ActiveCDN: Cloud Computing Meets Content Delivery Networks

Suman Srinivasan, Jae Woo Lee, Dhruva Batni and Henning Schulzrinne,
Columbia University, New York, NY
{sumans,jae,hgs}@cs.columbia.edu, dhruva.batni@gmail.com

***Abstract*—Content delivery networks play a crucial role in today's Internet. They serve a large portion of the multimedia on the Internet and solve problems of scalability and indirectly network congestion (at a price). However, most content delivery networks rely on a statically deployed configuration of nodes and network topology that makes it hard to grow and scale dynamically. We present ActiveCDN, a novel CDN architecture that allows a content publisher to dynamically scale their content delivery services using network virtualization and cloud computing techniques.**

I. Introduction

Content delivery networks (CDNs) have proved to be crucial for the Internet to scale with the current boom of multimedia-rich content consumption [1]. CDNs allow for content to be distributed at nodes distributed around the world, and played back from nodes that are closest to the user requesting content. This allows CDNs to serve content much faster to the end user, and at the same time, alleviate network load by pushing content closer to the edges. Today's CDNs, however, are statically deployed and centralized redirection mechanisms which impede dynamic deployability and result in performance penalties.

Despite this dramatic growth in CDN technology and use, most CDN operators and networks still rely on heavily static deployments and pre-configured network topologies to operate and serve content. This results in sometimes dramatic over- or under-provisioning, and can result in too many CDN nodes in an area that sees few request for content, or too few CDN nodes in an area that has a lot of demand for content. When such scenarios occur, it is hard for CDN service operators to dynamically migrate and instantiate new nodes, resulting in a lack of quick responsiveness to user demand and viral content.

We present ActiveCDN, a novel solution for this problem that utilizes cloud computing and network virtualization techniques to enhance CDN instantiation. ActiveCDN enables content publishers to dynamically deploy and instantiate CDN modules on participating nodes (such as core and edge routers with programming functionality) based on user requests, and to position and balance the location of these CDN nodes on an as-needed basis. Content publishers can thus choose to serve the content directly themselves, or instantiate new CDN nodes at locations closest to the users as more and more requests come in. In addition, the CDN modules can be set up to "time out" after a certain period of time if no new requests are seen at an ActiveCDN node, thus freeing up computing and space resources on that node.

We describe our motivation in more detail in section II. In sections III and IV, we describe our current implementations, one of which was highlighted and chosen as an alpha project for the National Science Foundation's (NSF) GENI next-generation networking project [2]. In sections VII and VIII, we present our future work and conclusion.

II. Motivation

The Internet has seen an explosive growth in traffic carrying multimedia content in recent years. Today, content providers either host their high-bandwidth multimedia content themselves, or more commonly host their content on content delivery network (CDN) providers such as Akamai and Limelight Networks which then deliver the content. When content is hosted on a CDN, a user request is usually redirected to a server closer to the user. Application-specific mechanisms are used for this, such as domain name resolution using DNS [3] or request redirection using HTTP [4]. CDNs are statically deployed and scaling them to adapt to sudden changes, such as unexpected flash crowds, is currently difficult to do without manual intervention and advanced over-provisioning. An example of such a flash crowd event was the inauguration of President Barack Obama in January 2009, which "generated massive Web traffic", leading to site slowdowns. [5]

CDN technology results in reduced traffic for the provider and the network. Pallis and Vakali [6] show that CDNs can bypass "traffic jams on the Web, since data is closer to user and there is no need to traverse all of the congested pipes and peering points." Pallis states that CDN costs are high (the paper cites costs from 2004), but CDN pricing has decreased recently. For example, Amazon's CDN services, such as Amazon S3 [7] and Amazon CloudFront [8], have been estimated to reduce hosting costs for low-bandwidth sites up to 75% [9]. Hosangar et al [10] present an analysis of how as CPU, infrastructure and traffic costs reduce, the pricing of CDNs will reduce further over time.

While cloud computing is delivering on the promise of elastic and flexible compute cycles in the cloud, we have yet to see it fulfill its promise and potential in the CDN space. While Amazon's S3 and CloudFront services allow for CDN functionality, it does not allow for dynamic deployment of nodes, and also relies on a highly static and pre-configured network topology.

We can see there is a need for being able to dynamically instantiate and deploy nodes at required places in the network.

We believe that ActiveCDN solves this problem.

In the following two sections (sections III and IV), we describe our current implementations of ActiveCDN, one of which processes and handles content on the edge node, and one of which processes metadata at the edge node while defering content processing to the client itself.

III. ActiveCDN on the Edge Node

ActiveCDN enables a next-generation content-delivery mechanism using CDNs, and also allows for "pop-up content store" nodes that appear on an as-needed basis on the Internet without having to be pre-deployed. Such a CDN architecture would also be able to serve content more efficiently in disconnected, opportunistic networks. For instance, consider a highly popular video that is in high demand at a certain location. ActiveCDN can automatically pop-up a CDN content store at nodes on some of the routes that the content traverses as it is delivered to the end users, and that node is able to cache the video on its local storage and serve the video directly to the end user.

Our current implementation of ActiveCDN operates on top of a network virtualization and cloud computing framework called NetServ [11] [12], which allows for participating nodes to dynamically instantiate and run modules signed by content publishers. ActiveCDN runs on top of NetServ's service virtualization framework.

Figure 1 shows how ActiveCDN works in a simple scenario where users are downloading a video file from a content provider. The content provider develops a ActiveCDN NetServ module and makes it available for download. The content provider's server machines respond to the content requests as usual.

When a content provider sees a request, it can - based on a number of factors such as content popularity - choose to use NetServ's signaling capabilities to install a module on a particular NetServ node along the path to the end user or users.

When a server notices the rate of requests from a network location above a certain threshold, the server initiates on-path signaling to tell the NetServ nodes in the path that they should download and install the provider's ActiveCDN module. The signaling message contains the URL and other information about the module, so that the NetServ nodes can determine if the module is compatible with the node's policy and capability, and where to download it from.

Once the module is installed and verified, the content provider can signal the ActiveCDN module to process the video if necessary, and redirect requests for content to the NetServ node that now holds the cached and processed video. After a NetServ node successfully downloads the module, it signals back the provider's server to register itself as one of the caching nodes. Now, the content requests originating from the vicinity of that caching node can be redirected to the node. The caching node will then fetch the requested content, sending it to the user as it is being fetched, and cache it for future requests.

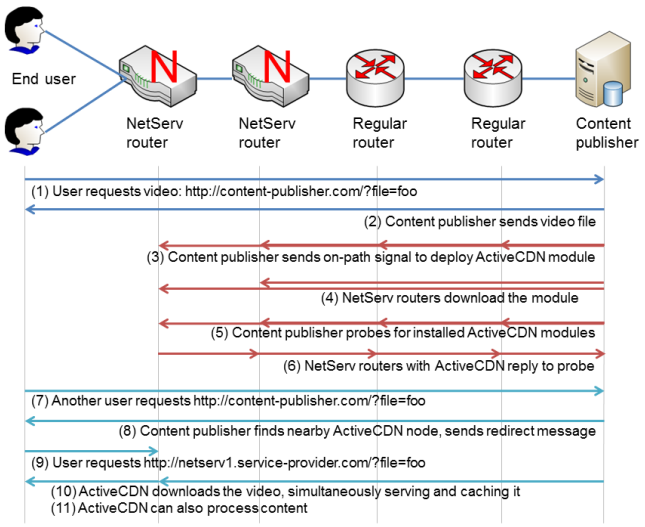

Fig. 1. How on-demand content caching using ActiveCDN works.

The ActiveCDN module functionality was implemented using Java, Java Servlets API and OSGi, and a Java library called Xuggler [13] was used for video processing functionality. The ActiveCDN module itself processes the video and adds a watermark video to the original video, and also serves the processed video to the end user. (In our NSF GENI demos, we added a watermark and local weather information as a text overlay on top of the video.)

In our current implementation, the content server keeps track of all the NetServ nodes that it has instantiated in a database, and upon getting a request from a client, calculates the closest NetServ node based on Euclidian distance between the two nodes (based on IP address and geo-location). The content server then redirects the request to the suitable NetServ node, and if no such NetServ node exists, it triggers an on-path installation of an ActiveCDN module that will install ActiveCDN on a NetServ node closest to the user.

The ActiveCDN module caches the content based on the request and serves the content, but also performs background processing on the cached content. The processing is determined by the functionality programmed by the content provider, and could include anything from ad insertion to news and weather overlay on top of the video. Subsequent requests to the content result in ActiveCDN serving the localized, cached content.

The ActiveCDN module will uninstall itself after a period of inactivity. And, of course, it can be reinstalled when new demands arise. The content provider controls the tunable parameters such as the inactivity time before module expiration. In fact, since the ActiveCDN module is written by the content provider specifically for the provider, the provider controls every aspect of the module's behavior, from the cache replacement policy to the algorithm to locate the nearest caching node. The module can even modify the video content on the fly, inserting watermarks or advertisements, for example. This

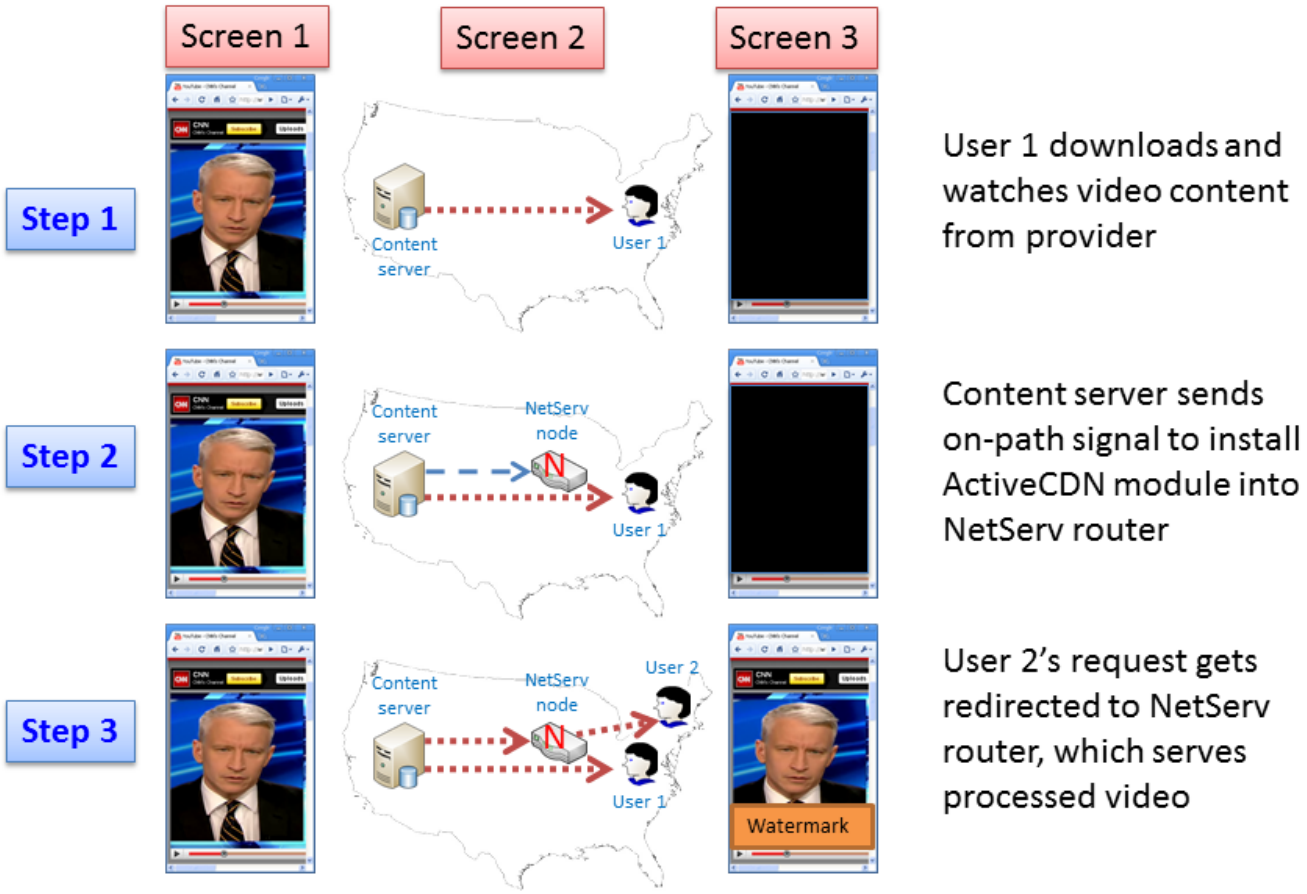

Fig. 2. A visual depiction of what happens in our ActiveCDN implementation. Our implemenation allows multiple users or clients to request video from the content server. The content server is able to control which nodes ActiveCDN is instantiated on, and redirect users to the node of its choice.

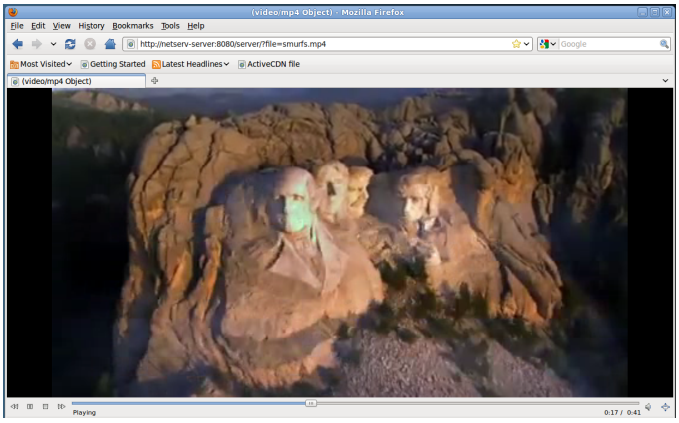

Fig. 3. A screenshot of the original video playing on a user's browser. This is the video that is directly served by the origin server.

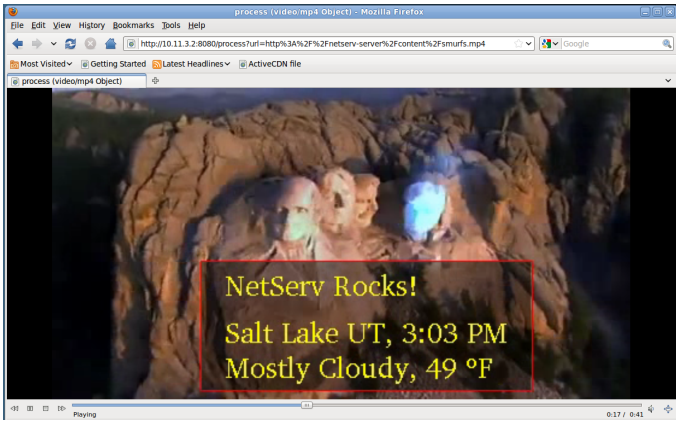

Fig. 4. A screenshot of the processed video playing from one of the ActiveCDN nodes. The video, in addition to being cached and served from the ActiveCDN node, has been processed with the local weather information added.

is indeed the biggest advantage of ActiveCDN compared to the traditional content distribution network. Using ActiveCDN, content providers can employ any distribution strategy that satisfies their need, rather than being locked in by the mostly static infrastructures of traditional CDN providers.

Figures 3 and 4 show screenshots of the content as seen by the user, for a request that is served from the origin server and by an intermediary ActiveCDN node respectively. As seen in the second screenshot, the ActiveCDN node can also perform content processing in addition to actively caching the content.

In this particular example, the ActiveCDN node retrieves the weather information for that location and creates a text overlay on top of the video.

Our current implementation gets the weather information from NOAA's web-based weather API service (weather.gov). This uses a latitude and longitude information, which in turn is obtained using MaxMind's GeoIP library that resolves IP addresses to a location. We use Xuggler to process the video and create a new video that has the weather overlay on top of the video.

IV. ActiveCDN on the Client

In addition to processing the content on the nodes, we have also worked on a mechanism to have content processed on the client, thus relieving the core or edge router from doing most of the processing. In this model, the ActiveCDN module would still do some processing, such as adding additional metadata and dynamic content information, but would only be adding textual or markup information, with all the heavyweight multimedia processing done on the client. This would free up resources on an edge or core router thus allowing it to handle a larger workload, while at the same time being indistuiguishable to the client who is able to see the same processed content. Finally, client-side processing using markup at an intermediate node allows true localization of content and information that is specific to the end user, while at the same time allowing the local edge node to tag the content with markup language that can be localized to a particular region.

For our current implementation, we used the SMIL markup language [14] to append content information which is sent to the client. SMIL is a multimedia integration, HTML-like language. It is typically used for multimedia presentations which integrate audio, video with images, text or any other media type. SMIL can be used to integrate different media objects on a NetServ router. In the previously described implementation Xuggler, a Java library, was used for content processing and data overlay while in this implementation, we generate a SMIL presentation file at the intermediate node which integrates different media along with the original video requested, and sends the SMIL markup to the end user.

The module which serves the SMIL file is implemented as a Java servlet process, similar to the previous implementation. It receives the video requests and serve a SMIL file, and also fetches the video from the content server and caches it.

The main components for the client-side ActiveCDN module are a content server, a NetServ node running the servlet process, and the Ambulant plugin for the Firefox browser which displays the SMIL presentation file to the user. Ambulant is an open source SMIL player which supports the SMIL 3.0 standard. The NetServ node stores a repository of text and video advertisements. For each video request, it fetches the local weather information as well as the latest news feed from a news source (BBC). A weather lookup (similar to that described in the previous section) is also carried out. We also include a video advertisement, which in our current

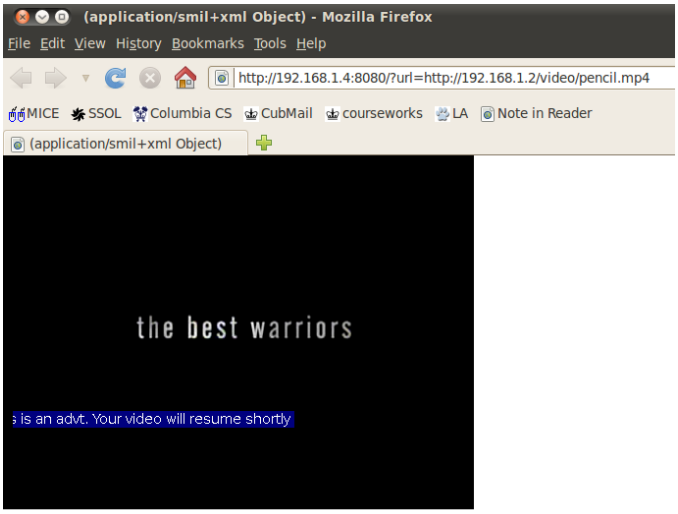

Fig. 5. Client processing: A screenshot of a video with a video advertisement in the middle of content.

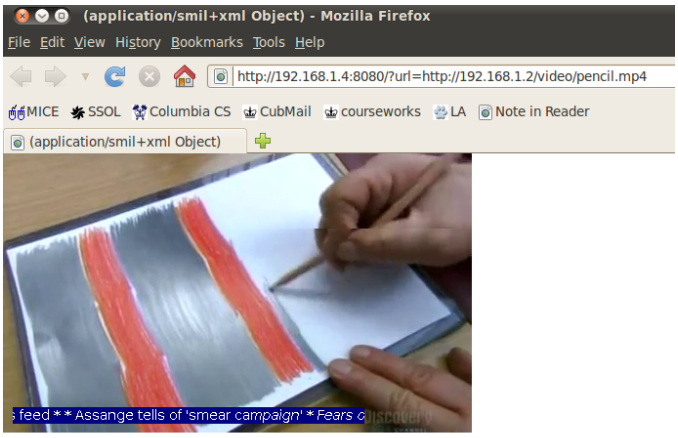

Fig. 6. Client processing: A screenshot of the video with a scrolling news ticker on top of the video.

implementation is simply a random video chosen from all the videos in a specified directory. All XML data are parsed with SAXBuilder class of JDOM Java library.

When the first request for the video comes in, this video is not in the NetServ node's cache. The servlet process in this case generates a SMIL file which points to the video on the content server and responds to the user with the generated SMIL file. The ActiveCDN module also downloads the video from the content server and caches it at the node.

For subsequent requests for the video, if the requested video is found in the cache, the servlet process generates a SMIL file pointing to the video found in cache. Along with this, it also fetches weather data, a video advertisement and latest news feed and adds them to the SMIL presentation. The video advertisement is randomly selected from a set of videos and inserted at a particular position in the original video. NetServ node runs a lighttpd server to serve the cached video when the request comes from Ambulant.

The final presentation to the end user in this implementation appears as follows:
1. Original video, with localized weather information as a text overlay
2. Advertisement video plays in middle of the original video
3. Rest of the original video has news ticker overlay on top of the video

V. Implementation

For the ActiveCDN server implemenation, we implemented and tested in on seven virtual machines, which were Fedora or Ubuntu Linux images running on VMWare. The seven machines consisted of one content server, two NetServ edge nodes and four clients. The networking and routing tables were set up so that the clients always connected through the NetServ nodes (the gateway) to the content server. We used Fedora for the content server and the NetServ nodes since we also ran the same experiment/setup on the NSF GENI Alpha testbed (which all ran Fedora). Our above-described implementation ran on the virtual machine topology, as well as the real-world GENI topology that spanned several states and timezones across the United States (particularly, Massachusetss, Kansas, South Carolina and Utah.)

We implemented ActiveCDN client (the "SMIL"implementation) on 1 physical machine running Ubuntu 10.04 and 2 virtual machines (on VirtualBox) running the same version of Ubuntu. The physical machine was used as the user machine running Ambulant plugin on Firefox browser. One of the virtual machines hosted the content server and the other virtual machine hosted the NetServ node running the servlet process.

The screenshots in the previous section were taken from running these implementations on top of the GENI topology and our local testbed respectively.

VI. Related Work

Traditional CDNs, such as Akamai [1] and Limelight Networks [15], started gaining traction in the late 1990s as large content providers resorted to using CDN services in lieu of their own hosting to save costs and increase efficient content delivery. In the early 2000s, large-scale ISPs (such as AT&T) started to build their own CDN functionality. While most of the early CDNs served large content providers (due to the costs involved in using a CDN), in recent years, services such as Amazon S3 [7] and Amazon CloudFront [8] have introduced the concept of pay-as-you-go CDN services which allow smaller websites to use CDN services for their content. There have been several studies of how commercial content-delivery networks operate, mostly through reverse-engineering [16], as well as research-oriented CDN services, particularly those such as CoDeeN [17] that run on PlanetLab. Static CDNs mostly comprise of statically deployed nodes placed at strategic positions and a tradeoff between cost and network efficiency. ActiveCDN introduces a new CDN paradigm by allowing the content publisher to create "pop-up" CDN nodes on-the-fly, dynamically.

Content-centric networking (CCN) [18] has gotten a lot of attention recently, and for good reason. Content-centric networking, which envisions computer networking based on content names rather than host names, allows for multiple, signed copies of content that can be fetched or placed at any location on the network, with requests for content being served from the closest location. However, while CCN discovery services such as flooding and broadcasting work well in local networks, it is hard to see how they scale to wide-area networks. None of the existing CCN implementations have yet been able to address this issue, thus making wide-area deployment and implementation of CCN applications difficult in the real-world. CCNx is one of the most complete architectures and implementations of content-centric networking today. There are other projects aimed at developing similar content-centric networking models, such as Nebula [19] and eXpressive Internet Architecture (XIA) [20], but they are still in their infancy and it remains to be seen how they fundamentally differ from the seminal CCNx work and what additional features they would have in comparison to CCNx.

ActiveCDN differs from CCNx significantly. In ActiveCDN, the publisher is able to instantiate nodes, as well as revoke node privileges, as necessary, while in CCNx, once content is published, it is free to be disseminated on any node. Naming in ActiveCDN is accomplished by the user typing in the regular URL to the content (which may be redirected as necessary), while CCN has a new naming architecuture. Routing in ActiveCDN is performed through redirection from the central content publisher to the node of the publisher's choice (usually determined by network location and geographic proximity), whereas in CCNx, routing in the local network is done through broadcast announcements for Interest packets till a corresponding Data packet is received in response. Routing on a wide-area network in CCNx is not clearly described yet. Finally, the most important feature, load-balancing, is done dynamically in ActiveCDN based on the number of requests and the density of requests coming from a particular area. In CCNx, the content is distributed all throughout the network, allowing for an infinite number of copies on all nodes.

CoralCDN [21] is probably the most prominent peer-to-peer, decentralized CDN, where content is served from P2P nodes that join and leave the network. Users, upon requesting content, have their DNS requests is resolved through a Coral DNS server which checks for DNS and HTTP proxies near the client's resolver. The user is then redirected to a Coral node near him with the content. The content publisher has no control over the location of the content, and in fact, no control over the quality of the network since the P2P nodes join and leave the network at frequent intervals. CoralCDN relies on a peer-to-peer network for content delivery, which may be unreliable as shown in the recent massive failure of Skype due to a bug that led to cascading failures [22]. ActiveCDN, in contrast, allows the content provider to dynamically control the instantiation of nodes as well as the positioning and location of nodes.

ActiveCDN allows for localization and processing of content. Also, it runs on NetServ-enabled nodes at edge routers that would be more reliable in real-world scenarios and backed by service guarantees.

Among the traditional CDN approaches that come closest to solving the CDN problem through a dynamic method are those that are optimized for handling flash crowds. A formal study by Yoshida [23] compares some of the state-of-the-art solutions for these in research, and I will contrast ActiveCDN with three of them. DotSlash [24] and FCAN [25] allow websites to deal with flash crowds through a pool of servers that are designed to handle flash crowds for websites. DotSlash allows several web sites (and their servers) to work with each other, using spare capacity to offset flash crowds to any of the participating sites. FCAN uses an overlay of caching proxies to store files and deliver them to users, and invokes this overlay when there are flash crowds. Globule [26], works in a manner similar to DotSlash and FCAN, and allows any web server to join the pool of available servers through the installation of an Apache module. These solutions require prior configuration of servers as well as spare processing power to handle flash crowd traffic. ActiveCDN is able to instantiate CDN nodes dynamically, based on demand.

CoopNet [27], DCDN (Distributed Content Delivery Network) [28] and SCAN (Scalable Content Access Network) [29] are hybrid CDN frameworks that combine both the infrastructure CDNs with peer-to-peer. DCDN uses the concept of ”surrogates” (Internet users with high bandwidth, similar to the ”supernode” concept in Skype) which handle content requests redirected from master or local DCDN nodes. CoopNet uses cached data at the clients to offload the central server, with a central server handling the redirects. These solutions require modification on the client-side in order to support the peer-to-peer mode of operation. ActiveCDN requires NetServ functionality at the edge routers, and does not require any modification on the client side.

MetaCDN [30] offers a ”mashup” service, allowing content providers to use cloud services such as Amazon CloudFront and others, through a single interface. MetaCDN takes care of replicating content and looking up the best location for each request. While MetaCDN is able to dynamically use cloud services APIs to store content in the ”cloud,” it is still restricted by the server locations of these cloud services since it does not actually provide any server functionality or control the location of the servers. ActiveCDN allows the content provider to install nodes at precise locations and points along the path.

Perhaps the feature that is most unique about ActiveCDN is its ability to do local processing, with the processing functionality completely controlled by the content publisher. While there is some existing work on video/content processing for networking applications, most of the work relates to the processing of the video in relationship to the video format itself [31] [32] or of text-based content [33], and not in regards to the content delivery network. Some recent patents [34] do discuss customized advertisement mechanisms, but assume the functionality is already installed, and focus more on the specifics of delivering customized ads, not on module placement and dynamic installation. ActiveCDN appears to be the first content delivery framework that allows the content provider to push custom content processing power into the delivery or service network.

VII. Future Work

We are hoping to expand ActiveCDN to allow for more advanced functionality. One of our current projects, an offshoot of ActiveCDN, is called

The ActiveCDN module could be used for more intelligent functionality in addition to caching and watermarking. They could provide the following features or services:
- Language translation: the content stores could have content translated into local languages using locale information.
- Content bridging: content stores could bridge wireless and wired networks, allowing for one content item to be streamed or played on multiple devices on a single wireless network. This would allow for video to be delivered on the local wireless network in a much smoother way, with less interference and less duplicate requests to the origin server for the same content.

VIII. Conclusion

In this paper, we described ActiveCDN, a novel approach to dynamic content distribution networks that can be deployed using network virtualization and cloud computing techniques. We believe that ActiveCDN is one of the first pieces of work to truly bridge the two disparate worlds of content delivery networks (CDNs) and cloud computing models. In addition to the dynamic cloud deployability features, we have also presented how ActiveCDN can perform processing and media transformation as required by the publisher, thus adding true CDN service virtualization and presenting it as an in-network functionality.

IX. Acknowledgment

This work was supported by the National Science Foundation (NSF) under Grant 04-54288.

References

[1] Tom Leighton, “Improving performance on the internet,” in *CACM (Communications of the ACM)*, February 2009.
[2] “Nsf geni project.” [Online]. Available: http://www.geni.net/
[3] P. V. Mockapetris, “Domain names concepts and facilities,” in *RFC 1034*.
[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol – http/1.1,” in *RFC 2616*.
[5] “Obama inauguration drives record web usage,” in *PC World*, January 2009. [Online]. Available: http://www.pcworld.com/article/158029/obama_inauguration_drives_record_web_usage.html
[6] G. Pallis and A. Vakali, “Insight and perspectives for content delivery networks,” in *Communications of the ACM*, January 2006.
[7] “Amazon simple storage service (s3),” http://aws.amazon.com/s3/. [Online]. Available: http://aws.amazon.com/s3/
[8] “Amazon cloudfront,” http://aws.amazon.com/cloudfront/. [Online]. Available: http://aws.amazon.com/cloudfront/
[9] S. Pandey, K. Gupta, A. Barker, and R. Buyya, “Minimizing cost when using globally distributed cloud services: A case study in analysis of intrusion detection work?ow application,” in *CloudCom 2009*.

[10] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang, "Optimal pricing of content delivery network (CDN) services," in *International Conference on System Sciences*, 2004.

[11] S. Srinivasan, J. W. Lee, E. Liu, M. Kester, H. Schulzrinne, V. Hilt, S. Seetharaman, and A. Khan, "NetServ: Dynamically Deploying In-network Services," in *ACM ReArch '09 (CoNEXT workshop)*, 2009.

[12] J. W. Lee, R. Francescangeli, J. Janak, S. Srinivasan, S. A. Baset, H. Schulzrinne, Z. Despotovic, and W. Kellerer, "Netserv: Active networking 2.0," in *IEEE FutureNet workshop*, 2011.

[13] "Xuggler java library," http://www.xuggle.com/xuggler/. [Online]. Available: http://www.xuggle.com/xuggler/

[14] "Synchronized multimedia integration language 2.0," http://www.w3.org/TR/2005/REC-SMIL2-20050107/. [Online]. Available: http://www.w3.org/TR/2005/REC-SMIL2-20050107/

[15] "Limelight networks," http://www.limelightnetworks.com/. [Online]. Available: http://www.limelightnetworks.com/

[16] C. Huang, A. Wang, J. Li, and K. Ross, "Measuring and evaluating large-scale CDNs," in *Internet Measurement Conference (IMC)*, November 2008.

[17] L. Wang, K. S. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and security in the codeen content distribution network," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2004.

[18] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of CoNEXT '09*, 2009.

[19] "Nebula," http://nebula.cis.upenn.edu/about.html. [Online]. Available: http://nebula.cis.upenn.edu/about.html

[20] "expressive internet architecture (xia)," http://www.cs.cmu.edu/ xia/. [Online]. Available: http://www.cs.cmu.edu/ xia/

[21] M. J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing content publication with coralCDN," in *USENIX 1st symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[22] S. official blog, "Cio update: Post-mortem on the skype outage," http://blogs.skype.com/en/2010/12/cio_update.html. [Online]. Available: http://blogs.skype.com/en/2010/12/cio_update.html

[23] N. Yoshida, "Dynamic CDN against flash crowds," http://www.springerlink.com/content/q2j5130831r84365/. [Online]. Available: http://www.springerlink.com/content/q2j5130831r84365/

[24] W. Zhao and H. Schulzrinne, "Enabling on-demand query result caching in dotslash for handling web hotspots effectively."

[25] C. Pan, M. Atajanov, M. B. Hossain, T. Shimokawa, and N. Yoshida, "Fcan: Flash crowds alleviation network," in *2006 ACM Symposium on Applied Computing (SAC 2006)*, 2006.

[26] G. Pierre and M. Steen, "Globule: A collaborative content delivery network," in *IEEE Communications Magazine*, August 2006.

[27] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *IEEE NOSSDAV02*, May 2002.

[28] J. Mulerikkal and I. Khalil, "An architecture for distributed content delivery network," in *IEEE Conference on Networks*, 2007.

[29] Y. Chen, R. Katz, H. Randy, and J. Kubiatowicz, "Scan: A dynamic, scalable, and efficient content distribution network," in *Proceedings of the First International Conference on Pervasive Computing (Pervasive '02)*, 2002.

[30] J. Broberg, R. Buyya, and Z. Tari, "MetaCDN: Harnessing storage clouds for high performance content delivery," in *Journal of Network and Computer Applications, Volume 32, Issue 5*, 2009.

[31] J. Magalhaes and F. Pereira, "Using mpeg standards for multimedia customization," in *Journal of Signal Processing: Image Communication (Elsevier)*, May 2004.

[32] Z. Liu, D. Gibbon, and B. Shahraray, "Multimedia content acquisition and processing in the miracle system," in *IEEE CCNC 2006*, January 2006.

[33] C. Yuan, Y. Chen, and Z. Zhang, "Evaluation of edge caching/off loading for dynamic content delivery," in *IEEE Transactions on Knowledge and Data Engineering*, October 2004.

[34] J. Tidwell, E. Samame, and B. Santangelo, "Methods and apparatus for evaluating an audience in a content-based network," in *U.S. Patent Application Publication #US20110016482*, January 2011.