

# SoCCeR: Services over Content-Centric Routing

Shashank Shanbhag<sup>†</sup>, Nico Schwan<sup>◊</sup>, Ivica Rimač<sup>◊</sup>, Matteo Varvello<sup>\*</sup>

<sup>†</sup> University of Massachusetts, Amherst, MA, USA

<sup>◊</sup> Bell Labs, Alcatel-Lucent, Stuttgart, Germany

<sup>\*</sup> Bell Labs, Alcatel-Lucent, Holmdel, USA

sshambha@ecs.umass.edu, {nico.schwan, ivica.rimac, matteo.varvello}@alcatel-lucent.com

## ABSTRACT

Content-Centric Networking (CCN) and Service-Centric Networking (SCN) are two novel paradigms that seek to change the way content and services are perceived. CCN is centered around content distribution, while SCN focuses on dynamic customization of in-network services.

We present SoCCeR—Services over Content-Centric Routing. SoCCeR extends CCN with integrated support for service routing decisions leveraging ant-colony optimization. SoCCeR adds a control layer on top of CCN for the manipulation of the underlying Forwarding Information Base (FIB). Without affecting content request and retrieval functionality of CCN, SoCCeR adds SCN functionality to CCN. Illustrating the interaction of SoCCeR with the routing layer, our simulation results show that SoCCeR routes service requests selectively to service instances with lighter loads and is highly responsive to network and service state changes.

## Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design; C.2.2 [Computer-Communication Networks]: Network Protocols

## General Terms

Algorithms, Design, Performance

## Keywords

service-centric, ant colony optimization, content-centric, services, service routing

## 1. INTRODUCTION

The Internet architecture has evolved substantially; though based on a host-centric communication model, it now mostly serves information-centric applications, e.g., Hulu, Facebook. This change has been enabled by the introduction of various ad hoc and application-specific solutions (e.g., CDNs [12] and P2P) that leverage techniques like content and service replication, caching, service-

load balancing and request routing. However, the interaction between many distributed components increases complexity and introduces its own set of problems, a barrier toward introduction of additional functionality, and high cost of managing and operating networks.

The research community addresses this problem with two paradigmatic shifts: *Content-Centric Networking* (CCN) and *Service-Centric Networking* (SCN). While CCN focuses on content dissemination and networking based on content identifiers rather than content hosts, SCN aims at integrating customized services into the network on demand with the help of mostly central controllers. However, separately treating content and services can barely account for their reciprocal relation — a service generates new content or performs a set of functions on existing content, whereas content discovery, caching, etc., are forms of content-centric services. Thus, there is a need to bridge the gap between content-centric and service-centric networks.

In this paper, we bring content-centric and service-centric networking together in order to achieve the vision of a truly information-centric network. We present *SoCCeR*—Services over Content-Centric Routing, the design of which is motivated by the goals of extending CCN for the support of services. CCN inherently cannot support services because of the emphasis on retrieving content. When multiple copies of a content item are available in the network, a request can be forwarded to all the copies resulting in redundant retrievals. This can turn out to be prohibitively expensive if a service request results in redundant consumption of service node resources (e.g. CPU cycles, memory, energy). Thus, CCN needs to be extended with mechanisms that can prevent duplicate invocation of services for one request and provide SCN functionality while eliminating the centralized controller of SCNs in favor of a distributed, scalable and fault-tolerant system.

To achieve the above objectives, SoCCeR works as a control layer on top of CCN that can manipulate the underlying Forwarding Information Base (FIB). The FIB manipulation allows us to perform distributed “service selection,” i.e., routing a service request to the best service instance at any point in time. For this purpose, SoCCeR leverages a distributed mechanism for gathering service information based on Ant Colony Optimization (ACO). The collected information is used to calculate heuristic measures followed by manipulation of FIB entries in the underlying name-based routing layer. As a result, the FIB entry for a service always points to a preferred instance of the service according to some metric (service load, path congestion, etc.). This is done without affecting content routing functionality.

The main contributions of our research can be summarized as follows: (1) we show how information-centric networks can be effectively achieved by integrating distributed service selection with

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICN'11, August 19, 2011, Toronto, Ontario, Canada.

Copyright 2011 ACM 978-1-4503-0801-4/11/08 ...\$10.00.

name-based routing; (2) we present the design of such a solution and a prototype implementation based on CCNx [4], a well-maintained and very popular CCN code base; and (3) through simulations we evaluate our approach and compare it with a Vanilla solution based on content-centric routing.

The remainder of the paper is organized as follows. Section 2 provides background information on content-centric and service-centric networking. The design details of SoCCeR along with a summary of the main research challenges are presented in Section 3. Section 4 discusses the prototype implementation as well as experimental methodology and results. In Section 5, we summarize previous work in service-centric networks before concluding the paper in Section 6.

## 2. TECHNICAL BACKGROUND

Named Data Networking (NDN) [9] is one of the most recent and comprehensive CCN designs. In this paper, we focus on the design of a solution based on NDN, for which we now briefly describe the relevant technical background. The principles of our approach are applicable to a wider class of CCN designs and therefore, in the rest of the paper, we use the term CCN instead of NDN.

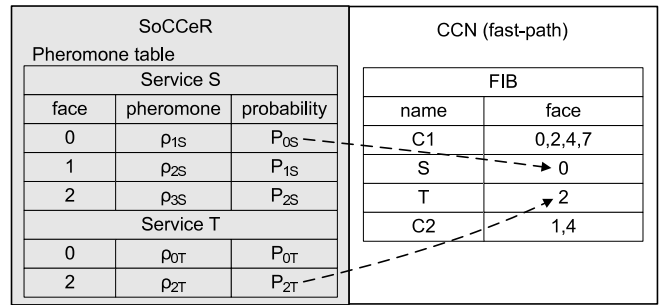
In CCN, content items are addressed directly by their hierarchical names in the networking layer. Content providers announce content availability through flooding within their local network, while routers use BGP-like content prefix announcements for inter-domain routing. CCN routers maintain a Forwarding Information Base (FIB) that associates content names to next hops (*faces*). When an application issues a content request, one (or more) *Interest* packets are generated. An Interest packet contains the name of the requested content, which routers use to select the face via longest prefix match. While propagating toward one or more content sources, Interest packets leave trails of “bread crumbs” used for routing corresponding *Data* packets on the reverse path to the requesting end points.

When multiple copies of a content item are available in the network, a router may associate multiple faces with the same content name. An Interest packet is then duplicated to all faces and may trigger duplicate Data packets. While a feature providing diversity to data delivery, redundant Interest packets may cause consumption of potentially expensive and scarce resources on service nodes (e.g., CPU, memory, energy). Thus, we argue that for service selection the routing layer needs to implement mechanisms that can prevent duplicate invocation of services for one request.

In today’s IP based service-centric architectures service selection is primarily based on central controllers. They act as intermediaries between clients and services with global knowledge about network topology, service instances and their hosts. State information such as service load, service node and link resources, etc., are routinely queried. Such a design is inconsistent with the principles of content-centric routing, and more importantly, it has significant limitations, with respect to scalability, large overhead in control traffic, and a single point of failure renders it susceptible to coordinated attacks. Therefore, we argue that a distributed solution leveraging a name-based router’s capability of making independent routing decisions on service names is superior to a central solution for CCNs.

## 3. SOCCER DESIGN

The main research challenge, which we call “Service Selection”, can be stated as follows: “Given many instances of a service sharing the same name, how do you route a request to the best instance?”. We assume that traffic targeting a shared name is not



**Figure 1: SoCCeR node design.** In this figure,  $(P_{0S} > P_{2S} > P_{1S})$  and  $(P_{2T} > P_{0T})$

restricted from being routed to independent service instances, i.e., there is no session or state dependency for subsequent requests.

The solution to the service selection problem depends on two aspects: the metrics that determine the cost of the solution, and the method used to gather the information about these metrics.

### 3.1 Overview

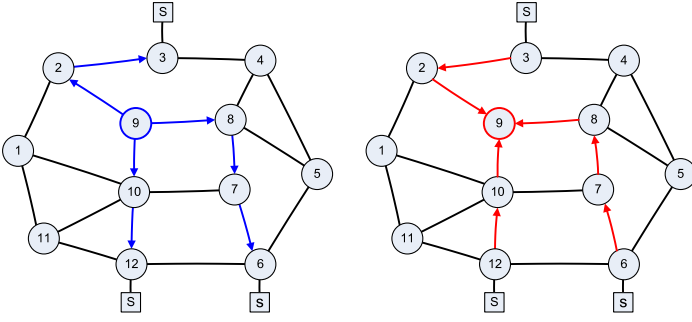
Our approach towards the design of SoCCeR is based on Ant Colony Optimization (ACO) [6], a decentralized and probabilistic optimization heuristic adapted to the purpose of retrieval of information about various services and their replicas in a network.

ACO algorithms are versatile enough to be applied in CCN. Many ACO algorithms have been applied to a wide variety of different optimization problems, ranging from scheduling, network routing, assignment problems and data mining to name a few. They have been proven to be highly adaptive to network state changes, be able to find optimum solutions to a given problem and converge quickly compared to other approaches. Other distributed approaches for the identified problem, like link state routing protocols, would not work for CCN. Here each node needs a complete map of the network connectivity implying each node has an address, which is not the case in CCN. While distance vector protocols, in principle, could be applied to this problem, we avoid using them due to their inherent problem of routing loops and the count-to-infinity problem.

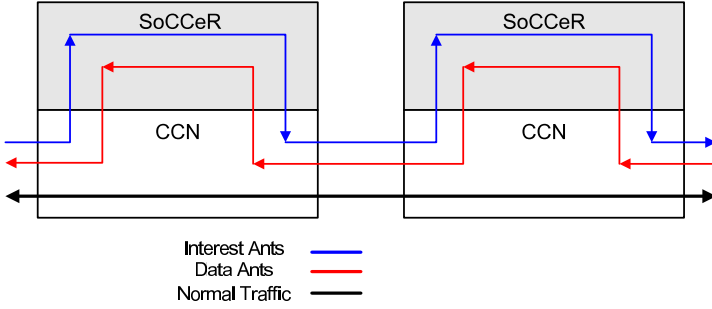
ACO algorithms mimic the collective foraging behavior of ant colonies. Individual ants are rather unsophisticated with very limited memory and exhibit random behaviors. However, collectively ant colonies are able to consistently find the shortest paths between the colony and various food sources. They are able to do this through “stigmergy”, a self-organizing mechanism that uses indirect communication between individual agents. In ant colonies, stigmergy is achieved by laying a trail of pheromones (chemical signals) which is sensed and enforced by other ants. Less optimal pheromone trails (longer paths) weaken over time through atmospheric evaporation as the time between enforcements is longer. We realize artificial stigmergy in our design by using internal data structures that are updated by ant-like agents traversing the nodes. In the following subsections, we detail the design of SoCCeR.

### 3.2 Node Design

The FIB entries in the CCN layer associate a service name with a list of faces through which the service can be reached. The problem of service selection then reduces to selecting the best face in this list of faces. To make optimal decisions on the face, a number of factors have to be taken into consideration, e.g., service load, path congestion, available bandwidth and jitter. We exploit the inherent advantages of CCN and extend it by ACO to gather this informa-



**Figure 2: Adaptation of Ant Colony Optimization for service-centric networking.**



**Figure 3: Paths taken by Interest/Data Ants and normal traffic.**

tion in a distributed manner. We emphasize the fact that every node in the network acts independently and asynchronously, i.e., we do not assume synchronized clocks. Furthermore, we also assume that every node hosts the SoCCeR layer which has access to the collection of service names already instantiated in the network (through initial announcements when services are instantiated).

Every node that hosts the SoCCeR layer incorporates internal data structures called “pheromone tables” (Fig. 1). The pheromone table consists of the service name, the associated faces, the corresponding pheromone values, and the probabilities determining which face is chosen calculated from the pheromone values. For every service entry, the FIB in the CCN layer below is manipulated to point to the face with the highest probability value as calculated from the pheromone table for the service. The content entries in the FIB are left untouched. Therefore, all service requests are forwarded in the fast-path to the best face instead of being forwarded to all the faces as done in CCN.

### 3.3 Optimization Framework

At the start of a local time window every SoCCeR node periodically generates an “Interest Ant” destined toward a randomly selected service. The Interest Ant contains a time stack used to record forwarding time at each node it traverses. The node pushes the (“forwarding time”) onto the stack and sends it onto every face associated with the service in the FIB. A SoCCeR node that receives the Interest Ant pushes its own local time onto the stack and forwards it onto the highest probability face in the pheromone table. This continues until the Interest Ants reach the targeted service instances, as illustrated in the left graph of Fig. 2. To avoid stagnation, i.e., the continued reinforcement of current optimal paths possibly leading to congestion and reduction of probabilities of other faces, an Interest Ant is forwarded to a random face with a proba-

bility called the “exploration probability”. This allows the detection of previously unknown and possibly better paths to services.

After receiving an Interest ant, the SoCCeR layer on the service node generates a “Data Ant”, with a copy of the time stack and service status information (service load, memory, etc.). The Data Ant follows the breadcrumb left behind by the Interest Ant, as illustrated in the right graph of Fig. 2. At each SoCCeR node encountered, the SoCCeR layer retrieves the service status information and its forwarding time from the time stack. The retrieved forwarding time and the Data Ant reception time is used to calculate the time elapsed, i.e., the round-trip-time delay from the node to the service instance, which is indicative of path congestion. The delay along with service status information is used to update the pheromone value for the face on which the data ant arrived for the service name (called the service-face pair). The amount of increase is an inverse function of the path delay and service load. Heavily congested paths and service instances with high loads should be avoided, and therefore, the corresponding faces should be enforced as little as possible. An increase in pheromone values for a particular service-face pair results in the increase in probability for the pair and a corresponding decrease in the probabilities of the other service-face pairs for that service. Furthermore, at the end of each time window, all pheromone values for all service-face pairs are evaporated by the “evaporation factor” making sure that the best faces are not continually enforced resulting in the other faces being ignored. Path congestion is detected due to large round-trip-times. Service or link failures are detected because of the absence of a data ant response.

This process is repeated at every node the Data Ant traces en-route to the originating node. Therefore, the pheromone table on a node is not only updated by its own Data Ant, but also by the Data Ants generated by other nodes traversing it. For example, in Fig. 2, node 9 pheromone tables are updated by its own Data Ants as well as those traversing through it. Interest and Data Ants always traverse the SoCCeR layer, while the normal content and service traffic is forwarded in the fast-path as shown in Fig. 3.

### 3.4 Metrics and Pheromone Update Equations

The pheromone updates are an inverse function of path metrics and service status. For simplicity, we use service load (denoted by  $l$ ) and the path delay (denoted by  $d$ ) between a node and the service as the metrics for service selection. SoCCeR is not limited to service load and path delay but can easily be extended to include other metrics as well.

Let  $G = (V, E)$  be the graph representing the network where  $V$  are the nodes and  $E$  are the edges in the graph. Let  $S$  be the number of service instances. The SoCCeR layer in every node  $i$ , maintains a pheromone table  $T_i^s$ , containing data associated with service  $s$  and all the faces  $N_i^s$  through which it can be reached. The normalized pheromone values  $\tau_{i,j}^s(l)$ ,  $\tau_{i,j}^s(d)$  and the probability  $P_{i,j}^s$  associated with the faces  $N_i^s$  represent the attractiveness of the face  $j$  for an Interest Ant traversing node  $i$  and destined towards service  $s$ .  $\tau_{i,j}^s(x)$  are normalized to 1 as:  $\sum_{j \in N_i^s} \tau_{i,j}^s(x) = 1$ , where  $s \in [1, S], \forall i \in V, x \in \{d, l\}$

Every node in the network generates an Interest Ant towards a randomly selected service,  $s$ , at the start of its time window,  $\delta_i$ . The Interest Ant is sent to all faces,  $N_i^s$ , through which the service is reachable. At each node  $i$ , the face,  $j$ , on which the Interest Ant is forwarded is decided based on the “exploration probability”  $\gamma$ . The value of  $\gamma$  determines whether the Interest Ant should be forwarded to a randomly selected face or to the face with the highest probability value  $P_{i,j}^s$  among all faces  $N_i^s$  for the target service,  $s$ . This probability is calculated as the normalized sum of the load and

delay pheromone values given by the equation:

$$P_{i,j}^s = \frac{(1 - \alpha)\tau_{i,j}^s(d) + \alpha\tau_{i,j}^s(l)}{\sum_{j \in N_i} ((1 - \alpha)\tau_{i,j}^s(d) + \alpha\tau_{i,j}^s(l))} \quad (1)$$

The weighting constant  $\alpha$  decides the contribution of the load and delay pheromone towards calculation of the probability. The choice of  $\alpha$  depends entirely on network policies and requirements of a service (QoS guarantees) and can be set for each service. A service provider whose priority is load balancing across instances will require  $\alpha$  to be higher. The pheromone values are continuously updated using the data gathered by the Data Ants and reflect the current and past status of the network and the services.

When a node  $i$  receives a Data Ant from a service  $s$  on face  $j$ , it extracts the service load and calculates the delay. The mean service load and delay are calculated using an exponential weighted moving average computed over all the values gathered by the Data Ants. The mean delay and service load are then normalized over all the faces for service  $s$ ,  $N_i^s$  to get  $d_{i,j}^s$  and  $l_{i,j}^s$ . The pheromone values for the face  $j$  on which the Data Ants arrive are incremented (enforced) according to the equation:

$$\tau_{i,j}^s(x) = \tau_{i,j}^s(x) + \Delta_x(1 - \tau_{i,j}^s(x)) \quad (2)$$

where  $\Delta_x$  is:

$$\Delta_x = -(x - 1) \exp(x), \quad x \in \{d_{i,j}^s, l_{i,j}^s\} \quad (3)$$

Ideally, a service request has to be forwarded to a service that is least loaded and lies on a path that is least congested. This is realized through Equations (2) and (3). Pheromone enforcement is an inverse function of service load and delay. Faces with low pheromone values and low observed service loads and delays are increased proportionally more than faces with larger pheromone values and higher observed loads and delays. This design ensures that a node quickly discovers new and better paths and reacts to changes. Equation 3 rewards lower values of load and delay while decreasing the amount of enforcement for higher values.

A Data Ant received from any face initiates the evaporation of the delay and load pheromones  $\tau_{i,j'}^s(x)$  for the other faces  $j'$ , by a constant factor as given by the following equation:

$$\tau_{i,j'}^s(x) = \tau_{i,j'}^s(x) - \rho\tau_{i,j'}^s(x), \quad j' \in N_i, \quad j' \neq j \quad (4)$$

where  $\rho$  is the ‘‘evaporation constant’’. If no data ant triggers evaporation, the node initiates pheromone evaporation on all faces at the end of the time window,  $\delta_t$ .

The routing overhead due to the ants is another factor we need to consider in the design. In SoCCeR, the routing overhead is a function of the Interest Ant generation time window ( $\delta_t$ ) and the number of hops the ants need to traverse to reach the service instances. The number of hops is a factor because at each hop, a small timestamp is pushed onto the timestack.

## 4. EVALUATION AND RESULTS

This Section presents a preliminary evaluation of SoCCeR. We compare SoCCeR with: a) *Randomized Service Selection* (abbreviated Random), where a router randomly forwards a service request towards one of the faces through which the service is reachable, and b) *Vanilla CCN* (abbreviated Vanilla), where a router forwards a service request toward all faces through which the service is reachable.

### 4.1 Methodology

We implement SoCCeR as an extension of the NDN prototype, named CCNx [4]. We modify CCNx interest and data packets in

order to include the time stack; moreover, we modify the CCNx protocol in order to account for SoCCeR traffic. We simulate a SoCCeR network by assigning nodes to different Unix sockets and building a realistic network topology via the GT-ITM tool [16]. We run our simulations on an Intel Core 2 Quad CPU Q9400 with 3 MB level-2 cache running at 2.66 GHz and 3 GB of memory.

We simulate a 50-node AS network where a link exists between any two nodes with a probability of 0.5. Each link is set a bandwidth of 100Mbps. Among the edge nodes, we randomly select 20 nodes as service nodes, i.e., host an instance of a service  $s$ , while the 30 remaining nodes are SoCCeR nodes, i.e., they generate and process Interest and Data Ants. Users are attached to the SoCCeR nodes and generate service requests which are uniformly distributed between 0 and 20 with exponentially distributed inter-arrival times with a mean of 1 second. A service request consumes one unit of a service node’s cpu resources and occupies those resources for a period that is exponentially distributed with a mean of 100 seconds. We set  $\alpha = 0.8$ ,  $\rho = 0.2$  and  $\gamma = 0.2$ . We treat *delay* as the product of the time elapsed between the sending of an Interest Ant and reception of the Data Ant and the number of hops traversed. Except where otherwise noted, each simulation lasts 10,000 seconds and is run for 10 trials.

## 4.2 Results

We start by analyzing how SoCCeR balances load among its nodes for a single service  $s$ . We assume that SoCCeR nodes generate Interest Ants for  $s$  at the start of a time window with duration  $\delta_t = 1$  second. In the simulation, service instances in SoCCeR, Random and Vanilla receive a total of 22,666, 24,499 and 45,603 service requests, respectively. The higher number of service requests in Vanilla is due to the fact that service requests are multiplicated at each hop much more than they are aggregated.

Fig. 4 shows the Cumulative Distribution Function (CDF) of the service load for SoCCeR, Random and Vanilla scheme at steady state. The vertical line shows the service load distribution achieved by an ideal service selection scheme: 20% service load at each node. Accordingly, SoCCeR successfully balances load between service instances, e.g., the service load distribution is between 10 and 30%. Conversely, the load distribution is between 24 and 60% for Vanilla. This is due to the fact that Vanilla forwards service requests to all reachable service instances, that in turn can receive multiple copies of the same request, one copy on each face. For the Random scheme, the load distribution is even worse than in Vanilla:

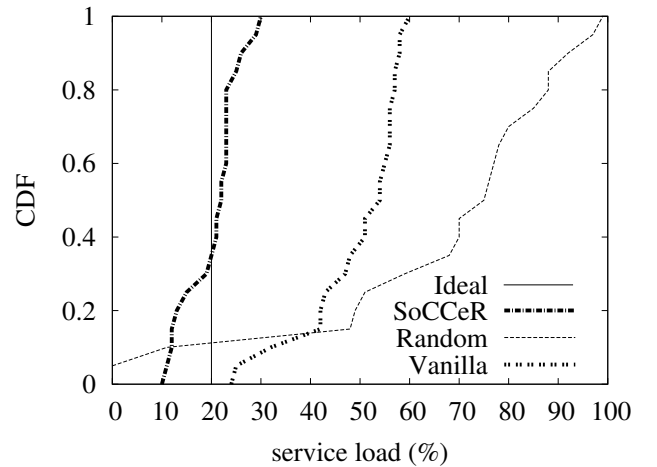


Figure 4: CDF of service load ; SoCCeR, Vanilla and Random service selection.

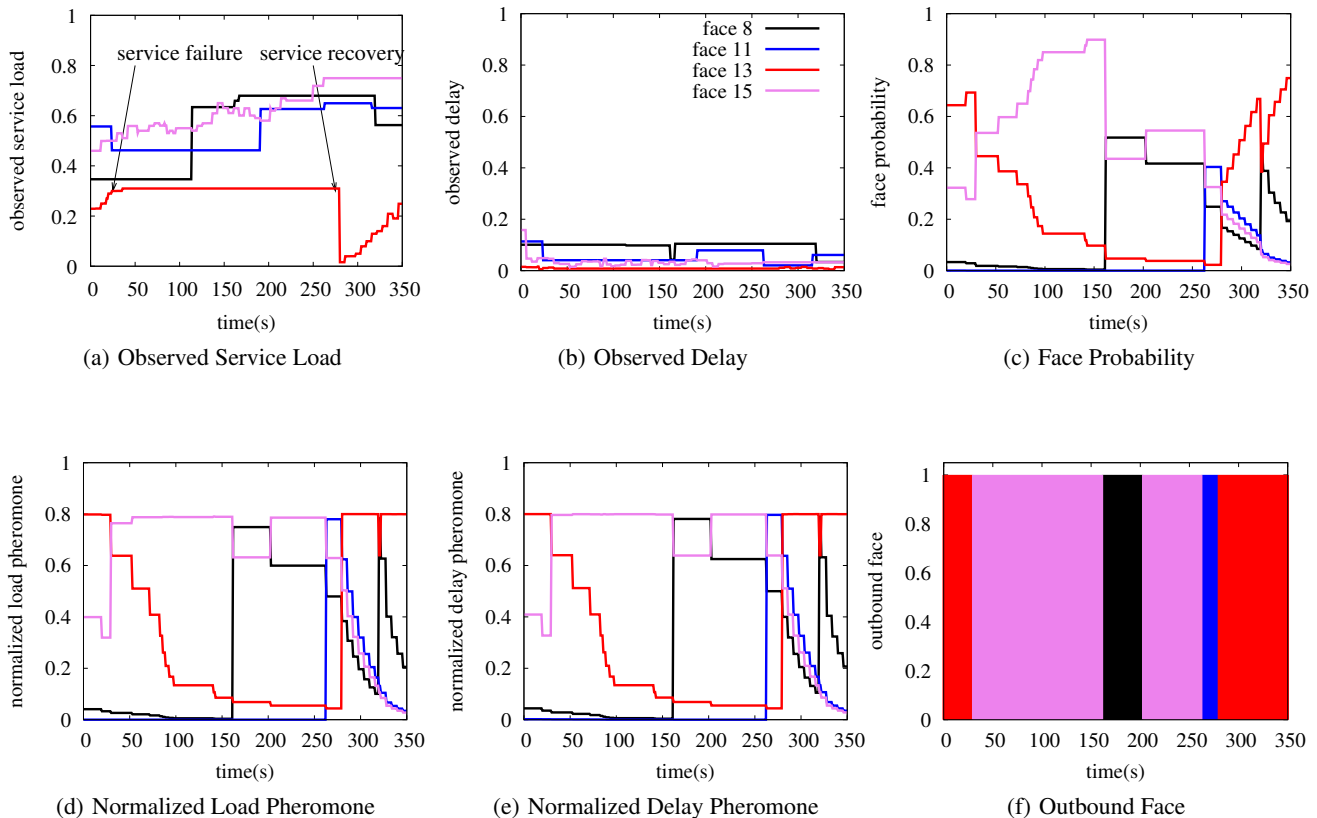


Figure 5: SoCCeR behavior on service failure and recovery.

5% of the nodes are not loaded at all, while 10% of the nodes are clearly overloaded, e.g., load > 90%.

We now analyze how SoCCeR reacts to service failure. To do so, we simulate a failed link or node failure by disconnecting a node where a service instance is running. As an observation point, we consider a SoCCeR node that can reach four instances of this service from face 8, 11, 13 and 15. At the beginning of the experiment, the service instance reachable on face 13 has the minimum load, 0.22-0.25% (Fig. 5(a)); for this reason, face 13 has the highest probability value, 0.65-0.7 (Fig. 5(c)), and 100% of the requests are forwarded toward face 13 (Fig. 5(f)). At  $t = 25$ s, the instance reachable from face 13 fails; consequently, the value of the load pheromone (Fig. 5(d)) and delay pheromone (Fig. 5(e)) associated with face 13 exponentially decrease. Almost instantaneously ( $< 1$  second), the pheromones for face 13 becomes lower than face 15 resulting in a higher face probability for face 15. This high reactivity can be attributed to the fact that pheromone evaporation is initiated by data ants traversing the node and also occurs at the end of the time window, one second in this experiment. Thus, any changes in service and network states are detected almost instantaneously. The SoCCeR node then starts forwarding service requests to face 15 instead of face 13 (Fig. 5(f)). The effect of lower delays can be observed at  $t = 160$  when the delay toward face 8 becomes lower than the delay toward face 15, resulting in higher face probability for face 8. Finally, at  $t = 275$ s the service instance reachable on face 13 recovers (Fig. 5(f)) resulting in the router forwarding all requests thereon to face 13.

Finally, we analyze the impact of the time window size,  $\delta_t$ , on delay and routing overhead, i.e., the ratio of bandwidth occupied by the Ants and the total available bandwidth. Fig. 6 plots the 90th

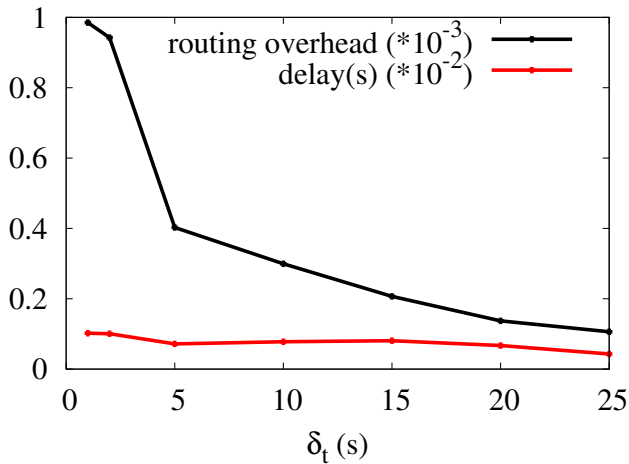
percentile of the delay distribution (scaled by a factor of  $10^{-2}$ ) and the routing overhead (scaled by a factor of  $10^{-3}$ ). The overall routing overhead is minuscule, e.g., lower than  $10^{-3}$  or 100 Kbps, even when  $\delta_t = 1$ s which provides extremely high reactivity to failure (Fig. 5). As  $\delta_t$  increases, the overhead quickly reduces, e.g., 40 Kbps when  $\delta_t = 5$ s; however, large values of  $\delta_t$  negatively affect SoCCeR reactivity to changes in network and service state. Similarly, the overall delay is low, few milliseconds, independent of the  $\delta_t$  value. This low overall delay indicates that the routing traffic does not contribute to any congestion in the network.

## 5. RELATED WORK

Researchers have proposed several paradigms for future network architectures, one of them being service-centric networking. Service-centric networks [7, 15] depart from the conventional view of connecting end-points to transfer data to a flexible architecture that deploys customized in-network services on-demand. A centralized controller with global network view gathers service state through regular queries.

Similarly, centralized load balancers adjust load among services running in a cloud [1-3, 5, 11]. The replicated service instances have explicit IP addresses, and therefore, a resolution of a service name results in a pool of IP addresses. The centralized load balancer receives requests from clients and distributes them to the appropriate service based on service state. Some of the problems with centralized schemes are: a single point of failure, high overhead due to periodic queries for network state, and latency in handling failure, service mobility, changes in service demand patterns.

Schoonderwoerd et al. [13] demonstrated the earliest application of the ant-colony optimization heuristic for load balancing in



**Figure 6: Routing overhead for different Interest Ant generation time windows.**

circuit-switched telecommunication networks. Ants deposit pheromone as a function of congestion in links along a path and the distance between nodes. Therefore, calls between nodes are setup as a function of the pheromone distributions in the links.

DONAR [14] is a recently proposed decentralized scheme for cloud-based services, in which a set of mapping nodes run a distributed server selection algorithm. While DONAR follows a decentralized approach and avoids the use of a central coordinator, it requires a back-end data store to coordinate the routing decisions of the mapping nodes.

SCAFFOLD [8] emphasizes services on end-systems focusing on flow-based anycast with service hosts that have topology dependent addresses. SCAFFOLD does not focus on routing per se; instead it focuses on randomized selection of multiple matching instances using a weighted proportional split method. Failures and service relocation are detected through in-band signaling updates. New services join or leave the network through explicit join, register, unregister, and leave messages sent to the SCAFFOLD routers.

STIR [10] is a stigmergy inspired routing scheme targeting content-centric delay tolerant networks. A utility metric at each (mobile) node defines the degree of interest of the node with respect to content and in effect, captures the spatio-temporal characteristics of the mobility of the node. Ant colony optimization is used to reinforce the utility metric leading to convergence to shortest paths between content and users in a delay tolerant network environment.

SoCCeR, in contrast, is a completely decentralized and distributed service routing and load balancing scheme. In /mboxSoCCeR each node actively gathers state information and makes independent decisions on service routes to the best instance. It is highly reactive to any changes in service or network state, and the reliance on names obviates the need for resolution.

## 6. CONCLUSIONS AND FUTURE WORK

This paper presented SoCCeR, a novel distributed scheme for solving the service routing and service selection problem. In order to demonstrate its functionality we implemented SoCCeR on top of CCNx, an open source CCN project. We used simulations to compare the performance of SoCCeR with that of a Vanilla CCN and a Random forwarding approach, respectively. Our results show that SoCCeR nodes efficiently select the service instance to forward a request to without any global coordination. This decision is based on path congestion as well as load at a service instance, allowing

the network as a whole to load balance effectively. In addition, we also demonstrated the responsiveness of SoCCeR to service failure, service recovery and path congestion.

As future work, we aim to further investigate the performance of SoCCeR for different parameter values. Another interesting problem we aim to tackle is the problem of node selection to instantiate a new service instance, taking into account service load, popularity of a service in certain network regions and other metrics like distance to clients. Although SoCCeR provides an efficient solution for stateless services, additional considerations are needed if subsequent service requests have to be forwarded to the same service instance.

## 7. ACKNOWLEDGEMENTS

The authors would like to thank Stefan Valentin, Torsten Braun, Tilman Wolf, and the anonymous reviewers for their helpful comments.

## 8. REFERENCES

- [1] Amazon cloudwatch service, <http://aws.amazon.com/cloudwatch>.
- [2] Amazon elastic compute cloud, <http://www.amazon.com/ec2>.
- [3] Amazon load balancer service, <http://aws.amazon.com/elasticloadbalancing>.
- [4] Project ccnx: Open source ccn implementation, <http://www.ccnx.org>.
- [5] Windows azure platform, <http://www.microsoft.com/azure>.
- [6] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.
- [7] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson. The silo architecture for services integration, control, and optimization for the future internet. In *Proc. of IEEE ICC*, pages 24–27, 2007.
- [8] M. J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordström, J. Rexford, and D. Shue. Service-centric networking with scaffold. Technical Report TR-885-10, Princeton University, 2010.
- [9] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Network Named Content. In *Proc. of ACM CoNEXT 2009*, Rome, Italy, December 2009.
- [10] A.-D. Nguyen, P. Sénac, and M. Diaz. STIR: STIGmergy Routing (STIR) for Content-Centric Delay-Tolerant Networks. In *LAWDN - Latin-American Workshop on Dynamic Networks*, 2010.
- [11] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. of IEEE/ACM CCGrid'09*, Washington, DC, USA, 2009.
- [12] G. Pallis and A. Vakali. Insight and perspectives for content delivery networks. *Commun. ACM*, 49:101–106, January 2006.
- [13] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks, 1996.
- [14] P. Wendell, J. Jiang, M. Freedman, and J. Rexford. Decentralized Server Selection for Cloud Services. In *Proc. of ACM SIGCOMM 2010*, New Delhi, India, August 2010.
- [15] T. Wolf. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of ICCCN*, Arlington, VA, Oct. 2006.
- [16] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM'96*, San Francisco, CA, Mar. 1996.