Unveiling the content-centric features of TCP

Suman Srinivasan*, Ivica Rimac†, Volker Hilt†, Moritz Steiner†, and Henning Schulzrinne*
* Columbia University
New York, NY 10027, USA
Email: {sumans,hgs}@cs.columbia.edu
† Bell Labs, Alcatel-Lucent
Holmdel, NJ 07733, USA
Email: {firstname.lastname}@alcatel-lucent.com

***Abstract*—Content-centric networking has been proposed as a new networking paradigm that is centered around the distribution of content. A key idea of content-centric networks is to address content by name and to enable nodes in the network to respond to content requests. Most proposals for content-centric networks require a "clean slate" approach and a replacement of today's TCP/IP protocol stack, which raises questions about a feasible deployment path.**

In this paper, we ask the question to which extent the ideas of content-centric networks can be realized on top of today's IP protocol suite. We explore an approach for name-based addressing that extends today's TCP/IP protocols in a fully standard compliant way. We implement our new method in order to demonstrate its feasibility and evaluate the performance of the system using both latency and processing overhead as measures. The obtained results demonstrate that name-based addressing on TCP/IP is feasible. We also acknowledge that content-centric networks designed from the ground up go beyond what can be achieved on top of IP.

I. Introduction

Content-centric networking has been proposed as a new networking paradigm centered around the distribution of content [11][14][13]. A key idea of content-centric networks is to address content by its name and not adsress the host serving the content. Any network node is enabled to cache and respond to a request if the node holds the requested content item. Current proposals for content-centric networks require a departure from today's IP protocol stack. However, replacing today's IP protocol stack with something new requires substantial investment in network infrastructure and end systems. Given the very slow uptake of the next version of the IP protocol, IPv6, it is questionable when and if at all a radically new protocol stack can see widespread deployment.

In this paper, we are questioning the common belief that content-centric networks requires a departure from the IP protocol suite. We explore to which extent content-centric networks can be realized on top of IP. The core element of content-centric networks is the ability to address content by name. We propose an appraoch for a name-based content nework, On-Path CDN, that extends TCP/IP in a fully standard compliant way. Our approach enables any node on-route between the end user and the content provider to serve requested content.

We have implemented our new method in order to demonstrate its feasibility and evaluate the performance of the system using both latency and processing overhead as measures. The obtained results demonstrate that it is possible to realize a name-based addressing mechanism on TCP/IP. We also show that On-Path CDN can enhance end-user experience when consuming data over the Internet. We acknowledge that content-centric networks designed from the ground up go beyond what can be achieved with an IP compliant approach.

We detail the operations of traditional and on-path CDNs in Section 2. We explain our mechanism and the implementation of a prototype system in Section 3 and Section 4, respectively. In Section 5, we evaluate the performance of our system using network latency and processing overhead as metrics. We discuss related work in Section 6.

II. CDNs for Content-Centric Networking

A. Traditional approach

We first contrast our work on On-Path CDNs to existing methods of content delivery by highlighting two issues.

The first issue is in regards to the method of content delivery itself. Today, content providers either host their high-bandwidth content themselves, or more commonly pay content delivery network (CDN) providers such as Akamai and Limelight Networks for the delivery of their content. When content is hosted on a CDN, a user request for it is usually redirected to a server closer to the user that is operated by the CDN vendor. Though this fundamental task appears to naturally fall into the realm of the network, the host-centric Internet architecture was not designed with such a service. This has led to the development of application-specific and non-interoperable mechanisms, the most common of which are: redirection through domain name resolution using DNS [17][18]; request redirection using HTTP [10]; and other application-level mechanism, e.g., based on HTML rewriting or distributed hash tables (DHTs).

The above redirection work-arounds require central control and impede cooperation of CDNs operated by different parties; e.g., to allow a "long-range" backbone CDN to reach into a metro network and make use of the CDN resources of the local operator. Once the redirection is set up, a local CDN will be unable to serve the content from another node even if it is closer to the end user than the node that the user has been redirected to. As a consequence, CDNs are rather statically

deployed and scaling them to adapt to sudden changes, such as unexpected flash crowds, is hardly possible.

The second issue is in regards to the networking architecture and naming perspective. Currently, requests for content are usually routed based on the Internet address (IP address) of the node that has the content. While this is in keeping with the current Internet architecture, it does not offer a correct way of addressing content, which is independent of the location of the content itself. Methods of addressing naming issues, such as Content-Centric Networking [11] as well as systems like DONA [14] and i3 [13] require a clean-slate implementation of the Internet in order to be useful.

Our work presents an implementation of content-centric networking that runs on today's Internet technologies and protocols. In this paper, we raise and answer the question of how far we can go in the direction of CCN based on today's IP protocol suite, with our additions being standards compliant. We propose a design for an IP compliant CCN architecture, present a prototype implementation, discuss limitations and performance as well as unexpected road-blocks in the implementation.

B. On-path content delivery

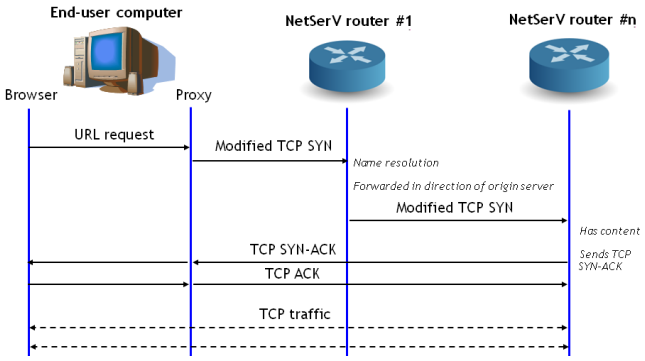

Fig. 1. The handshakes and networking messages used in the TCP-interception method of on-path CDNs.

We propose an alternative method for the delivery of multimedia content that enables delivering multimedia content from any node that is on the route from the end user to the content provider. In our approach, any intermediate node can respond to content requests and serve content if it has a copy of the content cached. This avoids explicit redirection to another server and reduces latency, thereby improving the end user's multimedia experience.

In addition, our implementation and solution also provides a method of serving content to the users without redirecting them to a particular node. Our on-path content delivery mechanism enables true content-based delivery without the network having to worry about which nodes the content resides on, etc. Our implemenetation also enables content-based networking on today's Internet, without any modifications to the underlying TCP/UDP/IP protocols, which we believe to be an extremely important contribution of our project.

Even though on-path content delivery is a simple concept, in its true form it departs from the current Internet design. Since true on-path content delivery would require changes to the existing networking protocol suite, it is not surprising that redirection is the preferred method of CDNs for serving multimedia content in a reliable and fast manner.

However, our approach to on-path content delivery is deployable on today's Internet infrastructure and uses existing networking protocols. Our on-path content delivery method makes use of signaling messages that piggyback on existing TCP handshake mechanisms (which are used to set up Internet sessions), as described in the next section.

C. Advantages of on-path content delivery

There are many advantages in allowing multimedia content to be served using intermediate nodes on the path from an end user to a content provider, in contrast to using a statically deployed CDN network, or worse yet, serving content from just one central location. Some of the advantages are:

- Network latency is reduced and there is less congestion on the Internet. Especially as video traffic is predicted to grow and account for 90% of the Internet traffic in 2013 [16], having video served from nodes that are closer to the end user will dramatically reduce congestion in the network.
- Regional networks and the Internet dynamically adjust to high loads of unanticipated traffic, e.g., such as in the case of "flash crowds" or highly viral content.
- Service providers can reduce redundant traffic on expensive transit links.
- CDN service providers can establish business relations with each other to provide better end-to-end multimedia experience to a larger user population.

We describe the design decision and details of the core mechanism of our solution, which eventually enables these advantages, in the following section.

III. Mechanism Details

Non-realtime video on the Internet today is delivered almost exclusively using HTTP, with TCP as its underlying transport protocol. Since the goal of our work is to enable a solution over today's Internet without any modifications to the end user and networking stack, we focus on a solution that is compliant with and requires no changes to the TCP specification [6].

At the beginning of a HTTP session the user application establishes a TCP connection to either the host indicated in the URL [8] that identifies the requested content or to a proxy if configured. Once the TCP session has been established, the user application then uses the URL to issue a HTTP request identifying the content. Thus, intercepting a content request on path for inspection and content routing decision requires terminating the TCP session first. Since an established TCP connection cannot be renegotiated and transferred, however, on-path routing without additional mechanisms could easily result in a daisy chain of TCP connections for a single user session.

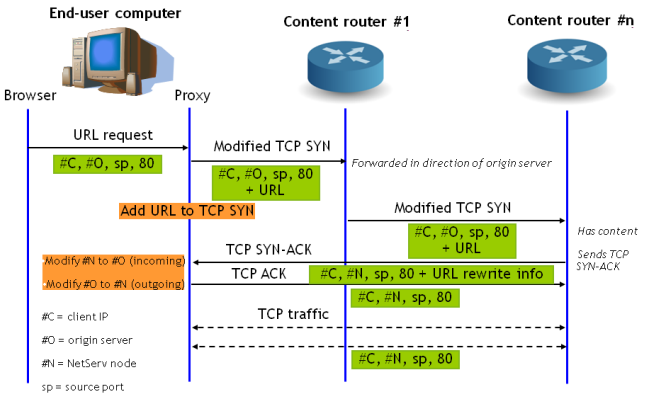

Fig. 2. The details of implementing the handshakes and networking messages in the TCP-interception method.

Session establishment in TCP is signaled using the same packet format as for the actual transfer of the application data; i.e., a TCP packet consists of a header and a payload data. Our method of on-path content delivery exploits the fact that the TCP packets for the initial handshake of a session don't include any payload. We piggyback some key information in two of three TCP handshake messages, particularly the TCP SYN and the TCP SYN-ACK packets. Though this would be possible by modifying the protocol stack at the user end, we enable this through use of a modified HTTP proxy that rewrites the TCP handshake messages as outlined in Figure 1 and depicted with greater detail in Figure 2. In the proxy scenario, the application (e.g., the browser) establishes a session with the proxy server, which rewrites the TCP handshake messages to enable the on-path delivery mechanism. Thus, there is no need to rewrite the networking stack at the end user's node, and at the same time, the user is able to benefit from an "opt-in" option to make use of this feature.

Our method uses the first TCP SYN message from the client to the content provider for carrying the pointer to the requested content (such as the URL) in its payload. This message moves through the network in the direction of the content provider. It passes through all intermediate nodes that are not able to serve the content (or are ignoring TCP SYN payloads). However, once an intermediate node detects it has the content referenced in this packet, it terminates the request forwarding process and replies to the TCP SYN packet with a TCP SYN-ACK packet.

The TCP SYN-ACK packet sent by the intermediate node contains a payload consisting of the initial content pointer (the URL), along with a delimiter and a an identifier of the node that has intercepted the request. This allows the networking stack in the proxy server that is serving the end user to realize that an intermediate node is responding to the initial content request.

When the client proxy receives the TCP SYN-ACK message, it replies with an ACK to the intermediate node, thus completing the TCP handshake. At this point, a TCP session is set up between the proxy and the intermediate node that can serve the content directly, and all future content requests are served directly from the intermediate node.

In order to proof our concept, we have a implemented a prototype described in the next section.

IV. Prototype Implementation

Our prototype implements the previously described the TCP handshake interception. Furthermore, a module in the prototype evaluates each handshake message to determine whether it is able to handle and serve the content that is referenced in the TCP SYN message, and only then does it decide whether to respond to it with a payload-added SYN-ACK message.

For our implementation we used iptables [2] and netfilter [4] to set up rules to intercept our required packets. We also use the library libnetfilter_queue (nfqueue) [4] to allow us to programatically set up hooks to network events. We intercepted requests for packets on both ports 80 (the normal port for HTTP) and 3128 (for Squid proxy caching, which we used.)

We implemented our prototype using the Python programming language and relevant network libraries for intercepting and modifying packets. We used the nfqueue-bindings [4] and the Scapy [5] libraries for advanced networking and packet functionality. Our Python scripts implemented the functionality required for the protocol as described in the previous section.

We ran our tests on Alcatel-Lucent (ALU) networks, and hence ALU is considered the local network. We use a HTTP proxy to communicate with nodes outside the Alcatel-Lucent network.

Our setup consisted of:

- A client with a browser, whose is set up to use our on-path CDN proxy to access the network.
- Our CDN proxy, which handles the initial TCP SYN packet and also performs the modification of network addresses in a manner similar to a NAT.
- Our CDN caching node, which intercepts the TCP SYN signaling message, responds with a TCP SYN-ACK with the node information payload, and serves the cached content from its local storage.
- The origin server, which is the original source for the requested content.

Of the above, we implemented the CDN proxy and the CDN caching node. Both of these were implemented in the Python programming language using the networking libraries listed.

One of the most interesting observations we made was with the implementation of the network stacks we experimented with. In order to intercept and serve content based on request, we added payloads to the TCP SYN and SYN-ACK packets, and correctly changed the size and the checksum on these packets. To our surprise, the TCP SYN and SYN-ACK packets were recieved and processed correctly by the network stack at the sending and receiving nodes, but the subsequent data packets were not handled properly, and we saw TCP RESET messages terminating the session.

We found that the reason this was happening was that the network stacks were assuming (incorrectly) that the TCP handshake messages had a zero payload and were expecting corresponding SEQ and ACK counters. In other words, the operating systems' network stack implementations completely neglected the data size and SEQ/ACK numbers set in the TCP handshake messages and started the SEQ/ACK numbers for the data packets assuming a zero payload. This caused some unexpected problems in the running of our protocol.

In order to resolve this SEQ/ACK problem, our packet processing had to change the SEQ and ACK numbers in the TCP packets, reducing or incrementing them as necessary to allow the network stack to recognize them as legitimate packets.

V. Performance Evaluation

For our performance measurements, we measured the network latency using the round-trip-time delay from one control node to four content servers that we worked with during the course of our experiments.

A. Experimental setup

We used one node in the Alcatel-Lucent network as the control node. We measured the round-trip time of TCP packets to the following nodes:

- A node on the same subnet
- A node on the edge of the network: Alcatel-Lucent's main web server (www.alcatel-lucent.com)
- Cable New Network (CNN's) main web server (www.cnn.com)
- Akamai's content server for CNN (ht.cdn.turner.com)

B. Round-trip time and latency

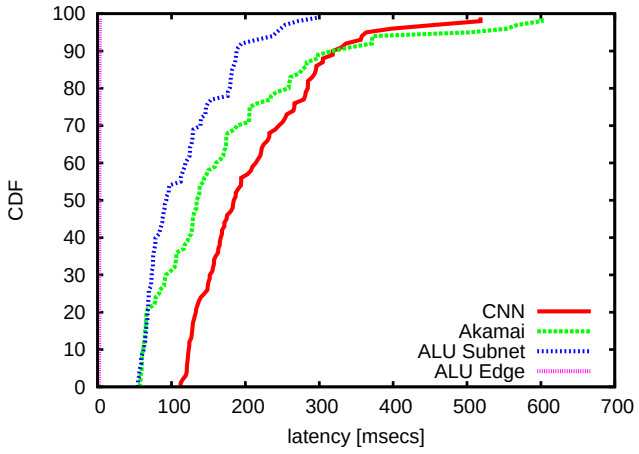

Fig. 3. The performance graph showing the latency/delays of the four test nodes. Latency for the edge node is far less than the latency in reaching the content provider or its CDN network.

We used apachebench [1] to test the delays. apachebench allows us to measure round-trip time delays for TCP signaling messages, which are very useful for us since we want to measure the delay in sending and receiving TCP messages. We used Python's scapy [5] network package for generating the network graph, as well as validating the results from apachebench.

The results presented in Figure 3 show that the round-trip times for the CNN and Akamai servers are very close to each other. We believe that this may be because our tests were conducted in the U.S., and both servers were sufficiently close enough in terms of network topology to respond to requests in similar time. The interesting result is the difference in the delay between the edge node (the Alcatel-Lucent web server) and the CNN and Akamai servers. Our measurements indicate that latency to the edge node is only around 50% of the latency for CNN and Akamai; and latency for the edge node is also far more consistent than the latency to the origin servers, whether CNN or Akamai's servers.

The above results show that an on-path CDN with deployed nodes at the edge of the network can indeed substantially reduce network latency, thus providing a far better multimedia experience to the end user. Furthermore, we conclude that it can reduce network congestion and other networking problems associated with large volumes of multimedia traffic on the Internet.

C. Processing overhead

Another important performance metric is the processing overhead that our mechanism induces on the network nodes. As the results of our performance measurements indicate, this overhead is very small and is not a potential for overload. The typical overhead for simply intercepting and checking the content of the TCP handshake messages is in the order of 400 to 500 microseconds, and the overhead of interception and modification of the TCP messages is in the order of 600 to 700 microseconds. Hence, the overhead added to the interception is the range of 40% to 50%.

It is important to note that the interception is done only on TCP handshake messages, particularly only on TCP SYN messages on the content router. Furthermore, the payload addition is done only for content that the router is able to handle. And finally, our prototype was built using the Python scripting language and a iptables/netfilter implementation that sent packets from kernel space to user space. With an implementation in a kernel module the speed of processing packets could be significantly improved.

VI. Alternate Implementations

There are several alternate ways of implementing our on-path method for serving content, such as by using UDP signaling. We did not implement these mechanisms since we believe that being able to intercept and serve content in response to TCP message signalling is the best way to handle content delivery on network flows that are being set up. In addition, such methods work on an "off-path" channel or as control messages, and we believe the TCP intercept mechanism allows for true content-centric networking to be implemented on today's Internet architecture.

VII. Related Work

There has been work on intercepting content requests to serve multimedia content from nodes on path, such as redirecting requests based on predetermined criteria [19]. However, the work is either not as comprehensive as ours or does not have specific architecture or implementation details.

Layer 4/7 switches [3], also known as web or content switches perform similar functionality. In contrast to our solution, however, they have are required to keep track of every TCP connection that they are routing. This is impractical in a large network.

TCP interception has been enabled by Cisco on its routers. While this seems to have been initially introduced to allow systems to reduce TCP SYN-flooding attacks, such as NetBouncer [20], it has also been used for a variety of purposes, such as speeding up content delivery on wireless networks [7] [15]. While we were not able to find related work using this feature for content delivery, we believe this feature allows for our method to be implemented on many real-world routers.

There is also some work done on content-based routing, such as QoS routing for multimedia applications [21] and multipath routing for unicast video [9], but these works on multimedia routing are fundamentally different from our work which involves intercept-based content services.

There has been a body of work in the fields of naming and redirection. For example, i3 [13] and OCALA [12] are naming overlays that work on top of existing Internet architectures. The Data-Oriented Network Architecture (DONA) [14] is a similar attempt to address the issue of Internet naming and name resolution.

In contrast to these naming and addressing approaches, Content-Centric Networking (CCN) [11] aims at treating content as a primitive for routing requests to the destination. However, most of the robust work on naming, addressing and content based routing require a clean-slate redesign of the Internet. We believe ours is the first example of work that enables use of signaling for delivery of multimedia content without requiring end-to-end modification to Internet protocols.

VIII. Conclusion

We have introduced the concept of on-path content delivery networks, which are able to serve multimedia content to the end user through signaling messages. We described our motivation for developing this new architecture, and detailed how this architecture helps in improving the network experience for the end user, while reducing networking problems such as congestion and latency. We have detailed the protocol call-flows for enabling On-Path CDN, and have also described the implementation of our prototype on existing Internet infrastructure. We also presented performance measurements that show how the on-path CDN is an improvement over existing statically deployed CDNs in terms of reducing latency and round-trip time.

Our current work implements a simple prototype of the on-path CDN using TCP handshake signalling messages. A more advanced version of this implementation could handle larger and more specific URL requests, and be able to dynamically adapt content delivery through detecting the URL in the TCP handshake packets.

Our approach enables a new class of content networking architectures with a large number of dynamically deployed content nodes. This departs drastically from the existing approach of statically deployed CDN networks. New networking models are required for this new class of CDN networks, and we will carry out performance analysis and evaluations of this new class of architectures in future work.

IX. Acknowledgments

The NetServ project is funded by the National Science Foundation as a part of its Future Internet Design initiative.

References

[1] Apachebench. http://httpd.apache.org/docs/2.0/programs/ab.html.
[2] iptables. http://www.netfilter.org/.
[3] Layer 4/7 switch (web switch). http://en.wikipedia.org/wiki/Multilayer_switch.
[4] nfqueue bindings for netfilter. http://software.inl.fr/trac/wiki/nfqueue-bindings.
[5] Scapy. http://www.secdev.org/projects/scapy/.
[6] Transport control protocol. Technical report, RFC 793, 1981.
[7] G. S. Barron C. Housel and D. B. Lindquist. Webexpress: A client/intercept based system for optimizing web browsing in a wireless environment. *Mobile Networks and Applications*, 2004.
[8] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (url). Technical report, RFC 1738, 1994.
[9] J. Chen and S. Chan. Multipath routing for video unicast over bandwidth-limited networks. In *Proc. of GLOBECOM'01*, 2001.
[10] R. Fielding, J. Gettys, and et al. Hypertext transfer protocol – http/1.1. Technical report, RFC 2616, 1987.
[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. of CoNEXT '09*.
[12] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. Ocala: an architecture for supporting legacy applications over overlays. In *Proc. of NSDI'06*.
[13] J. Kannan, A. Lakshminarayanan, I. Stoica, and K. Wehrle. Supporting legacy applications over i3. *IEEE Computer*, 2004.
[14] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *Proc. of SIGCOMM'07*, New York, NY, USA. ACM.
[15] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi. Split tcp for mobile ad hoc networks. In *Proc. of GLOBECOM'02*.
[16] J. Markoff. Scientists strive to map the shape-shifting net. *New York Times*, 2010.
[17] P. Mockapetris. Domain names - concepts and facilities. Technical report, RFC 1034, 1987.
[18] P. Mockapetris. Domain names - implementation and specification. Technical report, RFC 1035, 1987.
[19] J. D. W. B. Networks. Method and system for redirecting web page requests on a tcp/ip network, 2002.
[20] R. Thomas, B. Mark, T. Johnson, and J. Croall. Netbouncer: Client-legitimacy-based high-performance ddos filtering. In *Proc. of DISCEX III*, 2003.
[21] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE JSAC*, 1996.