NextServe Framework: Supporting Services Over Content-Centric Networking

Dima Mansour, Torsten Braun, and Carlos Anastasiades

Communication and Distributed Systems, University of Bern
Neubruckstrasse 10, 3012 Bern, Switzerland
{mansour,braun,anastasi}@iam.unibe.ch
http://cds.unibe.ch

Abstract. The future Internet architecture aims to reformulate the way services and content objects are requested in a location-independent manner. Information-Centric Networking is a new network paradigm, which tries to achieve this goal by making content objects identified and requested by names instead of addresses.
In this paper, we extend the Information-Centric Networking architecture to support services to be requested and invoked by names. We present the *NextServe* framework, which is a service framework with a human-readable self-explanatory naming scheme. *NextServe* is inspired by the object-oriented programming paradigm and is applicable in real-world scenarios.

Keywords: Service-Centric Networking, Content-Centric Networking, Information-Centric Networking, Future Internet Architecture.

1 Introduction

The design of current Internet architecture relies on the fact that every node has an IP address. The sender's packet should contain the IP addresses of the source and the destination. The increasing use of the Internet, the new requirements by mobility of users and security, as well as the expanding content volume prompt researchers to think about new designs for the Internet architecture [1].

There are many questions about the next-generation Internet architecture. Some of them are related to the way content objects are requested, some are related to the optimal routing scheme for content object requests, and others investigate the capability to build a suitable architecture for the current use case scenarios of the Internet.

Information-Centric Networking (ICN) [2] proposes some answers for those questions by giving content objects names instead of addresses. There are many implementations of ICN like Content-Centric Networking (CCN) [2], Publish-Subscribe Internet Routing Paradigm (PSIRP) [3], and Data-Oriented Network Architecture (DONA) [4] . These projects differ in design and implementation, but agree on the concept that content is the first-class citizen in the network.

There is one main limitation with those projects. They all support static content only and there is no natural support for services. By taking a look at the current Internet applications and user needs, we can see that a high percentage of user requests is for services. Some are simple like "user sign up" services, and others are complex like financial transaction services. So, we believe that future ICN projects and architectures should support dynamic services as well as static content.

In this paper we extend an ICN architecture and naming scheme to support services. Our approach allows services to be requested and invoked by their names to reach the concept of Service-Centric Networking (SCN), where services are also first-class citizens of the network. Our service naming scheme is inspired by the object-oriented programming paradigm and takes into consideration simple and complex service characteristics. *NextServe* is the implementation of our approach to support services over ICN. Our implementation is based on the CCNx project [5], which is an implementation of the Content-Centric Network (CCN) protocol [6]. We explain the architecture and the naming scheme of the *NextServe* framework and discuss the advantages and the limitations of our approach.

The rest of the paper is organised as follows: In Section 2, we introduce the necessary technical information regarding the CCN protocol. In Section 3, we explain our approach and its architecture. We demonstrate a detailed example and motivate our design decisions. In Section 4, we discuss the naming schemes of previous projects in Service-Centric Networking and show the advantages of *NextServe* over those projects. Finally, we conclude this paper in Section 5 and discuss future work in Section 6.

2 Technical Background

Content-Centric Networking (CCN) deals with content objects as separate entities regardless of the hosts'IP addresses. The elements of the CCN architecture and the CCN communication model is shown in Figure 1. Content publishers publish their content objects by advertising them to content Routers. The CCN communication model relies on two types of packets rather than IP packets. The consumer sends an Interest packet containing the content name. The producer sends a Data packet containing the corresponding data. The content router processes the request using three tables:

1. The Content Store (CS) is a cache memory that stores the retrieved data mapped with the corresponding content name.
2. The Forwarding Information Base (FIB) is a table of outbound faces for Interests. The FIB table is a standard routing table used for Interest forwarding based on content names rather than IP addresses.
3. The Pending Interest Table (PIT) matches between the content name and all faces that are interested to reach the corresponding data. Then, the router can remember the outstanding Interests.

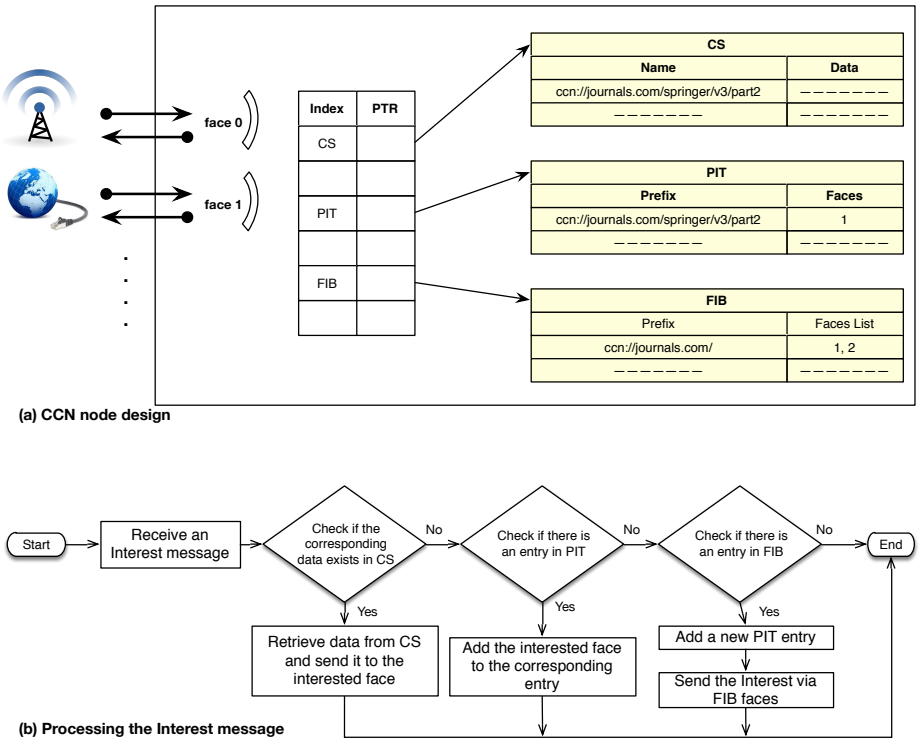

Fig. 1. A general overview of the CCN protocol.

When a client sends an Interest packet, the content router looks it up in the CS to retrieve the corresponding data directly. When there is no corresponding entry in CS, the router searches the PIT for an entry that has the same content name. If it finds one, it adds the interested interface to it. If it does not find a match, it adds a new entry with the content name and the corresponding interested interface. Finally, the router searches in the FIB. If there are matching prefixes in the FIB, it will forward the Interest to the corresponding faces. Otherwise (no matching prefixes in FIB), there is no way to reach the corresponding data.

The names in CCN are arranged in a hierarchical structure to facilitate the aggregation, management, and discovery of services. Each name consists of multiple components, which in turn can be any string of arbitrary length. These names have also information like versions and chunks of data. For instance, the name *ccn://Journals/springer/v3/part2* is to ask for the second part of the third version of the content *ccn://Journals/springer*.

3 The *NextServe* Framework

3.1 Naming Scheme

The naming scheme in *NextServe* is very similar to the method invocation style in modern programming languages like Java or C#. Service names follow the grammar in Figure 2. From those production rules, we can see that services have exactly the hierarchical names as content objects in CCN, but with the following additions:

- Services can accept parameters.
- Service parameters can be primitive values (string, integer, *etc.*), content objects, and other services.
- The returned result from one service can be a parameter to another service.This allows for service composition easily.
- The service parameters are contained between "*/(*" and "*/)*".
- The service parameters are separated using ",".
- When the parameters are primitive values, they are contained within double quotations.

To summarize, the service name is: */prefix1/prefix2/.../ServiceName/(param1, param2,.../)*. The parameters can be simple scalar values ("2", "3.14", "Hello World", *etc.*), content objects (*e.g.*,/unibe/iam/cds/schedule), or other services.

```
<CompleteName>   ::= / <ServicePrefix> <ServiceName> <Parameters>
<ServicePrefix>  ::= identifier/ | identifier/ <Component>
<Component>      ::= identifier/ | identifier/ <Component>
<ServiceName>    ::= identifier
<Parameters>     ::= /(<Params>/) | epsilon
<Params>         ::= epsilon | <ParamValue> | <ParamValue>,<Param>
<Param>          ::= <ParamValue> | <ParamValue>,<Param>
<ParamValue>     ::= <LocalParam> | <CompleteName>
<LocalParameter> ::= "<Value>"
<Value>          ::= text
```

Fig. 2. The grammar of the naming scheme in *NextServe*

3.2 Architecture

The layered architecture of our approach is shown in Figure 3. The topmost layer contains the services. These services contain the application-specific business logic. The middle layer contains the necessary components for publishing services, handling requests, responses, and service parameters, as well as invoking service implementations. The lowest layer contains the CCN core.

The "CCN Connector" component handles the communication with the CCN core. It manages the Interest and Data messages. The "Name Parser" component parses the service requests according to the grammar shown in Figure 2 to determine the service implementation and the parameter values. When a parameter is a content object or another published service, the "Parameter Retriever" fetches the corresponding content data or service reply through the "CCN Connector". All the aforementioned functionality is encapsulated and abstracted from the "Services" layer by the "Service Publisher" component, which is responsible for publishing and un-publishing services.

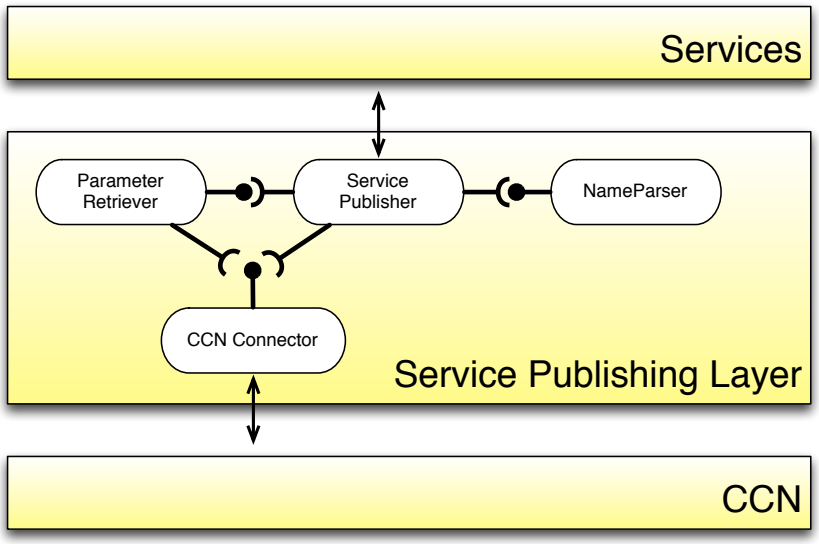

Fig. 3. The layered architecture of *NextServe*.

3.3 Concrete *NextServe* Use Case Scenario

To better explain the approach, we give a concrete example. This example is completely implemented using CCNx and *NextServe*. The setting of the test scenario is as follows (as in Figure 4):

- There are three content routers. Each one has a running "ccnd" daemon (the executable of CCNx).
- There are two service providers. Each one is running a "ccnd" daemon and *NextServe*.
- There is one content provider running a "ccnd" daemon.
- Each node in the setting is running on a separate virtual machine. All virtual machines are connected via a network as in Figure 4.

In our example, there is an encryption service called */scn/encrypt*. This service takes two parameters. The first one is the encryption password. The second parameter is the content object to encrypt. Then, the service encrypts the content object using the password. There is another service called */fileManager/zip*. This service takes a content object as a parameter and compresses it. A possible client might submit the request */scn/encrypt/("P@ssw0rd",/fileManager/zip /(/university/profile.pdf/)/)*. This request can be read exactly like a method invocation in Java or C#. This means that the client wants to compress the file */university/profile.pdf* and then encrypt it using the password "P@ssw0rd".

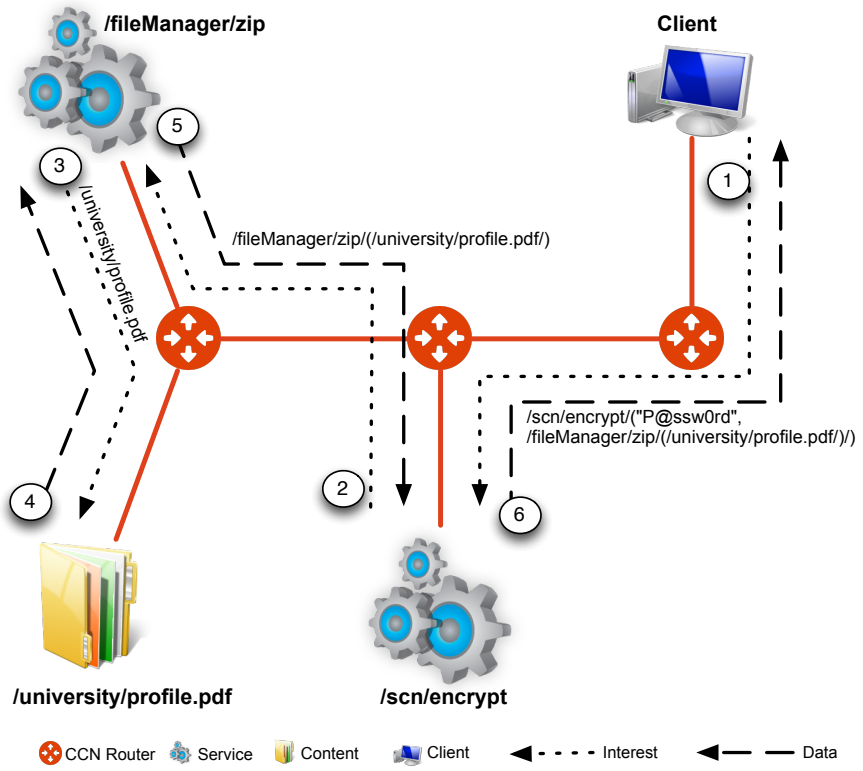

Fig. 4. The process of handling the request */scn/encrypt/("P@ssw0rd",/fileManager /zip/(/university/profile.pdf/)/)*

There are few points to notice here:

- The encryption service is published under the name */scn/encrypt*.
- The compression service is published under the name */fileManager/zip*.
- The profile content object is published under the name */university /profile.pdf/*.

Figure 4 shows the process of handling the service request using *NextServe*. When the CCN router receives a request for */scn/encrypt/("P@ssw0rd", /fileManager/zip/(/university/profile.pdf /)/)*, it performs name-based routing based on the longest prefix match. In this case it is */scn/encrypt*. The CCN router does not understand the components in the parameter part. The perception of the rest of the request is the responsibility of the "Service Publishing" layer in the encryption service node. When the request reaches the service encryption node, the service publisher component receives the request and uses the "Name Parser" to parse the service parameters. In this case, the service publisher extracts two parameters. The first one is the password "P@ssw0rd" and the second one is "*/fileManager/zip/(/university/profile.pdf/)*". Then the service publisher uses the "Parameter Retriever" component to retrieve the second parameter from the CCN network by sending a request for "*/fileManager/zip/(/university/profile.pdf/)*". Again, when the CCN router receives that request, it forwards it to the compression service node by maximum name component matching, which is in this case "*/fileManager/zip*". In the same way, the service publishing layer parses the request name and extracts the parameter for the compression service. This parameter in this case is "*/university/profile.pdf*". Then, the "Parameter Retriever" component places a request for that content from the CCN network. When the Data message for "*/university/profile.pdf*" is received, the "Service Publisher" invokes the "zip" service on that content object data. Then, it sends the result in a Data message to the CCN network, which routes it to the requester, *i.e.*, the encryption service node. After receiving the result, the "Service Publisher" component in the compression service node invokes the service "encrypt" using the password and the content parameters. Then it sends back the result in a Data message to the client via the CCN network.

3.4 Implementing and Publishing Services

Publishing services is easy since the only interface that has to be dealt with is the *Service Publisher*. The Service Publisher has a method called *publish* that takes two parameters: The first one is the name of the published service as a string (*e.g.*,/scn/encrypt). The second parameter is an object that has a method called *execute*, which takes a list of byte arrays, which represent the parameter values. When a request for a service is received, the "Service Publisher" fetches the parameter values from the request and transforms them to a list of byte arrays. Then it invokes the *execute* method on that ArrayList.

The service owner only needs to implement the parameter mappings to the desired types and then invokes whatever business logic that needs to be invoked. This mapping can also be automated through Java Reflection but it is not done yet.

3.5 Motivating our Design Decisions

We chose the layered architectural style because it is very appropriate for network applications and protocols. It also achieves the separation of concerns design principle allowing each layer to evolve separately without affecting other layers.

The decision of putting the responsibility of handling services, service names, and service parameters into the application or into the service provider is based on the discussion in [7]. The authors in [7] discuss three approaches to implement services over CCN: One of them is to implement services in the core of CCNx. In this case some modifications are needed in the existing infrastructure of the CCNx core. Another approach is to implement services at the publisher side. Then, all publishers have to be modified to provide the service to all clients. The third approach is to implement services as a separate application. In this case, there is no need to touch the CCNx core.

NextServe follows the third approach and uses the CCN infrastructure as it is without any modification. Our approach follows the CCN protocol guidelines regarding naming, which specifically states that *"CCNx content names are not interpreted in the operation of the CCNx protocol itself, just matched"* [8]. In this way, our approach allows for caching because the parameters are components in the Interest name. So, if two clients ask for the same service on the same parameters, the CCN routers will only fetch one and forward the answer to both clients. If a third client asks for the same service with the same parameters, the CCN router gets the result from its Content Store.

4 Related Work & Discussion

There are many projects that aim at supporting services over ICN. CCNxServ[9] is also built on top of CCN. It adopts the same hierarchical structure of the names as CCN. The naming scheme is *ContentName + ServiceName*, where *ContentName* is exactly as in CCN. *ServiceName* is the name of the service module that should be invoked on the requested content. CCNxServ assumes that services are implemented as separate Java JAR files. CCNxServ retrieves the service file (as a JAR file) and the content file, then it executes the service on the content. The CCNxServ architecture has three main components. The first component is the CCN network. It is responsible for handling Interest and Data messages. The second component is the *ServiceProxy*. It is responsible for interpreting the name in the Interest message to get the service name and the content name. ServiceProxy intercepts the Interest message and creates two Interest messages instead. The first one is for the service file and the second one is for the content file. After getting the files, ServiceProxy deploys and executes the service on a service execution framework called NetServ[10], which is the third component. NetServ is responsible for executing the service and returning the result to the ServiceProxy, which in turn, returns the result to the client.

NextServe has the following advantages over CCNxServ:

- *NextServe* allows any number of parameters.

- *NextServe* allows for parameters to be sent from the client itself.
- *NextServe* allows for service composition.
- *NextServe* does not make any assumption about how the service is implemented as long as it provides a compatible interface.

Named-Function Networking (NFN)[11] is built on top of CCN to support services. Service naming in NFN is inspired by the λ-expression language. For instance, the corresponding grammar for the λ-expression: f(g(data)) has the following CCN name: *[ccn:nfn | /name/of/data | /name/of/g | /name/of/f]*. This is a request for applying function "f" on the result of applying function "g" on "data". NFN is a promising approach but it still does not support local parameters as in *NextServe* and its naming scheme is not as user-friendly as the naming scheme in *NextServe* especially when the request is complicated.

Another important project in Service-Centric Networking is Serval[12]. Serval changes the TCP/IP protocol stack to provide special interfaces to deal with service allocation and connection. In Serval, every service has an ID, which consists of three parts: *Provider-Prefix + Provider-Specific + Self-Certification*. *Provider-Prefix* can be the company name (*e.g.*, Apple). *Provider-Specific* can be the service name (*e.g.*, iMessage). *Self-Certification* is the hash of the public key and the service name allowing the services to be self-authenticating without relying on a central certifying authority. Serval introduces a new kind of routers called "Service Routers", which combine the functionality of load balancers, proxies, and DNS. When a client asks for a certain service, the service routers are responsible for finding the best service replica to serve the request. Serval has many advantages in its architecture. It allows for load balancing, mobility, sessions, and fault tolerance.

NextServe differs from Serval in the following points:

- *NextServe* does not change the underlying TCP/IP protocol stack. This allows for easier adoption and integration into the current Internet architecture.
- *NextServe* supports caching naturally.
- Serval has been mainly developed to support data centers. *NextServe* can be adopted by any service provider or service client.

As mentioned in Section 3, *NextServe* does not change the implementation of CCN. Hence, any evaluation results of CCN can be applied on *NextServe*. It is shown in [6] that CCN performs better than TCP and also scales to numbers of requests to exponential magnitudes of nowadays needs. Similar results in [13] show that CCN introduces an overhead of 19% when compared with TCP/IP. But as the number of consumers increases, CCN outperforms TCP/IP and the download time in CCN is 25% less than it is in TCP/IP. In [14], it is stated that network topology has no effect on the efficiency of CCN but multi-path routing plays an important role in the performance of CCN. The main advantage of CCN is coming from the caching mechanism. *NextServe* inherently supports caching because of the naming scheme design. This leads to the fact that the evaluation results of CCN can be extended to cover also *NextServe*.

5 Conclusions

In this paper, we introduce the field of Service-Centric Networking (SCN) and the necessary background. Then we demonstrate *NextServe*, which is an SCN framework to support services over Content-Centric Networking (CCN). We show that *NextServe* overcomes most of the problems and shortcomings of previous projects by applying a naming scheme that is inspired by object-oriented languages. *NextServe* is implemented in a way that does not put any limitations on service implementations. *NextServe* does not change the underlying CCN network but rather implements services in the application layer on the service provider side.

6 Future Work

In CCN there might be redundant content retrievals if the router FIB table has no entry for that specific content object and uses broadcast to find the content object, or if the content object is not in the router Content Store (CS), and the router FIB table has many route entries for the content object. In those cases the Interest packet might reach many content replicas, all of them will respond with Content packets but only one will reach the client. This redundancy becomes very expensive when we deal with services. There are few attempts to overcome this issue [15]. In the future we are going to investigate how we can find an optimal or near-optimal routing solution in *NextServe*. Moreover, *NextServe* does not support sessions. But after solving the routing problem, session support comes for free because a client can connect to the best service replica from the beginning and keep sending the following requests to the same replica. We also plan to use Java Reflection to allow the automatic mapping between the parameters in the Interest packet and the actual parameters in a Java method. In this way, we decrease the amount of effort needed to publish any Java method as a service over *NextServe*. Another important direction in the future is to evaluate the efficiency and performance of *NextServe* in comparison with current service technologies like web services.

References

1. J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *Communications Magazine, IEEE*, vol. 49, pp. 26–36, July 2011.
2. V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, (New York, NY, USA), pp. 1–12, ACM, 2009.
3. N. Fotiou, D. Trossen, and G. Polyzos, "Illustrating a publish-subscribe internet architecture," *Telecommunication Systems*, vol. 51, no. 4, pp. 233–245, 2012.
4. T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 181–192, Aug. 2007.

5. "Ccnx project official website." `http://www.ccnx.org/`. Last Checked, December 06th, 2013.
6. V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, (New York, NY, USA), pp. 1–12, ACM, 2009.
7. E. Cheriki, "Design and implementation of a conversion service for content centric networking," Master's thesis, Institute of Computer Science and Applied Mathematics University of Bern, 2012.
8. "Ccn protocol overview." `http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html`. Last Checked, January 06th, 2014.
9. S. Srinivasan, A. Singh, D. Batni, J. Lee, H. Schulzrinne, V. Hilt, and G. Kunzmann, "Ccnxserv: Dynamic service scalability in information-centric networks," in *Communications (ICC), 2012 IEEE International Conference on*, pp. 2617–2622, 2012.
10. J. W. Lee, R. Francescangeli, W. Song, J. Janak, S. R. Srinivasan, M. S. Kester, S. A. Baset, E. Liu, H. G. Schulzrinne, V. Hilt, Z. Despotovic, and W. Kellerer, "Netserv framework design and implementation 1.0," technical report, Columbia University, http://academiccommons.columbia.edu/catalog/ac:135424, May 2011.
11. C. Tschudin and M. Sifalakis, "Named functions for media delivery orchestration," in *Packet Video Workshop (PV), 2013 20th International*, pp. 1–8, 2013.
12. E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman, "Serval: an end-host stack for service-centric networking," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), pp. 7–7, USENIX Association, 2012.
13. P. H. V. Guimaraes, L. H. G. Ferraz, J. V. Torres, D. M. Mattos, P. Murillo, F. Andres, L. Andreoni, E. Martin, I. D. Alvarenga, C. S. Rodrigues, *et al.*, "Experimenting content-centric networks in the future internet testbed environment," in *Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 1383–1387, IEEE, 2013.
14. D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, 2011.
15. S. Shanbhag, N. Schwan, I. Rimac, and M. Varvello, "Soccer: services over content-centric routing," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, (New York, NY, USA), pp. 62–67, ACM, 2011.