CCNxServ: Dynamic Service Scalability in Information-Centric Networks

Suman Srinivasan[†], Amandeep Singh[†], Dhruva Batni[†], Jae Woo Lee[‡],
Henning Schulzrinne[‡], Volker Hilt[⋆] and Gerald Kunzmann[◇]
[†‡] Columbia University, [⋆]Alcatel-Lucent Bell-Labs USA, [◇]DOCOMO Communications Laboratories Europe
[†]{srs2117,as3947,dlb2155}@columbia.edu, [‡]{jae,hgs}@cs.columbia.edu,
[⋆]volker.hilt@alcatel-lucent.com, [◇]kunzmann@docomolab-euro.com

***Abstract*—Content-centric networks promise to address content networking issues in a better way than today's host-based networking architecture. But content-centric networking does not inherently address the issue of services, particularly service scalability and mobility. We present our work on CCNxServ, a system that allows for dynamic service deployment and scalability in a content-centric networking implementation (CCNx) through an intuitive use of the content naming scheme. It thus extends the concept of content-centric networking towards services.**

I. Introduction

Content-centric networking (also known as Information-Centric Networking) (ICN) aims to address a key problem in computer networking: content constitutes a large portion of bandwidth and hence cost in computer networking. However networking today is host-based, and allows nodes to address each other for communication. However, requests are more often made for a specific piece of content rather than for a specific node. Content-centric networking projects, such as CCNx [1], XIA [2] and Nebula [3], aim to improve content networking by providing an entire network stack centered around content and handling content requests. Information-centric networking typically focuses on static content objects, but many content services do not deliver the same bits to every receiver, but transform and personalize them, examples including personalized advertisements.

We believe that a complete networking architecture will focus not only on content-centric networking, but will involve services as well. In this paper, we present our architecture and implementation of CCNxServ which allows for dynamic services and service scalability on top of the CCNx content-centric networking framework. We present how we use the CCNx naming scheme to add service functionality to the purely content-centric CCNx architecture, thereby leveraging the existing content-centric features of CCNx and allowing for service scalability and mobility in CCNx. We add composable media transformation services as an integral component of the CCNx framework.

In addition, running services over CCNx and similar content-centric networks will allow for services to be deployed and scale dynamically, as they can be distributed and duplicated dynamically similar to how content would be replicated in such networks. In contrast, today's host-based networks require precise information of the network topology as well as knowledge of node location in order to be able to deploy services to those locations. For example, the popular content delivery network Akamai [4] offers some value-added services on top of its content delivery network, such as its "Advertising Decision Solutions", which allows for companies to "seamlessly incorporate real-time anonymous Web browsing information with anonymous online purchasing data from advertisers' websites to present the most relevant ad" [5]. However, such services provided by Akamai and other large players are still restricted to predefined services and statically located data centers and do not allow for dynamic deployment of services in the network, while our services framework running on top of a content-centric network can provide such services and dynamic deployability. In addition, by being able to replicate services on demand, CCNxServ can scale dynamically and be replicated closer to hotspots where there is a lot of demand for similar services.

In addition, there is a class of services that run based on information from the end user at the edge of the network. For instance, when a user requests a YouTube page or a website from a content provider, it is not uncommon for an advertisement to show up alongside the video or webpage. However, these advertisement requests involve a separate request for the advertisement content which is still served from an origin server. This is a separate request, and as such, could be potentially blocked by the end user by ad blocking software. If instead the advertisement was served to the end user after being processed at an edge node that had a replica of the service, it would be one piece of content that would be customized to the user but indistinguishable from the original content request. (We are not trying to argue for or against advertisement on video or other pieces of content; this is just an example of how in-network service processing would differ dynamically from current service requests.)

We note that in our current service implementation, we are mainly concerned with media transformation services which provide transformative processing services on media without having to maintain state for each request or user. We acknowledge that there are a wide variety of services that require some maintenance of state, databases and personal information (such as social media and banking applications, to name just two

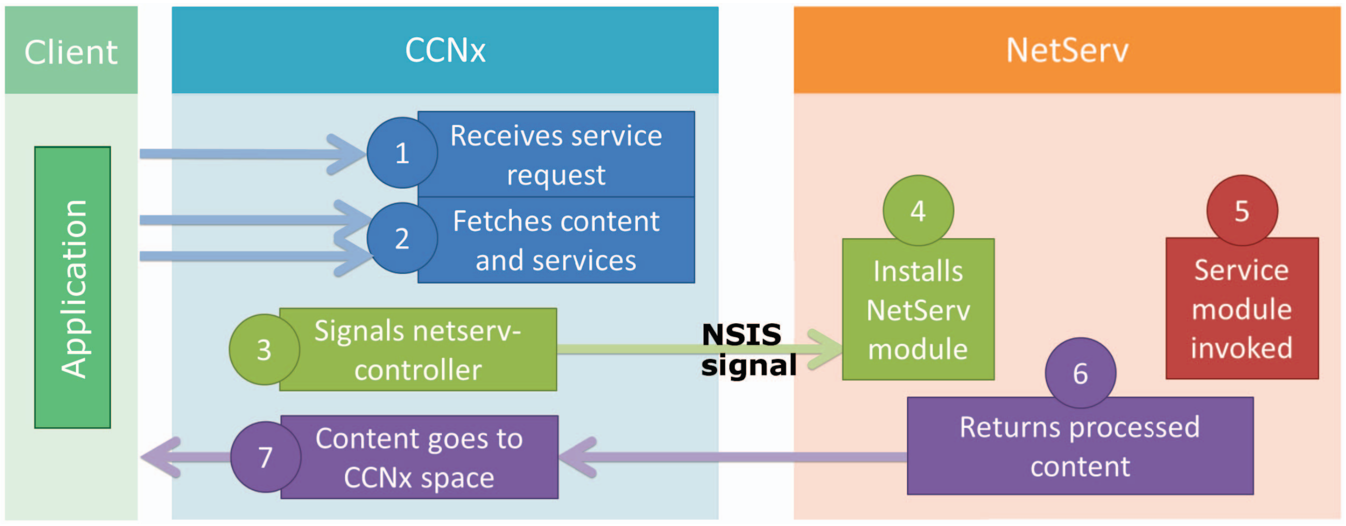

Client
CCNx
NetServ
Application
1 Receives service request
2 Fetches content and services
3 Signals netserv-controller
NSIS signal
4 Installs NetServ module
5 Service module invoked
6 Returns processed content
7 Content goes to CCNx space

Fig. 1. Our overall CCNxServ architecture. The CCNx controller (called CCNxServiceProxy) handles service requests and passes them onto a service framework (such as NetServ), where the content is processed and sent back into CCNx space.

such examples), and we believe that a complete treatment of such services in a content-centric network is beyond the scope of this paper.

II. Scaling Services Dynamically in CCNx

In our paper, we present our implementation of service-centric networking on top of CCNx [1]. We believe that CCNx and its current implementation (available as open-source on the CCNx website [6]) are a solid starting point for building a service networking architecture.

The architecture for our current implementation is shown in Figure 1. Our current CCNxServ implementation allows for a single service to be invoked alongside the content name (our naming scheme is *contentname+servicename*). When such a name is seen by a content router, the content router is able to parse the name into the content name and the corresponding service that is to be invoked on it. If it does not yet have a copy of the requested content, it retrieves the file via CCNx. Also, if it does not yet have a copy of the service module corresponding to the requested service, it fetches the corresponding service module via CCNx. Then, the content router invokes the service on the content and, after the processing is complete, it serves the processed content to the requesting client. In addition, it places the processed content back into the CCNx namespace so that future requests for this combination of *contentname+servicename* will directly be able to fetch the processed content.

Today's data center infrastructures usually require data to be moved to central locations where they are processed and transformed. CCNxServ allows for services to be moved right to where the data are situated, thereby enabling service mobility to locations where they are most needed, and allowing for transfer of lightweight executable modules rather than requiring large data sets to be moved around.

Note that in CCNx, we can also define such services implicitly in the content request: because CCNx allows for the concept of ”prefix matching,” any content router that implements our architecture could add a service name to the end of the name. Our current implementation will work in both centralized and distributed infrastructures that are supported by content-centric networking.

III. Architecture

For our current implementation, we focused on creating and running three exemplary services on top of the framework we describe below. The ”weather” service takes a video file as input and generates output with the latest weather information from www.weather.gov as an overlay on top of the video. The ”ads” service inserts a random video advertisement (from a given list of ad files in a certain directory) in the middle of the video. The ”news” service generates an output with the latest newsfeed (from BBC) as a marquee on top of the video.

An application running on a client device or a network node makes a request for a content name and a service it wishes to invoke on the content (e.g. *ccnx://video.mp4+ad*). The request is converted to a CCNx interest packet and forwarded to CCNx. The request is intercepted by any of the nodes that operate as a Content Router (1), and if the (processed) content ”video.mp4+ad” does not exist, the Content Router looks for a service corresponding to the service name. If it does not find a service in its local cache, it fetches it by issuing a CCNx interest packet for that service module (2). The same is true for the content; if the file (video.mp4) is not in the local cache, it is fetched from the content-centric network (2). Once both content and the service module are located and downloaded on the Content Router, the service module is installed (4) and executed on the content (5), thus producing a processed version of the content, which is returned to the client (6). In addition, the processed content is put back into the CCNx namespace for future requests (7).

In order to demonstrate that we are able to provide ”plugin” functionality for a fully robust service platform, we integrated the CCNx services implementation with the NetServ service virtualization framework [7]. NetServ was used because it is a robust service virtualization architecture for routers that allows custom service modules to process packets in the network core. It uses a general purpose open-source operating system as the forwarding engine, and layers a dynamic module system on top of it where new functions can be added. With NetServ, an edge router can become a platform for publishers' content and services, allowing content publishers to dynamically deploy within NetServ their services on these edge routers.

We created a CCNx controller that signals the NetServ service stack and runs service functionality through NetServ and its OSGi stack [8], while still using the content networking functionality provided by CCNx. The NSIS signalling protocols [9] [10] help to instantiate modules. The NSIS protocol is used for the routing and transport of per-flow signaling messages along the path taken by that flow through the network. The CCNx controller complements the IP-based signaling controllers with CCNx-based signaling.

In addition, as a result of our network-based signaling, one CCNx controller will be able to signal and control multiple NetServ nodes. This mode of operation could be useful in large data centers, where some of the nodes could be CCNx-enabled and others only IP-enabled, with some running a service framework and others a content-networking framework. In addition, such a topology would allow for services and content transformations to be scaled and moved to appropriate nodes depending on load factors.

IV. Implementation

We implemented our solution on top of the open-source implementation of CCNx provided by PARC [6]. The reference implementation provides the core CCNx protocol stack implementation along with a few sample applications and utilities. We use some of the existing utilities in the CCNx implementation, and have modified and implemented our own functionality on top of it.

One of the utilities that we use is the `ccnfileproxy`, a proxy for the file system that makes files on the local file system available over CCNx. It takes a directory from which to serve files, which it treats as the root of its content tree, and an optional CCNx URI to serve as the prefix for that file content as represented in CCNx. For example, if there is a directory /foo in the file system and the CCNx URI is defined as ccnx://testprefix, ccnfileproxy is called with the arguments *-/foo ccnx://testprefix*, and a request for *ccnx://testprefix/file.txt* would return `file.txt`.

For our implementation, we modified the ccnfileproxy as follows: we intercept the interest packet and scan the content name to see if any service names are included. If no service name is included, we allow the ccnfileproxy to continue its normal mode of execution. If there is a service name specified, we map it to an internal service module file name, issue another interest for the corresponding service file (a JAR file in our implementation) using `ccngetfile`, and dynamically load it. Then we find the appropriate class in the JAR file, and

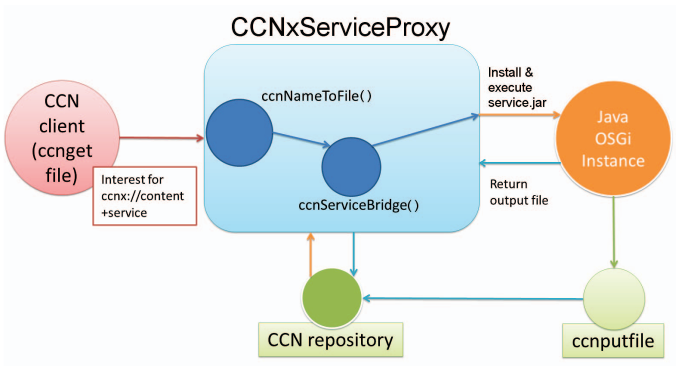


Fig. 2. The architecture of our CCNxServiceProxy implementation and how it interacts with the various CCNx utilities.

call the appropriate service method in the class on the content file. We call this modified proxy CCNxServiceProxy.

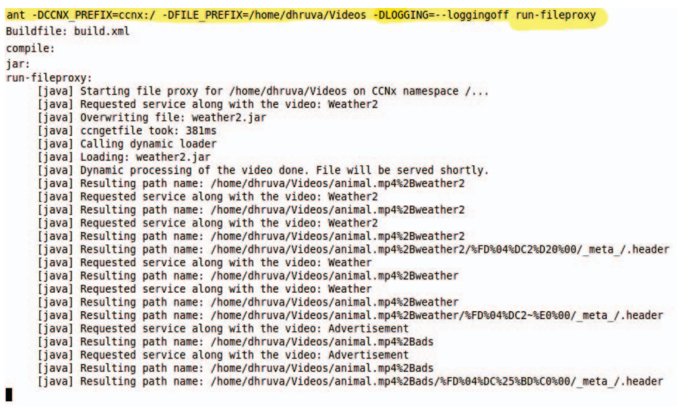
The pseudocode for our implementation (in pseudo-Java) is shown below:

```
ccnName = "ccnx://content+service";
array(service, file) = parse(ccnName);
bundleFile = download("ccnx://service"
   + ".jar");
content = download("ccnx://content/");
controller = intializeOSGi();
serviceBundle = controller.installBundle(
   bundleFile);
processedFile = serviceBundle.execute(
   content);
putFileIntoCCNx(processedFile);
```

To add a new service to our implementation, a new JAR file with the service has to be created. The most important file in the JAR file is the service class, which has to implement our *CCNxService* interface for the service implementation. The *execute(Object param)* method has to be overridden in this service class to provide service functionality specific to the class. When creating an OSGi bundle, the CCN-Service attribute is used in the manifest file of the JAR, and this points to the class which implements the *CCNxService* interface. An Activator class (for OSGi) can be included in the service bundle as well. All the service related class files are packaged into a `service.jar` file. The JAR files are then loaded into CCNx namespace through ccnxfileproxy or a similar utility.

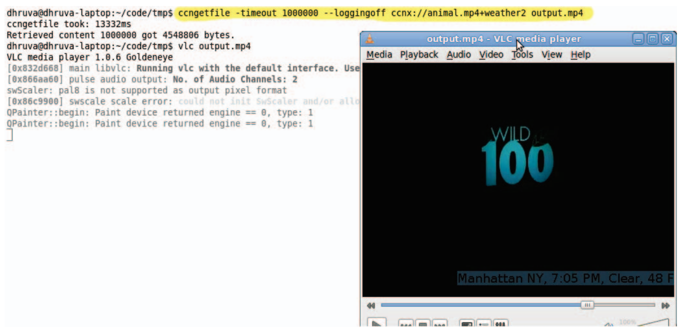
The overall architecture of our CCNxServiceProxy is shown in Figure 2. When the CCNx interest reaches a CCNxServiceProxy, we invoke the *ccnNameToFile()* method on the interest object. This function call translates the CCNx content name into the corresponding file name while parsing it. If any service is found in the content name, the *ccnServiceBridge()* method is invoked. This method handles the loading of the service module, invoking it on the content, and returning the processed content into the CCNx space.

When the *ccnServiceBridge* method is invoked, we check whether the output file, of the form



```
ant -DCCNX_PREFIX=ccnx:/ -DFILE_PREFIX=/home/dhruva/Videos -DLOGGING=--loggingoff run-fileproxy
Buildfile: build.xml
compile:
jar:
run-fileproxy:
     [java] Starting file proxy for /home/dhruva/Videos on CCNx namespace /...
     [java] Requested service along with the video: Weather2
     [java] Overwriting file: weather2.jar
     [java] ccngetfile took: 381ms
     [java] Calling dynamic loader
     [java] Loading: weather2.jar
     [java] Dynamic processing of the video done. File will be served shortly.
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2
     [java] Requested service along with the video: Weather2
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2
     [java] Requested service along with the video: Weather2
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2/%FD%04%DC2%D20%00/_meta_/.header
     [java] Requested service along with the video: Weather
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather
     [java] Requested service along with the video: Weather
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather/%FD%04%DC2~%E0%00/_meta_/.header
     [java] Requested service along with the video: Advertisement
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bads
     [java] Requested service along with the video: Advertisement
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bads
     [java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bads/%FD%04%DC%25%BD%C0%00/_meta_/.header
```

Fig. 3. A screenshot showing the processing of CCNx content after the Content Router interprets the content request.



```
dhruva@dhruva-laptop:~/code/tmp$ ccngetfile -timeout 1000000 --loggingoff ccnx://animal.mp4+weather2 output.mp4
ccngetfile took: 13332ms
Retrieved content 1000000 got 4548806 bytes.
dhruva@dhruva-laptop:~/code/tmp$ vlc output.mp4
VLC media player 1.0.6 Goldeneye
[0x832d668] main libvlc: Running vlc with the default interface. Use
[0x866aa60] pulse audio output: No. of Audio Channels: 2
swScaler: pal8 is not supported as output pixel format
[0x86c9900] swscale scale error: could not init SwScaler and/or allo
QPainter::begin: Paint device returned engine == 0, type: 1
QPainter::begin: Paint device returned engine == 0, type: 1
```

output.mp4 - VLC media player
Media Playback Audio Video Tools View Help
WILD 100
Manhattan NY, 7:05 PM, Clear, 48 F

Fig. 4. The transformed content being played in VLC player. There is an overlay containing weather information at the bottom right of the video. This corresponds to the invoked "weather" service.

`CONTENT_NAME%2B%SERVICE_NAME`, is present in the file repository. If the file is present, we check for any further service invocations. If the file is not present, we check whether the OSGi bundle corresponding to the service name is already installed. If the bundle is installed, we invoke *executeModule()* method on the OSGi controller with the bundleID and content name as the parameters. The *executeModule* call returns the file name of the processed file which is used as the input content for the next service in the chain. This way we can execute multiple services on a single content through chaining, for, e.g, *ccnx://content+service1+service2*; this call will result in two service invocations, service1 on "content" and service2 on the "content+service1" content file.

When the *executeModule* method is called, we load the CCN Service class using the bundle header, `CCN-Service`. After creating an instance of the CCNxService type class, we can invoke the execute method of CCNxService interface directly on this class instance. The invocation returns the file name of the processed file.

Videos are requested as shown below:

```
ccngetfile -timeout <value> --loggingoff
  ccnx://video.mp4+weather <output file>
```

Figures 3 and 4 show screenshots from the CCNxServ testbed. Figure 3 shows the processing of CCNx content after the Content Router interprets the above content request for the weather service invoked on a video file (*ccnx://animal.mp4+weather2*). After intercepting the interest packet, the content name is scanned and the service name "weather2" is detected. An interest for the corresponding service file *weather2.jar* is issued using ccngetfile. After successfully downloading the JAR file, the service is dynamically loaded and the content is processed. Finally, the file is returned into the CCNx space and to the requesting user. Figure 4 shows the processed content being played at the client device. The invoked services added an overlay containing the local weather information at the bottom right of the video.

V. Challenges and Incentives

In this section, we present the economic incentives for the CCNxServ work as well as the technical challanges that we faced while implementing it.

A. Economic Incentives

Video and content consumption on the Internet are rapidly growing and require enormous amounts of bandwidth. At the network core, the increasing consumption of multimedia content on the existing wired networks and in mobile systems is putting a strain on the network core. Cisco Systems estimates that video will constitute 90% of all Internet traffic by 2013 [11]. Content distribution networks (CDNs) are becoming more and more popular as a means of efficiently distributing multimedia content to end-users on the Internet. Pallis and Vakali [12] show that CDNs can reduce "traffic jams" for web traffic, since data is closer to user and there is no need to traverse all of the congested pipes and peering points. Content-centric networks (CCN) take the CDN argument and implementation further with the concept that content, not hosts, are the cornerstone of the Internet today.

CCN proponents [1] have argued that CCNs can decrease the cost of content delivery on the Internet. Park et al [13] argue that a backbone network provider could see as much as 30 percent traffic reduction using content-centric networks. In this light, the CCNxServ work that we are doing will enable a new class of services to run on top of the emerging CCNx architecture at reduced cost. In addition, as mentioned in the Introduction, CCNxServ enables lightweight executables to be migrated to locations of "big data", achieving service mobility to cover large datasets, further reducing networking costs.

B. Technical Challenges

We will briefly describe some of the technical challenges that we faced while building CCNxServ.

The first challenge was making CCNxServ work on a completely content-centric network infrastructure. Because the entire foundation of CCN networks rests on the concept of content, CCNxServ needs to share services by expressing them as a form of content, and at the same time, use the naming mechanism to indicate their use as executable services and not static content. This can be thought of as introducing "active networking" into a content-centric network. We believe our mechanism of addressing this solves the services problem while constraining it to work in a content-centric network.

The second challenge was integrating systems that combined the best features of content-centric networking and the traditional networking model, particularly for services. We chose to integrate our CCNxServ implementation with NetServ [7]. We were helped by the fact that while NetServ was initially built for IP networks and for host-based communication, its core service modularity allowed its networking layer to be reworked to support content-centric networking. Through refactoring CCNxServ, we were able to expose its core functionality as a controller to NetServ, and thus leverage NetServ's service APIs while being able to use CCNxServ's naming and service delivery mechanism on top of a pure content-centric network. However, this may prove a challenge with frameworks and libraries that have been hard-coded to the IP networking stack, such as OpenFlow [14] and others.

VI. Related Work

While there has been a lot of work done in the field of information-centric networking (ICN) the field of service-centric networking, especially as applied on ICNs, is fairly new.

SCAFFOLD [15] allows multiple service instances represented by one common name, which the specific service represented through serviceIDs being selected through anycast routing through service routers. MILNGENI (million-node GENI) [16] allows for services to be deployed and run on top of many end-systems that are connected via the experimental GENI testbed. But these projects operate on top of the IP layer and require host-to-host communication. Hence, their APIs and middleware are built for host-based networking.

Service-centric networking (SCN) [17] aims to supersede CCNx and thus involves building a superset of CCNx. In contrast, we are building a service platform on top of only CCNx, providing a service stack on top of CCNx, and we have been able to implement a fully service-oriented networking architecture on top of a pure ICN stack. We believe this demonstrates that it is possible to build services on ICN networks without requiring a superset of the features offered in ICN implementations. SoCCeR [18] is a related paper that builds on CCNx, and it works as a control layer to manipulate the underlying Forwarding Information Base (FIB), thereby performing distributed best-service selection using an ant colony optimization (ACO) approach. In contrast to our work, it requires modifying the CCNx interest and data packets to enable the control layer to work. In addition, our work deals with service deployment and dynamic scaling, and thus relates to the problem of effective service placement, which would complement SoCCeR's best service selection algorithm.

Another area that is somewhat related to our CCNx services work is in the field of data migration to data centers closer to the services. This work deals with large data sets and data centers, including tackling design issues to address data migration and latency. Tiwana et al [19] propose exposing the network location of data to the service, so that the service is able to optimize processing of data based on location. Volley [20] attempts to automate application data placement across data centers efficiently. PADS [21] provides a data plane mechanism for transmitting data and maintaining consistency in large distributed applications and data centers. Time-shifted TV [22] uses CCNx to improve localized and cooperative caching using content routers. It takes advantage of proximity features in CCNx to do cooperative caching of content. However, all of these approaches only deal with data latency and moving data across locations to bring them closer to the application. Our approach, in contrast, is about moving services themselves to where the data is located, and thus being able to move small executable modules to places with large data, allowing the data to be processing more efficiently.

Research on service placement is also related to our work. Service placement algorithms dynamically try to find the optimal number and locations of service instances given a certain service demand and network topology. A good overview of related work in the area of service placement is given in [23]. In contrast to pure caching solutions, service placement tries to increase the performance of a service based on application-specific quality metrics, while at the same time minimizing the overall network load and service costs. Therefore, services are replicated in the network and the service locations are dynamically adapted to the changing network conditions and service demands.

An architecture supporting a dynamic, distributed service provisioning for mobile users is described in [24]. It supports both optimized service placement within an operators network and placement of service components in a foreign network. This leads to both more efficient resource usage and better quality for the users.

VII. Future Work

While our current implementation handles media transformation service functionality effectively, we have to provide a more advanced architecture to handle more advanced service functionality such as database heavy applications and social media. For these services, we need to consider that they use state and database information since those need state information to be moved along with the service. In some cases, we may need to map a central database and distribute it along with the service, and allow for distributed transactions on the data and at the same time maintain data consistency across the network.

We can currently run a certain function or module on a piece of content, thus processing it, and also chain these events so that multiple services can be applied to content through a single request. However, we plan to extend our implementation such that more complex requests are supported, which require multiple services being processed on multiple content files. This could be achieved by adding a configuration file (e.g. an XML file) to the CCNx request. Then, for example, the weather service introduced above could be parameterized by the application to, e.g., overlay the weather information at different corners of the video or with a different overlay size.

At the moment, a new JAR file would have to be implemented in order to show the overlay at the top left corner of the video.

Moreover, there might be some services that may need to do more than execute certain functions on content. Instead they may want to control every step of the processing, and apply certain transformations based on certain conditions. Some services might require event-based processing and might need to register event handlers for service functionality. It would thus be useful to have a feedback or event-based mechanism in our implementation, so that an event or alert is triggered and sent out to all objects listening on an event.

VIII. Conclusion

We believe that services are central to network operations. In our work, we aimed to show that it is possible to build service functionality including dynamic service invocation, scalability, and mobility on top of content-centric networking, thus extending its features towards a service-centric network. We have a working implementation of scalable, dynamic service architecture implemented on top of the CCNx protocol stack. Our implementation enables dynamic invocation of services and true service mobility and scaling in a purely content-centric network based on need. By exposing services in a content-centric networking framework and the intuitive use of CCNx's content naming scheme, we are able to provide true service functionality in a future Internet platform that is centralized on both content and services.

IX. Acknowledgment

This work was supported by the National Science Foundation (NSF) under Grant 08-31912 and DOCOMO Communications Laboratories Europe GmbH.

References

[1] 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*. New York, NY, USA: ACM, December 2009.

[2] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machado, W. Wu, A. Akella, D. Andersen, J. Byers, S. Seshan, and P. Steenkist, "XIA: An Architecture for an Evolvable and Trustworthy Internet," Carnegie Mellon University, Tech. Rep., January 2011.

[3] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. Freedman, A. Haeberlen, Z. Ives, A. Krishnamurthy, W. Lehr, B. T. Loo, D. Maziéres, A. Nicolosi, J. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon, and C. Yoo, "NEBULA - A Future Internet That Supports Trustworthy Cloud Computing," http://nebula.cis.upenn.edu/NEBULA-WP.pdf, University of Pennsylvania, Tech. Rep., 2010.

[4] "Akamai," http://www.akamai.com/.

[5] "Akamai introduces advertising decision solutions," http://www.akamai.com/html/about/press/releases/2008/press_102108.html, October 2008.

[6] "Project CCNx," http://www.ccnx.org/.

[7] J. W. Lee, R. Francescangeli, W. Song, J. Janak, S. Srinivasan, M. Kester, S. A. Baset, E. Liu, H. Schulzrinne, V. Hilt, Z. Despotovic, and W. Kellerer, "NetServ Framework Design and Implementation 1.0," http://www.cs.columbia.edu/ jae/papers/netserv-tech-report-1.0.pdf, Columbia University, Tech. Rep., May 2011.

[8] "OSGi," http://www.osgi.org/About/Technology.

[9] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, "Next Steps in Signaling (NSIS): Framework," 2005. [Online]. Available: http://tools.ietf.org/html/rfc4080

[10] H. Schulzrinne and R. E. Hancock, "GIST: General internet signalling transport," Internet Engineering Task Force, RFC 5971, Oct. 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5971.txt

[11] J. Markoff, "Scientists Strive to Map the Shape-Shifting Net," http://www.nytimes.com/2010/03/02/science/02topo.html.

[12] G. Pallis and A. Vakali, "Insight and Perspectives for Content Delivery Networks," in *Communications of the ACM*, January 2006.

[13] C. Park, Y. Seo, K. youl Park, and Y. Lee, "The concept and realization of context-based content delivery of ngson," in *IEEE Communications Magazine*, 2012 January.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, March 2008.

[15] M. J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordstrom, J. Rexford, and D. Shue, "Service-Centric Networking with SCAFFOLD," Princeton University, Tech. Rep., September 2010.

[16] "A Prototype of a Million Node GENI," http://groups.geni.net/geni/wiki/MillionNodeGENI.

[17] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-centric networking," in *2011 IEEE International Conference on Communications Workshops (ICC)*, June 2011.

[18] S. Shanbhag, N. Schwa, I. Rimac, and M. Varvello, "SoCCeR: Services over Content-Centric Routing," *ACM Workshop on Information-Centric Networking (ICN)*, August 2011.

[19] B. Tiwana, M. Balakrishnan, M. K. Aguilera, H. Ballani, and Z. M. Mao, "Location, location, location!: modeling data proximity in the cloud," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, October 2010.

[20] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," in *Proceedings of the 7th USENIX symposium on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, April 2010.

[21] N. Belaramani, J. Zheng, A. Nayate, R. Soulé, M. Dahlin, and R. Grimm, "PADS: a policy architecture for distributed storage systems," in *Proceedings of the 6th USENIX symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, April 2009.

[22] Z. Li and G. Simon, "Time-Shifted TV in Content Centric Networks: The Case for Cooperative In-Network Caching," in *2011 IEEE International Conference on Communications (ICC)*, June 2011.

[23] G. Wittenburg, "Service Placement in Ad Hoc Networks," Ph.D. dissertation, Department of Mathematics and Computer Science, Freie Universität Berlin, October 2010.

[24] H. Lundqvist, Z. Despotovic, G. Kunzmann, J. Frtunikj, and W. Kellerer, "Service Program Mobility," in *Proceedings of IEEE Globecom 2011, Mobile Computing and Emerging Communication Networks Workshop (MCECN)*, December 2011.