibility to find the nearest content replica and load-balance requests resides with the network components. The concept of ICN led to several forwarding and caching optimizations to improve content delivery [9] [10] [11].

Both CDN and ICN are developed to support static data retrieval and do not consider complications introduced by services: services are prone to dynamic service times, service requests often need to consider input data and user-specific requests can't be cached. Currently, services often reside in datacenters or cloud sites, although the induced network latency and large bandwidth required between users and the cloud is often not desired for data-processing applications. Similar to how CDNs and ICN bring content closer to the user, we can also move services to clouds located in the network edge [12] [13] [14]. However, current solutions to facilitate the distribution of real-time data-processing services are limited to specific cloud infrastructures; an ICN-like solution could overcome this problem.

This led to the development of Service-Centric Networking (SCN) [15] which expands the principles of ICN to allow efficient provisioning, discovery and execution of services in the network. Requests are processed by service routers and forwarded to the instance with the lowest response time. Similar to ICN, SCN also uses object names to identify the desired services.

In this paper we present a static service selection algorithm which minimizes the average system response time in Service-Centric Networking. Our proposed algorithm, Queue and Latency (QuLa), considers both network latency and server processing times while calculating the optimal load distribution. Our selection algorithm runs on a centralized broker which contains knowledge of both server and network metrics. The outcome of this algorithm is a load distribution matrix which maps the demand generated by the users to the service instance replicas deployed in the network. Using this load distribution matrix, a service router is able to forward requests to the nearest replica based on the client source location and the requested service name (*source-based routing*). As we envision a large amount of services and users in a SCN network, the state maintained in service routers poses a scalability problem. To overcome this problem, we present a *weighted average* method to reduce the amount of state required in the service

978-1-4673-8776-7/16 $31.00 © 2016 IEEE
DOI 10.1109/PDP.2016.11

router forwarding tables while minimizing the performance loss caused by this data reduction. This concept has already been proposed in a prior publication [16] using artificial networks for preliminary results. To verify the benefit of our selection algorithm and *weighted average* forwarding approach in more practical environments, we performed simulations on real-life network topologies which are presented in this paper.

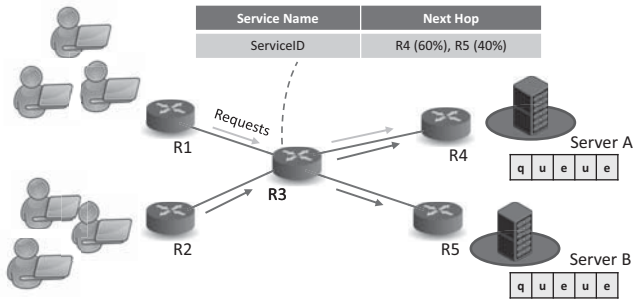


Fig. 1. Users send demand on the network which is forwarded to a service instance by service routers using a name-based forwarding scheme.

The remainder of this paper is structured as follows. Section II provides an overview of related work on ICN, SCN and name-based service selection and routing. The QuLa service selection algorithm is presented in section III. Section IV explains the weighted average method to eliminate the client source from the forwarding process. Our simulation results show that QuLa is able to approximate benchmark results with minimized router state, as presented in section V. Finally, in section VI we present our conclusion and discuss future work.

II. Related work

In this section we briefly discuss related work to the different aspects of QuLa. Although there is much related work on architectural designs for SCN (e.g. Serval [17] and SCAFFOLD [18]), we will focus on service selection and name-based routing in this paper.

Service selection. Service selection is responsible for finding the best instance for each request based on a certain objective. In case of the Zoom-in-zoom-out algorithm [19] the objective is to select a minimum set of servers while keeping the experienced delays below a certain threshold. A more generic approach such as DONAR [20] uses a generic cost function during the selection process which needs to be minimized. While Zoom-in-zoom-out only considers network delays, DONAR also takes server processing times into consideration. QuLa considers both network and server metrics while minimizing the average system response time as experienced by the users. Unlike DONAR, which suggests a decentralized mapping scheme, QuLa runs as a one-time centralized selection algorithm for a certain service placement and request pattern.

Name-based Routing. As described in the principles of SCN, the network components are responsible for forwarding a request to a service instance without using a destination locator such as an IP address. This means that the final destination is unknown during the forwarding process, similar to native IP anycast where one address can correspond with several destinations. IP anycast is limited to the use of only network metrics when forwarding a request. Related work such as a load-aware anycast CDN [21] attempted to overcome these limitations by introducing a centralized CDN controller which considers both network and server metrics in the redirection mechanism. However, this solution was focused on CDNs and does not overcome the limitations of CDN itself.

Research shows that the use of an additional selection component such as an Akamai-server is more beneficial than IP routing in more than 50% of the scenarios [22]. Therefore, our work uses a similar forwarding scheme to the ICN framework Content Centric Network / Named Data Networking (CCN/NDN) [4]; each router forwards a request to the next router based on the requested service name. This allows the network to quickly set up alternative paths when network hotspots are detected, even if multple services are located on the same destination.

There are multiple ways to populate the forwarding tables. *A first approach* is to modify existing protocols to support name-based requests (e.g. OSPF-N [23]). A second approach is to populate the forwarding tables at runtime by learning which interfaces receive the fastest response (e.g. Greedy Ant Colony Forwarding (GACF) [24] and CCN/NDN [4]). Another popular and *third method* is to combine the previous two approaches by modifying existing ICN approaches, focused on static content retrieval, to support service requests. SoCCeR [25] is a decentralized routing protocol which adopts this method by building on top of CCNx (an implementation of CCN) and uses Ant Colony Optimization to gather both network and server metrics to configure the Forwarding Information Base of CCNx nodes. While SoCCeR is a plausible way to populate the forwarding tables at runtime by learning the best interfaces to forward on, QuLa aims to find the best load distribution and maps this to the forwarding tables in one iteration, minimizing the activity of the routing plane at runtime.

An important implication of expanding ICN/CCN principles to services is that processing costs of services are generally much higher than retrieving a static content object. In contrary to static data retrieval techniques such as ICN/CCN where each request can be answered by multiple destinations and only the fastest is accepted, each service request in SCN should be processed by only one service instance to avoid unnecessary use of computing resources. Therefore, the QuLa selection mapping and the forwarding components presented in this paper will assign each request to exactly one service instance.

III. QuLa: finding the best load distribution

The main goal of our selection algorithm is to minimize the average system response time, which depends on both the time spent on server and the time a request travels in the network. Using standarized routing protocols would be insufficient to get a realistic prediction of the response time

as they usually only consider network metrics (e.g. hop-count, bandwidth, network latency ...) which are often used in shortest path algorithms such as Dijkstra's algorithm [26]. To find the response time resulting from a certain load distribution we use the queue size to calculate the average time a request will spend on a server and we use network latency for the Round Trip Time (RTT) of each request. This is why our algorithm is called Queue and Latency (QuLa).

Our implementation of QuLa is based on Simulated Annealing [27] and as a benchmark we adopted (1) a greedy shortest path approach, (2) assigning equal load to each server and (3) a dynamic assignment of each request to the shortest queue upon request arrival. In this section we present the problem statement solved by the QuLa selection algorithm and provide a more detailed description of the implemented algorithms.

A. Assumptions

Mapping user demand to a service instance for each user on the network individually is not feasible when load-balancing must be done in real-time. To overcome this scalability problem, we aggregate the demand of all users in a nearby geographical area to one demand node, which we will refer to as client node i. Using this approach our selection becomes more coarse-grained but is more scalable as it induces less state to be kept in the forwarding tables. We use server node j to refer to a collection of computing resources located in a nearby geographical area (e.g. datacenter or cloud site).

Next, we assume that each server node has a queue for incoming requests and produces stable, reproducible processing times. In our experiments the demand coming from users arrives with an average rate ressembling a Poisson process. We model the server queuing times using an M/G/1 queue, although the principles of this paper apply to any other queue as well.

Last, we assume that the given service placement is fixed for the duration of the experiment; no service instances are deployed, migrated or stopped during our experiments to focus on the selection quality. The user demand pattern also does not change over time; each user may have a different request rate but the average of each user is fixed throughout our experiment. If any of these fixed values were to change over time, a new selection should be made to maintain an optimal load distribution.

B. Problem statement

To solve the selection problem we consider a network graph containing edges E, nodes N and services S. The request rate from node i for service s is denoted as the lambda value $\lambda(i, s)$. All client nodes belong to the collection $N_c \subset N$. Server nodes $N_S \subset N$ are nodes which host at least one instance of a service $s \in S$. The outcome of the service selection is a load distribution matrix $R(i,j,s) \in [0, 1]$ which maps a fraction of load from client node i for service s to a server node j which hosts at least one instance of s. The time to process a request for service s on server node j after it leaves the server queue is called the average service time, denoted as $\overline{T_{j,s}}$.

The cost to minimize is the average response time, which is largely affected by the time spent in the network and the time spent on the server. The more demand a server must process, the longer new requests must spend in queue before they get processed. This leads to the following objective function:

$$\min \frac{\sum_{i \in N_c} \sum_{j \in N_S} \sum_{s \in S} \left(T_{Lat.} + T_{proc.} \right) * R(i,j,s) * \lambda(i,s)}{\sum_{i \in N_c} \sum_{s \in S} \lambda(i,s)} \quad (1)$$

Taking the sum of $T_{Lat.}$ and $T_{proc.}$ gives us the response time of a single request. $T_{Lat.}$ is the network Round Trip Time (RTT) between client node i and server node j. The total time spent on a server, including queue delay and the service processing time, is denoted as $T_{proc.} = f(R(i, j, s))$. Since we're only sending a fraction $R(i, j, s)$ of demand $\lambda(i, s)$ to the service s on server j, we must only consider that fraction of the response time for each request in the total sum. Therefore, we multiply the response time of each request with $R(i, j, s)$ and $\lambda(i, s)$. In each iteration of our sum, we consider the response time generated by assigning a fraction of demand to a server node. In order to find the average system response time for all users and services combined, we normalize the numerator by dividing by the total user demand.

Calculating the time spent on server depends on the expected queue size and processing time. Using a M/G/1 queuing system as an example and assuming that we have found a load distribution matrix, the expected processing time can be calculated as follows; $T_{proc.} = \frac{\left(\lambda * \overline{T_{j,s}^2}\right)}{2*(1-\rho)} + \overline{T_{j,s}}$ (Pollaczek-Khinchin mean value formula) which is the sum of the average queue delay and the average service time. We use ρ to represent the total incoming request rate on node j divided by the service rate. Now consider that we are only assigning a fraction $R(i, j, s)$ of demand to service instances, we find $\rho = \overline{T_{j,s}} * \sum_{i \in N_c} \lambda(i, s) * R(i, j, s)$ where $1/\overline{T_{j,s}}$ is the service rate.

We add a constraint to guarantee that the demand of each client node is fully satisfied:

$$\forall\ i \in N_c,\ \forall\ s \in S : \sum_{j \in N_S} R(i, j, s) = 1 \quad (2)$$

Our goal is to find a load distribution matrix R which minimizes the average system response time and we use (1) to measure the quality of a load distribution. However, as we assign fractions of demand to servers we also create a infinitely large solution space of floating numbers. Therefore we present a search heuristic to find a solution in a plausible time.

C. Simulated Annealing

Simulated Annealing (SA) is a search heuristic to scan a large solution space and return the best found solution in a finite time. The principle of SA is to explore many possible solutions at the start, even if the new solution is worse than the previous one, while towards the ending SA focuses on a smaller search space around the current best solution.

The higher the starting temperature, the more solution space we explore before converging to local minima. However,

TABLE I
PARAMETERS USED FOR *SA*

Parameter	Value	Description
T	10 000	The temperature decides the likelihood of accepting a solution worse than the current best one. At higher temperatures SA is more likely to accept a worse solution to continue exploring the search space.
T_{stop}	1	The temperature at which SA stops exploring the search space and returns the best found solution.
repetitionCount	2	This variable determines how many solutions are explored at one temperature value.
coolingRate	0.01	The speed at which the temperature decreases.
Delta (Δ)	0.1	Indicates the amount of change made to a solution when generating neighboring solutions.

the starting temperature along with the cooling rate heavily influence the execution time before a solution is returned. Therefore, we performed a parameter sweep to evaluate the performance increase when running SA for a longer time and found a suitable parameter set for our experiments, presented in Table I. For the topologies used in our experiments, the execution time of SA is between 1 second for a small topology and 60 seconds for the largest topologies. The characteristics of these topologies are described in more detail in section V-B.

For a more detailed explanation of Simulated Annealing we refer to [27].

D. Benchmark algorithms

To evaluate the performance of our *SA* implementation, we implement two static selection approaches and one dynamic selection algorithm, presented in the paragraphs below.

1. Greedy Algorithm. User demand is assigned to servers by prioritizing client-server pairs with the smallest latency. This selection approach starts with the client-server pair inducing the smallest network latency and assigns that user's demand to the respective server until either the server capacity (maximum amount of requests processed per time unit) is met, or until all client demand has been fully assigned. Although intuitively we expect this approach to induce the least latency, our results show cases where the overall response time would benefit from a non-greedy selection by prioritizing certain client-server pairs with slightly higher network latency. We discuss these results in section V.

2. Equal Share. As *Greedy* fully utilizes nearby servers before using a more distant server, we also investigate the performance of an equal distribution (*Equal*) where each server is assigned an equal amount of demand.

3. Joint Shortest Queue (JSQ). Any static selection algorithm is sensitive to unexpected load conditions. A dynamic selection approach allows components (in our case, the service routers) to react to changing network and server conditions at runtime but requires both accurate measurements to be available when assigning a request to a server. To avoid scalability problems, most dynamic algorithms only consider a smaller amount of metrics when assigning a request. One of these dynamic algorithms, Joint Shortest Queue (*JSQ*), is an often used selection algorithm for server farms [28] which only considers the server queue size. When a request arrives on a selection component it is assigned to the server with the least requests in its queue to minimize the mean response time on every server [29].

Contrary to (1), JSQ does not consider any network metrics. In section V-D we study whether or not the additional network metrics considered in the static *SA* selection algorithm can outperform the dynamic responsiveness of *JSQ*.

IV. Populating the service router forwarding tables

The QuLa service selection algorithm returns a load distribution matrix R which maps fractions of all user demand to service instances. The goal is for the service routers to forward requests exactly as dictated by the load distribution matrix and induce the same response time as predicted with (1). We foresee a scalability problem if each forwarding table must consider both the source of the request as well as the desired service name (we refer to this method as *source-based routing*, illustrated in Fig. 2). Similar to the hop-based forwarding scheme from CCNx, we propose that service routers only consider the desired service name when selecting the next hop to forward a request to.

In this section we present a statistical load-balancing method used by each service router on the path to a service instance, without considering the source of the request. We claim that this approach induces almost the same overall average response time as induced by *source-based routing* when accurately following the QuLa load distribution matrix. As our load-balancing technique is based on taking a weighted average of the QuLa load distribution, we refer to this as the *QuLa weighted average* approach. In section V-C we evaluate the performance degradation caused by using approximated values in QuLa weighted average compared to *source-based routing*.

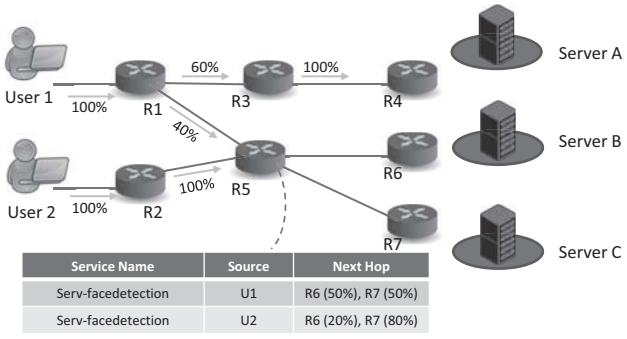
A. VARIANT 1: SOURCE-BASED ROUTING

Using this approach, each service router is configured with a separate forwarding table for each client node as dictated by the QuLa load distribution matrix. As described in our problem statement, $\lambda(i, s)$ is the total demand from client i for service s and $R(i, j, s)$ is the fraction of that demand assigned to server j. Now consider $P_k^{in}(i, s)$ the incoming percentage of $\lambda(i, s)$ on service router k, and $P_{kl}^{out}(i, s)$ the percentage of $P_k^{in}(i, s)$ forwarded to router l on router k. Initially, all values of P^{in} and P^{out} are set to zero. We now configure the forwarding tables as follows: (1) for each client-server pair (*i,j*) and a given service s, find the path with lowest network latency between i and j. (2) For each router k on the selected path, add the value of $R(i, j, s)$ to both $P_k^{in}(i, s)$ and $P_{kl}^{out}(i, s)$, with l being the next service router on the path to j. (3) Once we iterated through

each pair *(i,j)* for service *s*, we write $P_{kl}^{out}(i,s)$ as fraction of $P_{k}^{in}(i,s)$ to normalize all values in range [0,1].

We illustrate a sample configuration in Fig. 2: 40% of user 1's demand is sent to R5, which in turn forwards 50% to R6 and 50% to R7. As dictated by the service selection, 20% of user 1's demand reaches zone B and 20% reaches zone C.

The forwarding table of a service router *k* is populated by setting $P_{kl}^{out}(i,s)$ as outgoing fraction for each pair (i,s) and each neighbor router *l*.



Service Name	Source	Next Hop
Serv-facedetection	U1	R6 (50%), R7 (50%)
Serv-facedetection	U2	R6 (20%), R7 (80%)

Fig. 2. Source-based routing visualized.

B. Variant 2: Weighted Average

To overcome the scalability challenge caused by the large amount of state required in *source-based routing*, we propose a new forwarding table configuration method which does not require the request source to be considered. We take a weighted average of the values calculated in the previous paragraph to eliminate the source address from the equation. By keeping the amount of traffic on each link close to the predicted values we aim to minimize the performance degradation caused by this averaged approximation.

Consider the values $P_{k}^{in}(i,s)$ and $P_{kl}^{out}(i,s)$ which represent the fraction of demand from client *i* for service *s* expected to arrive at router *k* and the outgoing percentage on each link respectively. Using $\lambda(i,s)$ as the amount of traffic generated by client *i* for service *s*, we find the total incoming demand for service *s* on router *k* to be $D_{k}^{in}(i,s) = \sum_{i \in N_c} \lambda(i,s) * P_{k}^{in}(i,s)$, and the outgoing demand on the link to router *l*, $D_{kl}^{out}(i,s) = \sum_{i \in N_c} \lambda(i,s) * P_{k}^{in}(i,s) * P_{kl}^{out}(i,s)$. Taking the weighted average to eliminate the source address, we get

$$P_{kl}^{out}(s) = \frac{D_{kl}^{out}(s)}{D_{k}^{in}(s)} = \frac{\sum_{i \in N_c} \lambda(i,s) * P_{k}^{in}(i,s) * P_{kl}^{out}(i,s)}{\sum_{i \in N_c} \lambda(i,s) * P_{k}^{in}(i,s)} \quad (3)$$

where $P_{kl}^{out}(s)$ now represents outgoing percentages while considering only the service name as input. Forwarding tables are populated with the service names as input and the respective values of $P_{kl}^{out}(s)$ as output. Using the newly calculated values of $P_{kl}^{out}(s)$, the amount of traffic on each link and thus the load induced on each server approximates the same load distribution as *source-based routing*, keeping the system response time near the expected value.

To avoid routing loops after eliminating the source address from the forwarding tables, traffic should only follow links belonging to a minimum spanning tree of the network graph. A minimum spanning tree is constructed with a minimal subset of edges which has the lowest overall edge weight to connect every vertex of a graph. We use Kruskal's algorithm [30] to construct a spanning tree and as edge weight we use the inverted amount of expected traffic on each edge, based on the service selection load distribution. Using this metric as edge weight, we ensure that edges which are important for an efficient load distribution are included in the spanning tree.

V. Simulation Results

In this section we present our results on the performance of our service selection algorithms achieved through simulations on several network topologies. First, we describe the different network topologies used in our simulations in section V-A. Next, in section V-B we describe our simulation setup followed by a comparison of both *source-based routing* and *QuLa weighted average* in section V-C. Last, in section V-D we study the performance of the static QuLa algorithm which runs once but uses both network and server metrics, in comparison with the dynamic *JSQ* algorithm based on only server queue size.

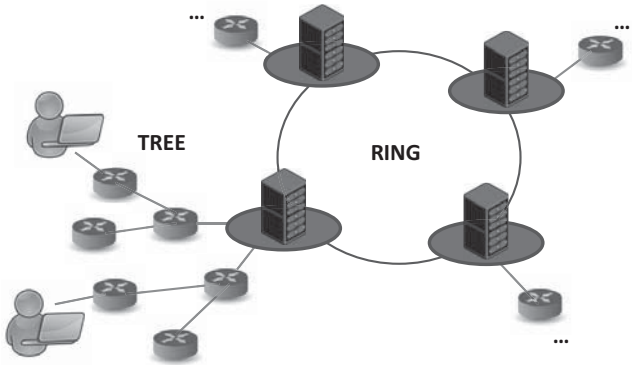


Fig. 3. A visualization of the simulated DSL-inspired networks.

A. Network topology

Our simulations are performed on variations of two network topologies: (1) Digital subscriber line (DSL) networks and (2) the European GTS CE topology. In a Digital subscriber line (DSL) network, clients are wired to one regional connection point and a ring of connection points is used to connect all the users on the network. We simulate this by connecting all servers to a ring and each server is the root of a network tree where the clients are located at the leafs. This type of network is illustrated in Fig. 3. The European GTS CE topology is a real-life topology available in the dataset of the Topology Zoo [31], an ongoing project to collect data network topologies from around the world. To approximate the edge latency between nodes in the GTS CE topology we used the Haversine distance and assumed that information travels at the speed of light.

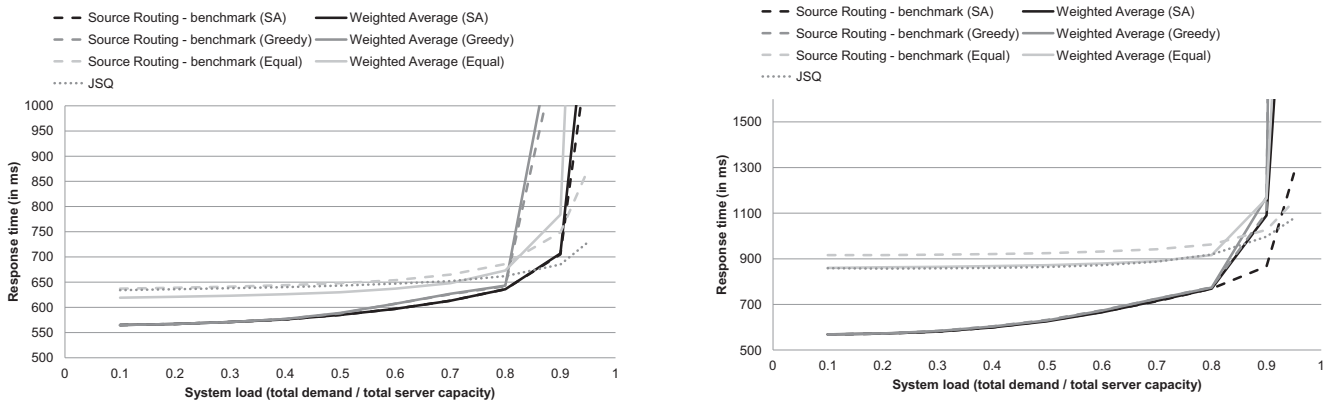


Fig. 4. response times using source-based routing and weighted average for (a) a DSL inspired network and (b) the European GTS CE topology.

For both topologies, we generate 50 sample networks with different client-server placement and load distributions. The characteristics of each sample network are identical, as shown in Table II.

TABLE II
NETWORK CONFIGURATION OF SIMULATED TOPOLOGIES.

	DSL	GTS CE
#Service routers	200	130
#Clients	10	10
#Servers	3	3
Router edge degree	2	2
Avg diameter	8	10
Link latency distribution	uniform [10,100]	Haversine
Service time	25ms	

B. Simulation setup

Our simulations are based on an extension of CloudSim [32], a framework for modeling and simulating cloud computing infrastructures and services. We extended each CloudSim datacenter object with a forwarding table and exactly one virtual machine for each service hosted on that datacenter so that multiple services do not influence the service time. In our implementation, client applications are implemented as a Poisson process, requests are handled in order of arrival and with a deterministic service time ($\overline{T_{j,s}^2} = \overline{T_{j,s}}^2$). This allows us to model our servers as an M/D/1 queue, a special case of M/G/1. In this case, the server processing time described in section III-B can be written as $T_{proc.} = \frac{(2-\rho)}{2*(1-\rho)} * \overline{T_{j,s}}$.

The service selection algorithms described in section III run on a centralized broker which contains the demand patterns and network characteristics prior to the simulation start. The load distribution matrix is mapped to the forwarding tables as described in section IV. Next we run the simulation using the known demand pattern and measure the average system response time as quality representation of each algorithm. All simulations are performed on the iLab.t Virtual Wall [33] using a server with a Hexacore Intel E5645 (2.4GHz) CPU, 24GB RAM, 1x 250GB hard disk and 1-5 gigabit network interface cards.

C. QuLa weighted average vs. source-based routing

We study the performance of the different service selection algorithms presented in section III using the average response time as quality measurement. Using *source-based routing* we are able to map the selection to service routers and achieve the desired response time. However, to reduce the router state scalability problem we investigate the performance degradation of *QuLa weighted average* which attempts to achieve the same average response time as *source-based routing* but with less state required. To avoid routing loops, weighted average routing was simulated on a minimized spanning tree as described in section IV-B.

Consider Fig. 4, the measured response times (Y-axis) for a set of fixed load values (X-axis) are illustrated for both *source-based routing* (dashed) and the *QuLa weighted average* approach (solid). Each curve color represents a different service selection used to obtain the load distribution. For each topology type we generated 50 sample networks and averaged the response times to obtain the values shown in Fig. 4.

First, we study the dashed curves (*source-based routing*) to compare the performance of the selection algorithms without taking the performance degradation of *QuLa weighted average* into account. A *Greedy* selection works well under low load conditions as it minimizes the network latency while the servers run stable. However, *Greedy* assigns demand to a server until it reaches its maximum capacity before sending requests to a more distant server, making it more sensitive to peaks at high system load and resulting in high response times (above 80% in Fig. 4). *Equal* is the opposite of *Greedy*, it assigns load equally to all servers from the beginning and does not consider network latency. *Equal* does not fully utilize nearby servers, inducing higher response time for low load conditions, but is able to keep the system stable when load increases (servers only operate near maximum load when the total system load approaches 95%). With a good set of

parameters, *SA* will lean more towards a greedy selection for low load conditions and converges towards an equal distribution at high load values. *JSQ* performs similar to *Equal* but is able to make a more fine-grained decision by examining the measured queue sizes at runtime. We discuss *JSQ* in more detail in section V-D.

Next, consider the solid curves representing the performance of the *QuLa weighted average* forwarding approach. Compared to *source-based routing* we can no longer assume that the measured system response time equals the expected response time, as the network is now limited to a minimal spanning tree and the forwarding tables contain reduced state with approximated values. Using *QuLa weighted average* with a minimal spanning concentrates the traffic flow on fewer and longer paths towards the servers. Each server can decide to process a request or to forward it to the next server. The more servers located on a path towards another server, the higher the chances that a request is processed by one of the intermediate routers. Using *Equal*, a percentage of requests assigned to the furthest servers will be processed by the intermediate servers on the path, making this a more greedy approach which favors the network latency. This greedy effect is visible in Fig. 4 where *Equal* performs better with *QuLa weighted average* than with *source-based routing* for load values below 80%, as the network latency is most important in that area.

The small deviation between *source-based routing* and *QuLa weighted average* at high load values can be explained by the use of a spanning tree which will force traffic to pass more intermediate servers than expected. Because of this, nearby servers will operate at maximum capacity before more distant servers reach their maximum, which can reduce their availability and increase the response time. For other load cases we observe that the *QuLa weighted average* approach is able to approximate the average system response time of *source-based routing* but with reduced state in the service router forwarding tables.

D. Static QuLa weighted average vs. dynamic JSQ

In this section we determine the performance trade-off between our static QuLa selection algorithm which runs once prior to the simulation start and the dynamic *JSQ* algorithm. To evaluate *JSQ* we consider the best case scenario where each service router knows the server queue sizes upon request arrival and forwards each request on the lowest latency path to the server with the smallest queue.

Using Fig. 4 we observe that *JSQ* has a similar pattern to *Equal* as both algorithms tend to keep the server load equally distributed. *JSQ* generally performs better than *Equal* due to load-balancing at runtime with an exact knowledge of queue sizes upon request arrival. This performance difference becomes more visible for higher system load as more precision is required and thus *JSQ*'s runtime information becomes more valuable. Compared to *SA*, we notice that *JSQ* generally performs worse as it does not consider network latency which is an important factor for low to average system load. When the systems are under high load and the time spent on server becomes more critical, *JSQ* is the better performing algorithm (90% mark in Fig. 4).

Consider the response time distribution for both *SA* and *JSQ* in Fig. 5 for (a) 50% and (b) 90% server load. When the system runs stable at 50% load, small peaks in user demand will not heavily affect server queuing times, making the network latency an important factor to consider. For these scenarios, *SA* is able to deliver a better response time for all the users in the network (Fig. 5 a). As the load increases it becomes more important to load-balance requests across multiple servers to avoid overloading the closest instances. For these higher load values we observe that 80% of the users experience a lower response time with *SA* than with *JSQ*. However, the remaining 20% of users in *SA* experience large response times due to overloaded servers, up to 2-3 seconds. *JSQ* keeps the servers stable and therefore guaranteed a maximum response time of 1 second to 99% of it's users (Fig. 5 b).

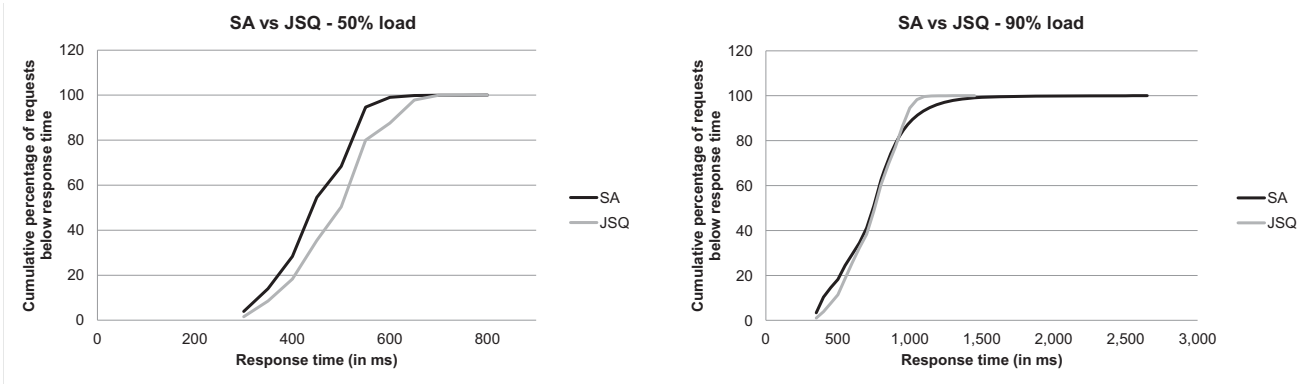
Assuming that the network latency is not negligible compared to the server queuing times, we conclude that our proposed *SA* selection algorithm, in combination with the *QuLa weighted average* configuration approach (cfr. section IV), results in a lower response time than obtained through *JSQ*. Although *JSQ* requires less information monitoring than *SA*, our static selection only requires one run for a given setup while *JSQ* must make a selection upon each request arrival.

VI. Conclusion and future work

In this paper we presented a name-based forwarding approach to forward requests to service instances so that the average response time is minimized. First, we described a service selection algorithm which maps user demand to service instances based on both server and network characteristics. To avoid scalability problems, we use QuLa weighted average to reduce the state required in service routers. Next, we studied the performance degradation of this reduced state compared to source-based routing. Last, we study the performance difference between our static selection algorithm and a dynamic selection at runtime.

We conclude that our weighted average approach is able to approximate the same response time as source-based routing but with less state required. Our service selection algorithm implemented as *SA* is able to find a near-optimal load distribution regardless of the network characteristics or amount of user demand. However, for high load values we conclude that the dynamic *JSQ* algorithm performs better due to its ability to keep the server load equally distributed at runtime, whereas the approximation of *SA* makes the system less stable.

In future work we focus on the placement of resources across the network. There is plenty of existing research on placement algorithms when the location and demand pattern of users is known. However, this problem becomes more complicated when users can come online from any location and still desire a response below a certain threshold. We will focus on the deployment of resources across the network so that a minimum service quality can be guaranteed to users, regardless of their location or demand.



(a) (b)

Fig. 5. response times using a static configuration found by *SA* and a dynamic *JSQ* selection for a DSL-like topology running (a) at 50% and (b) 90% load.

Acknowledgments

This project was partly funded by the UGent BOF-GOA project "'Autonomic Networked Multimedia Systems"', by the FWO-V project "'SPEC: Intelligent SuPer-Elastic Clouds"' and by the 7th Framework Programme of the European Commission through the FUSION project under grant agreement no. 318205.

References

[1] E. Nygren *et al.*, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[2] G. Carofiglio *et al.*, "From content delivery today to information centric networking," *Computer Networks*, vol. 57, no. 16, pp. 3116–3127, 2013.

[3] B. Ahlgren *et al.*, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, 2012.

[4] L. Zhang *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[5] N. Fotiou *et al.*, "Developing information networking further: From psirp to pursuit," in *Broadband Communications, Networks, and Systems*. Springer, 2012, pp. 1–13.

[6] W. K. Chai *et al.*, "Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services," *Communications Magazine, IEEE*, vol. 49, no. 3, pp. 112–120, March 2011.

[7] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, Aug. 2007.

[8] C. Dannewitz *et al.*, "Network of information (netinf): An information-centric networking architecture," *Computer Communications*, vol. 36, no. 7, pp. 721 – 735, 2013.

[9] Y. Xu *et al.*, "A novel cache size optimization scheme based on manifold learning in content centric networking," *Journal of Network and Computer Applications*, vol. 37, no. 0, pp. 273 – 281, 2014.

[10] W. K. Chai *et al.*, "Cache less for more in information-centric networks (extended version)," *Computer Communications*, vol. 36, no. 7, pp. 758 – 770, 2013.

[11] C. Dannewitz *et al.*, "Hierarchical dht-based name resolution for information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 736 – 749, 2013.

[12] OGF, *Open Cloud Computing Interface | Open Standard | Open Community*, OGF Std., 2013. [Online]. Available: http://occi-wg.org

[13] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[14] A.-F. Antonescu *et al.*, "Sla-driven predictive orchestration for distributed cloud-based mobile services," in *Communications Workshops (ICC), 2013 IEEE International Conference on*, June 2013, pp. 738–743.

[15] T. Braun *et al.*, "Service-centric networking," in *Communications Workshops (ICC), 2011 IEEE International Conference on*, 2011, pp. 1–6.

[16] P. Smet *et al.*, "Qula: Queue and latency-aware service selection and routing in service-centric networking," *Communications and Networks, Journal of*, vol. 17, no. 3, pp. 306–320, June 2015.

[17] M. Freedman *et al.*, "Serval: An end-host stack for service-centric networking," in *Proc. USENIX NSDI*, 2012.

[18] M. J. Freedman *et al.*, "Service-centric networking with scaffold," *Princeton University, September*, 2010.

[19] K.-W. Lee *et al.*, "Adaptive server selection for large scale interactive online games," *Computer Networks*, vol. 49, no. 1, pp. 84–102, 2005.

[20] P. Wendell *et al.*, "Donar: decentralized server selection for cloud services," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4. ACM, 2010, pp. 231–242.

[21] H. A. Alzoubi *et al.*, "Anycast cdns revisited," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 277–286.

[22] A.-J. Su *et al.*, "Drafting behind akamai (travelocity-based detouring)," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 435–446, 2006.

[23] L. Wang *et al.*, "Ospfn: An ospf based routing protocol for named data networking. university of memphis and university of arizona," Tech. Rep, Tech. Rep., 2012.

[24] C. Li *et al.*, "Ant colony based forwarding method for content-centric networking," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, March 2013, pp. 306–311.

[25] S. Shanbhag *et al.*, "Soccer: Services over content-centric routing," in *ACM SIGCOMM Information-Centric Networking (ICN) workshop, Toronto, Canada*, 2011.

[26] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[27] K. A. Dowsland and C. Reeves, "Modern heuristic techniques for combinatorial problems," *Simulated Annealing. In Reevees, CR, Editor, John Wiley and Sons, NY, USA*, no. 2, 1993.

[28] S. Abimannan *et al.*, "Join-the-shortest queue policy in web server farms," *Global Journal of Computer Science and Technology*, vol. 10, no. 4, 2010.

[29] V. Gupta *et al.*, "Analysis of join-the-shortest-queue routing for web server farms," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1062 – 1081, 2007, performance 2007 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation.

[30] J. Kruskal, Joseph B., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. pp. 48–50, 1956.

[31] S. Knight *et al.*, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765–1775, 2011.

[32] (2013) The clouds lab: Flagship projects - gridbus and cloudbus. [Online]. Available: http://www.cloudbus.org/cloudsim/

[33] (2013) ilab.t virtual wall | internet based communication networks and services. [Online]. Available: http://www.ibcn.intec.ugent.be/content/ilabt-virtual-wall