2016 IEEE 30th International Conference on Advanced Information Networking and Applications

ContentSDN: A Content-based Transparent Proxy Architecture in Software-Defined Networking

Alex F R Trajano
Universidade Estadual do Ceará
Fortaleza, Ceará, Brazil
alex.ferreira@uece.br

Marcial P Fernandez
Universidade Estadual do Ceará
Fortaleza, Ceará, Brazil
marcial.fernandez@uece.br

***Abstract*—In recent years, there has been a trend towards changing the Internet paradigm from location-based to content-based. Based on the principle that networks should allow users to focus on data, rather than having a reference to a physical location where that data is, the adoption of new paradigms can be justified. Information-Centric Networking (ICN) is a new paradigm that gives a potential improvement to content distribution, such as local content caching, which helps to reduce delay and increase delivery speed. From another point of view, Software-Defined Networking (SDN) defines a new Internet architecture where network devices are simple and the control plane is managed by a centralized controller, providing an inexpensive and reliable architecture. This paper presents ContentSDN, an architecture that integrates ICN and SDN, providing transparent in-network caching for content delivery. ContentSDN uses the concept of Network Functions Virtualization (NFV), allowing network orchestration and high scalability at runtime. Content caching is driven by manageable policies that can be adjusted according to business needs. The proposed architecture has been evaluated on an emulated environment where multiple users request content through the HTTP protocol. Results show that ContentSDN improves the Quality of Experience and make a more efficient use of network resources.**

***Index Terms*—Information-Centric Networking (ICN); Software-Defined Networking (SDN); Network Functions Virtualization (NFV);**

I. Introduction

The original project of the Internet, made almost 50 years ago, focused in point-to-point communication and technical users. Due to its evolution and popularization, the Internet became a successful, effective, global-scale communication system. Furthermore, the arise of new services like e-commerce, social networks, file sharing, Voice over IP (VoIP), online games, video stream, and others, showed the limitations of the original design. However, in order to support such evolution process, the Internet's architecture has been suffering many amendments, becoming it one more complex and increasing the cost for implementation, maintenance and management of new applications. This process is known as the Internet ossification [1].

The Internet application profile changed over the years. We have seen a huge growth in video streaming services, even live and on-demand. In 2016, the annual global IP traffic will surpass the zettabyte (1000 exabytes) and is expected to reach two zettabyte in 2019. In 2019, Internet video traffic will be 80 percent of all Internet traffic, up from 64 percent in 2014, excluding video exchanged through peer-to-peer (P2P) file sharing. Considering only IP traffic that crosses the Internet and is not confined to a single provider's network, internet video will grow 33% per year and online gaming will grow 40% per year from 2016 to 2019 [2].

Currently, content requests are handled individually, leading to an independent flow from Content Delivery Network (CDN) to each user. Such an unicast content delivery paradigm ignores that much of the content is identical to other user's requests made a few moments earlier. Therefore, a very large amount of identical media objects are delivered on the same network segment repeatedly, forcing an unnecessary increase in provider's bandwidth to support the growing number of users and content popularity. Then, we believe that new approaches should be necessary to improve the content distribution efficiency.

The triggering this context is reflex of incompatibility between models, the original Internet's architecture and current web applications. IP packets were predicted for simple end-to-end applications, but the nature dynamic of the Internet requires more flexibility. The great content production, 500 exabytes in 2008 [3], is only one example it. Today, the Internet users are looking by "what" content they want. However, the Internet architecture is based on "where" is the content. In the 90s appears the most successful Internet application: the web search engine. They simply associate user's content terms and try to suggest sites where this content could be satisfied.

The programmable network concept is coming back in evidence thanks to Software-Defined Networking (SDN) architecture. The proposal was started by the OpenFlow approach, that defines an open protocol that sets up forward tables in switches, which can be actively modified by the network user [4]. This proposal brings an abstraction layer of the switch, where is possible decouple data and control planes. The switch is now used only as a hardware device, to forward data, while the policy management might be done totally separated.

The OpenFlow in SDN architecture provides several benefits: (1) OpenFlow centralized controllers can manage all flow decisions reducing the switch complexity; (2) a central controller can see all networks and flows, giving global and optimal management of network provisioning; and (3) OpenFlow switches are relatively simple and reliable, since forward decisions are defined by a controller, rather than by a switch

1550-445X/16 $31.00 © 2016 IEEE
DOI 10.1109/AINA.2016.103

firmware [5].

Over the last few years, Network Functions Virtualization (NFV) [6] has been becoming one of the most promising study areas for developing new computer network technologies. NFV poses a novel way to develop network services, by using software and virtualization aiming the replacement of proprietary hardware appliances that run network functions, like Network Address Translation (NAT), Intrusion Detection System (IDS), caching, etc. In NFV, these services, called Virtualized Network Functions (VNFs), are implemented through software and deployed in Virtual Machines (VMs), allowing new and efficient ways of network deployment. NFV allows to manage and customize network services according to business needs, enabling tremendous cost savings and more agility to serve the daily changes that networks are susceptible.

This work presents ContentSDN, a content-based caching architecture for SDN that is completely transparent for both users and network applications. ContentSDN is designed to provide high scalability and manageability, through an architecture based on the principles of NFV and the microservice architectural pattern, providing high deployment flexibility and high availability independently of the network's nature. Furthermore, ContentSDN allows management driven by policies, having distributed decision-making components and multiple caches distributed across the network, which interacts with the SDN centralized control plane. The proposed architecture has been designed for enabling in-network caching for any protocol that follows the client-server model that identifies contents by unique names, like HTTP.

The rest of the paper is structured as follows. In Section II, we present some related work. Section III introduces the NFV and SDN fundamentals; and in Section IV we present the ContentSDN architecture. Section V shows the proposal evaluation and Section VI shows the results. Finally, Section VII concludes the paper.

II. Related Work

In recent years, some proposals tried to change the current end-to-end IP packet and web search engines to a new content based network architecture, called Information-Centric Networking (ICN) [7]. The most remarkable proposal was the Named Data Networking (NDN) [8], initially known as Content-Centric Network (CCN) [9]. It is based on the principle that the Internet should allow a user to focus on the data it needs, rather than referring a physical location where this data could be retrieved. The current Internet architecture is host-based; it was created to allow the dissemination of few fixed computers to many geographically distributed users. The NDN seeks to change the Internet architecture to the new usage behavior, where the routing is based on content's name instead of the IP address. Several works tried to adapt ICN architectures to operate on OpenFlow networks [10], [11], [12], [13].

Chandra et al. propose a caching architecture for HTTP atop of a SDN infrastructure [14]. The authors discuss that even though OpenFlow maintains an abstraction of control and forwarding planes, it faces serious problems when dealing with ICN, since it does not provide abstractions for content. They discuss that the flow granularity does not work well with content, which can be easily understood when analyzing HTTP requests, where multiple content requests can pass through a single TCP channel, which is handled as an individual flow on OpenFlow switches. The proposed architecture is based on a unique proxy and several caches spread over the OpenFlow network. All HTTP GET requests are forwarded to the proxy through static flows installed on the switches. When a request reaches the proxy, it queries the OpenFlow controller in order to resolve the final destination of the request, which can be either a web server or a local cache. If a request reaches a local cache, it will fetch the content on the origin web server and store the content in order to serve further requests. An interesting feature of this architecture is the centralized content management layer, that allows transparent in-network caching and maintains the TCP semantics without modification to both clients and servers. The main drawback in this architecture is the existence of a single proxy, which can trigger serious scalability problems, since every HTTP request would be forwarded to the proxy. Besides, the OpenFlow controller would be flooded with queries originated from every HTTP request. The evaluation showed that their implementation reduces the time to download files from a remote server, when compared to a non cache approach.

Georgopoulos et al. present the OpenCache; an OpenFlow-assisted in-networking caching system specialized in Video-on-Demand (VoD) [15]. OpenCache consists in two basic components, the OpenCache Controller (OCC) and the OpenCache Node (OCN). OCC is responsible for identify which videos should be cached, while the OCN provides the storage needed for video caching. OCC requires the network administrator to call a method on its Application Programming Interface (API) in order to start the caching of specific videos. If some video has been configured for being cached, the OCC tells the OpenFlow controller to create flows to redirect the VoD traffic to the closest OCN instance of the network, leading the traffic directly to the cache. When video requests reach an OCN instance, it acts as a proxy if the video has not been cached yet, forwarding the request to the origin server and storing the video response to serve further requests. Each OCN instance is responsible for handling a set of closest users, without any cooperation of other OCN instances, which could cause local network congestion and does not provide great scalability due to the limited resources within a single OCN instance. Furthermore, it is not practical to require the network administrator to start manually the caching of some video since this type of content is being produced at a much higher rate than the network administrator can handle. Although the experiments made on OpenCache have shown good results on video startup delay, external link utilization and video quality, the experiment setup has used just a single video client, which is not capable of give an overview of how OpenCache would behave in production networks with multiple accesses.

III. NFV/SDN Overview

Network Functions Virtualization (NFV) offers a new network architecture to design, deploy and manage network functions and services in a virtualized infrastructure. NFV decouples the network functions, such as firewalls, intrusion-detection system, load balancers, Network Address Translation (NAT), and caching, from proprietary dedicated hardware appliances. It is designed to deploy networking components to support a virtualized infrastructure, including virtual machines, servers, storage, etc. The basic idea was proposed in 2012 by a group from the European Telecommunications Standards Institute (ETSI) [16].

Software-Defined Networking (SDN) is an approach to network control and management, which allows administrators to manage network services by an abstraction of lower network protocol functions. This is done by decoupling the control plane, that takes decisions about forwarding traffic through the network infrastructure, from data plane, the users traffic itself. The objective is simplify the network control in high speed traffic. SDN requires some mechanism (protocol) for the control plane to communicate with the devices. The most used SDN protocol is the OpenFlow [4].

NFV is a complementary approach to SDN. While both intend to manage networks, they rely on different point of view. While SDN separates the control and forwarding planes to offer a centralized view of the network, NFV focuses on implement network services on a unique network configuration infrastructure, e.g., based on SDN. There are inherent benefits in leveraging SDN to implement a NFV infrastructure. Basically, when we were looking the management and orchestration of VNF features in a multi vendor SDN infrastructure.

A. OpenFlow Architecture

The OpenFlow architecture has several components: a remote control process, usually called ”controller”, devices (switch) and the protocol [4]. The OpenFlow approach considers a centralized controller that configures all devices. Devices should be kept simple in order to reach better forward performance, and the network control is done by the controller.

The controller is the centralized entity of an OpenFlow network. It maintains all devices, topology information and monitors the network status of the entire Openflow network. The OpenFlow device is any OpenFlow-enabled device in a network such as a switch, router or access point. Exploiting the fact the all modern Ethernet switches and routers contain flow-tables, each device maintains a flow-table that indicates the processing applied to any packet of a certain flow. The OpenFlow Protocol works as an interface between the controller and the switches setting up the flow-table. The protocol should use a secure channel based on Transport Layer Security (TLS).

The controller updates the *Flow Table* by adding and removing Flow Entries using the OpenFlow Protocol. The Flow Table is a database that contains Flow Entries associated with actions to command the switch to apply some actions on a certain flow. Some possible actions are: forward, drop and encapsulate. Each OpenFlow device has a Flow Table with flow entries. A Flow Entry has three parts: header field, counters, and action. The Header field is used to define the matching conditions to an exact flow; Counters are used to count the rule occurrence (for management purposes), and Action defines the actions to be applied to a specific flow. When a packet arrives to the OpenFlow Switch, it is matched against Flow Entries in the Flow Table. The Action will be triggered if the Header Field is matched and then, the Counter is updated. If the packet does not match any entry in the Flow Table, it will be sent to the Controller over a secure channel.

IV. ContentSDN: Transparent Content Caching on SDN Network

The ContentSDN architecture is based on the principle of a centralized control plane, based on SDN, responsible for forwarding packets to a proper device, managing cacheable requests that follow the client-server model. The ContentSDN architecture is shown in Figure 1 and it is based on two components: the ContentSDN Proxy and ContentSDN Cache. The idea is to map the Traffic of Interest (TOI) to the content coordinator, the ContentSDN Proxy, that forwards requests to some ContentSDN Cache instance based on the content's shortest path and/or a set of customizable policies. The cache component is responsible for store and serve cacheable responses of requests triggered by any user within the network.

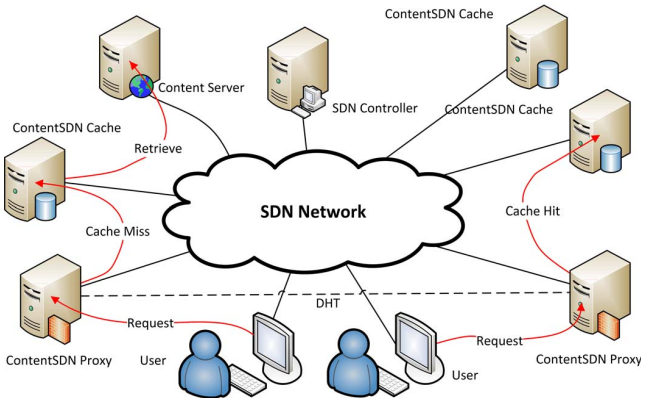


Fig. 1. The ContentSDN Architecture

It is possible to deploy multiple instances of the ContentSDN Proxy over the network, where each instance coordinates an exclusive ContentSDN Cache island. The SDN controller can distribute content requests from clients to the closest proxy on the network, minimizing congestion. Each proxy instance is part of a Distributed Hash Table (DHT) overlay network that stores the content-cache instance mappings of the entire network. The ContentSDN Proxy forwards content requests by looking if the content has been already cached at some ContentSDN Cache instance. If the content has been cached at some cache instance (cache hit) the proxy will forward the request to that instance. Otherwise (cache miss), the proxy will forward the request to the closest cache,

which will request the content from the server, caching the received content and serving the user request.

ContentSDN has been designed considering high scalability and manageability at runtime. Any ContentSDN Proxy or ContentSDN Cache can be added or removed at runtime, providing an elastic solution that is employed over a VNF. This directive allows several advantages, since it is well known that the network workload can be variable during the day, turning the use of NFV a good choice for real-time network orchestration.

A. ContentSDN Forwarding

ContentSDN depends on the SDN infrastructure to perform content dispatching. Such dependency comes from the need to forward the requests to cacheable content to a proxy instance on the network in a transparent way. This is an efficient task when using SDN, e.g. OpenFlow protocol, since it is possible to create flows on the switches that maps either TCP or UDP ports of specific applications, without changing the packet header.

When installing flows on the switches, there are two approaches for creating flows: reactive or proactive [17]. The proactive approach aims to install the necessary flows from the clients to the closest ContentSDN instances before a request is made to a server, while the reactive approach aims to install such flows only when a request is made by some client. The proactive approach has the advantage of avoiding the switches in the request's network path to ask the controller for new flows, avoiding extra latency when forwarding requests. The main disadvantage is that the path can easily become congested, since all packets are routed through the same network segment. When using the reactive approach the user suffers of some impact on latency, since the switches must ask the controller for new flows, but in this case, it is possible to use some load balancing techniques that are capable of using alternate paths, minimizing network congestion. Independently of which approach is used, ContentSDN always maps users' requests to reach the closest proxy instance of the network in order to minimize latency and link usage.

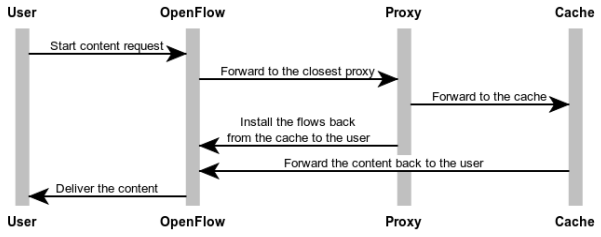


Fig. 2. ContentSDN Forwarding Steps

Once a request reaches a proxy, it performs a deep inspection of the request in order to decide to which cache the request will be forwarded (see next subsection). Of course, the packet payload must be readable by the proxy, which makes the inspection of HTTPS packets an hard and illegal task, for instance. At this moment, the proxy sends a command to the controller, through a Representational State Transfer (REST) API, in order to install the necessary flows to the cache send the response directly to the user. This process is illustrated in Figure 2. This approach guarantees that the cache handling is completely transparent for the clients, so there is no need to change any client application implementation details in order to be supported by ContentSDN. In fact, when the underlying transport protocol is TCP, both proxy and cache nodes must be aware of details of connection handling, since the connection state is exchanged between proxy and cache [18].

B. ContentSDN Proxy Design

The ContentSDN Proxy is the most important module on ContentSDN architecture. This component is responsible for handling the clients' requests in order to deliver the requested content to the closest available cache. The basic idea behind it is to forward the request to the cache node that contains a copy of the requested content, if any. To do so, the proxy uses an index that maps the content's unique name to a cache node. When there is no available copy in the network, the proxy forwards the request to the closest cache node from the client, making that node the *candidate* for holding a copy of the requested content. If there are multiple cache nodes with the same distance from the user, the proxy performs a round-robin [19] in order to load balance the requests.

Since it can exist many ContentSDN Proxies spread over the network, each proxy instance shares a DHT with other proxies. This DHT is used as the content index, which makes these proxies to be part of a Distributed Forwarding Unit (DFU). Despite there are multiple proxy instances over the network, it is important to notice that the users' requests will always be handled by the same proxy node (the closest one).

The proxy and the cache instances maintain an overlay network that is used for exchanging cache state and commands. Every time a proxy decides to forward a request to a cache, it does this task by sending a command to the chosen cache informing that it must serve a given request from a given client. At this time, the proxy listens for a *possible* acknowledge that the related content was successfully cached, which allows the proxy to update its index in order to forward further requests to the same content to that same cache instance. This acknowledgement may also inform the content's Time to Live (TTL), defining the maximum amount of time that content will be cached by ContentSDN. When a TTL expires, the index entry that maps to the related content must be automatically removed from the DHT.

It is important to notice that if a requested content has already been stored in any cache, any proxy will forward the request to that cache instance, even if that instance is not being managed by that proxy. For instance, suppose that the proxy A manages the cache A while proxy B manages cache B. Cache A is empty and cache B is holding a copy of content C. Now, a client close to proxy A send a request for content C, which is handled by proxy A due to the proximity. In this scenario, proxy A would forward the client's request to cache

B, because the DHT index informed to proxy A that a copy of C is on cache B. See Figure 3.

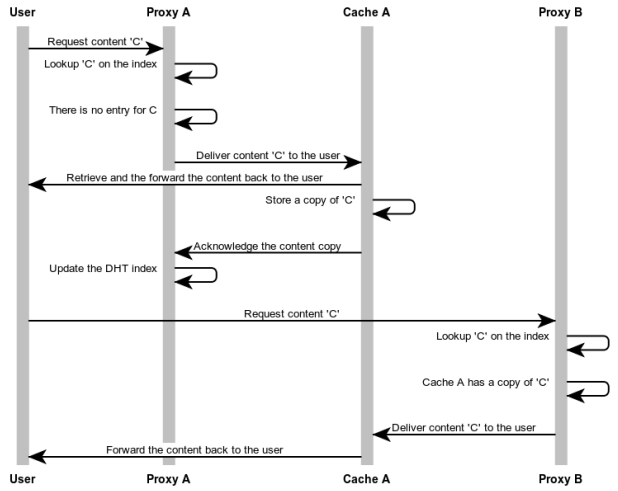


Fig. 3. ContentSDN Proxy Index and Forwarding Steps

C. ContentSDN Cache Design

The ContentSDN Cache is the component responsible to serve clients requests that were forwarded by the ContentSDN Proxy. The ContentSDN Cache sequence diagram is shown in Figure 4. When a request command reaches a cache instance, the cache check if the content has already been stored in its memory, by using a Hash Table for fast lookup. If the content can be locally resolved, then the cache sends the requested content directly back to the client through the previously configured OpenFlow path. If not, then the cache will forward the client's request to the target server informed within the request, which would be the normal behavior if there was no installed ContentSDN on the network. When the target server sends back the response, the cache immediately sends the response to the client and performs a deep packet inspection in order to infer if such response is cacheable or not. The cacheability inferring mechanism will depend exclusively on the nature of the protocol used by the client and server.

If the content is cacheable, and it has just been fetched from the server, the cache acknowledges as soon as possible the proxy that the current cache instance now holds a copy of the content, possibly with a TTL. Again, the inferring mechanism of the TTL value depends on the used protocol. Contents that can be held indefinitely on the cache do not have any TTL associated with it. If the content is not cacheable, the response is simply sent to the client, and no caching is performed on that content.

D. ContentSDN Cache Management

Each ContentSDN Cache instance is specialized in caching content. To this extent, a cache instance has a layer that provides the storage capability. ContentSDN Cache instances were designed in such a way that the storage layer of an instance does not interfere with the other instances' layers,

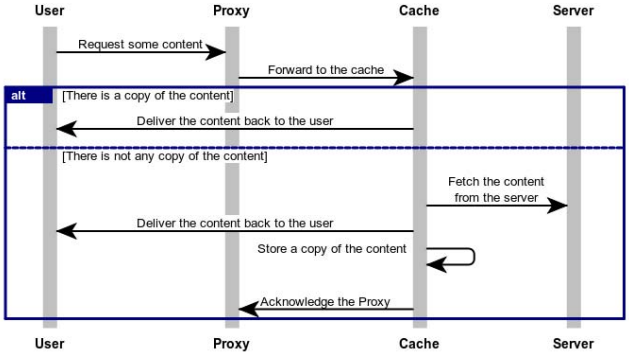


Fig. 4. ContentSDN Cache Steps

allowing several deployment scenarios, each one with its own specificities.

An important aspect must be covered by ContentSDN cache: the decision about where to store the content. It is possible to use an in-memory approach or to use the local disks. The capacity of the memory is much less than the disk capacity, but the first provides faster access than the second. Such decision depends on the policy that the network administrator wants to take, since the use of an in-memory approach leads to the caching of data with high volatility at high workloads. Another aspect that must be chosen carefully is what eviction policy will be used at that instance. Cache eviction policies are well discussed in [20] and [21].

It is possible to use a hybrid approach when dealing with the decision of store content on memory or disk. A hybrid approach would consist in the use of a L1 and L2 schema, where the most recent (or frequent) data stays in the L1 (memory) while the rest stays in the L2 (disk). If a content is in the L2 and is accessed frequently, it the managing algorithm could move the data to the L1, providing faster access.

The fact that the ContentSDN Cache instance is a VNF gives great flexibility. Once the cache is a virtualized machine, it is possible to adjust the storage capacity at runtime. If the cache becomes saturated and the eviction rate increases below some threshold, the NFV Orchestrator (NFVO) can automatically attach a new virtualized Solid-State Drive (SSD) or even increase the available memory and change the eviction policy.

E. ContentSDN Policy Management

Several design strategies were used in the ContentSDN project, addressing efficient content delivery and also considering the possibility of dealing with complex real-world policies. Policy management is an important feature to consider, since there is no solution that fits all kinds of network traffic and business needs.

Policy management is performed in ContentSDN Proxy component. Since the proxy is a decision-maker responsible for a set of cache instances, this is a natural and efficient choice. Considering that all requests of a subset of network nodes will pass through the proxy, it is possible to extend the

own deep inspection task in order to be aware of manageable aspects of data. The current stage of ContentSDN development has addressed five caching policies in order to adapt the solution to several common workloads:

1) Cache everything
2) Cache only content whose name matches any of a given regular expressions set
3) Cache only content whose size matches some criteria
4) Cache only content served by some set of target domains
5) Cache only content of a given type (text, video, etc)

In order to process these policies, it is used a processing pipeline that allows the use of multiple policies from the most restrictive to less restrictive one, serving as a set of policy filters. When a request reaches the proxy, it will pass the deep inspection phase and then run the policy pipeline, expecting the allowance or denial of caching the requested content at the closest cache instance from the user. If a request is denied by the defined policy set, the forward command sent to the cache instance will have a flag indicating that this content cannot be cached by that instance, which in this case imply that no cache acknowledgement will be expected.

Note that some requests are not able to enter the pipeline processing until it has been served by the origin server. For instance, in some HTTP requests, the requester allows receipt of both text and video responses, but only the actual response will tell the actual type, through the *ContentType* header. The content's size is also another good example of this situation. It is not possible to predict how many bytes the content will have until the response has been received. To this extent, the same processing pipeline can be run at the cache instances, but only to this especial case and if the flag set on the proxy's command allows caching, avoiding the cache to override any previous decision made by the proxy.

V. Evaluation

The current stage of the ContentSDN development project supports the caching of requests made through HTTP. To this extent, several experiments were performed in order to evaluate the effectiveness of the proposal and give some insights about further deployment in production networks.

The evaluation environment was based on a virtualized network using Mininet [22]. The Mininet system permits the specification of a network interconnecting virtualized devices. Each network device, hosts, switches and controller are virtualized and communicate via Mininet. A Python script is used to create the topology in Mininet, and the traffic flow control is made by the OpenFlow controller. Therefore, the test environment implements and performs the actual protocol stacks that communicate with each other virtually. The Mininet environment allows the execution of real protocols in a virtual network. The possibility to set link bandwidth and delay in Mininet became the experiment similar to an actual scenario. The chosen OpenFlow controller was the Floodlight [23], due to its simplicity and development flexibility.

Once there are several applications that ran over HTTP (from plain text web sites to video streaming), the experiments were made with a couple of content sizes, varying the HTTP responses' sizes from 10 to 3200 kilobytes. Augustin et al. [24] shows many statistics about the bandwidth usage of some Web 2.0 applications, from email to on-demand video, which shows that this experimenting approach can address that range of applications.

The first experiment's objective was to analyses how the deployment of ContentSDN would impact the users Quality of Experience (QoE) and the overall network load. Figure 5 shows the topology built for testing (1) how long a client would wait to retrieve contents from a server and (2) how many packets and how many bytes are exchanged within the OpenFlow network to allow such a retrieval process by all clients. Naturally, it is expected that client's delay, and the amount packets/bytes will be reduced when ContentSDN is deployed in the network.

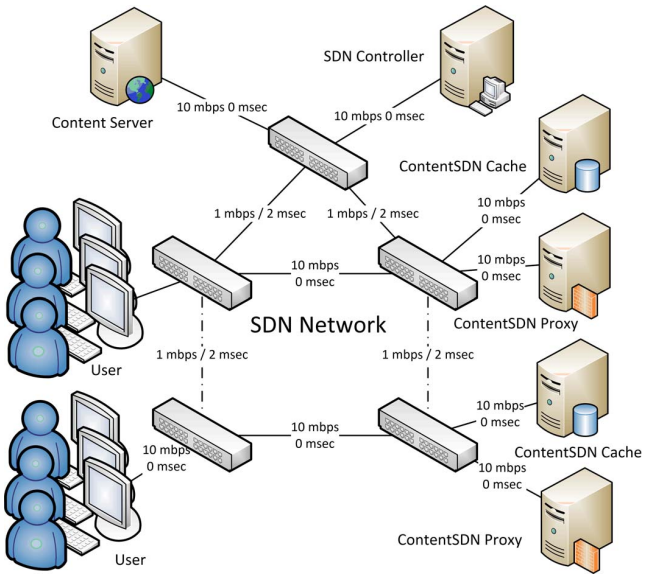
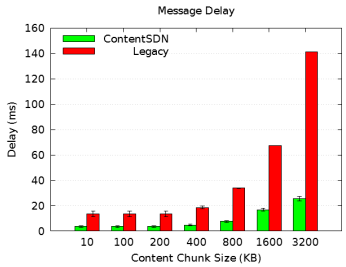


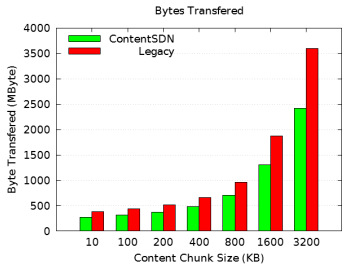
Fig. 5. Evaluation Topology

Each client was implemented in Java and has used the Apache's HTTP Client in order to perform HTTP requests. Each client is single-threaded, and all clients send requests concurrently. The cache hit rate was set to 70%. The measurement of the perceived clients' delay is calculated within each client, from the moment it sends a HTTP request to the moment the response is received by that same client. The amount of bytes and packets are retrieved after each experiment round, from the OpenFlow controller, which queries each OpenFlow Switch port counter value through *StatsRequest* messages [25].

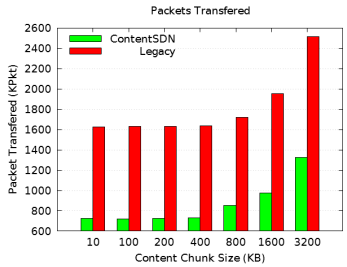
The second experiment tries to give a view of how the content server throughput is affected when the underlying network is served by ContentSDN. The clients used were the same from the first experiment, and the responses' sizes have varied from 10 to 3200 kilobytes. The throughput measures were collected from the server's network interface. It is expected that the server throughput decreases as the hit rate increases.



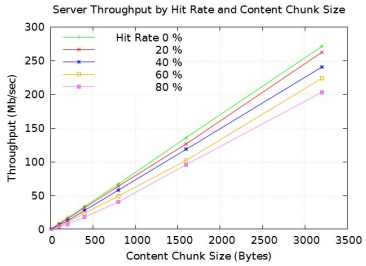
(a) Client's perceived delay with responses at different sizes, with hit rate of 70%. Results shown with 95% confidence interval.



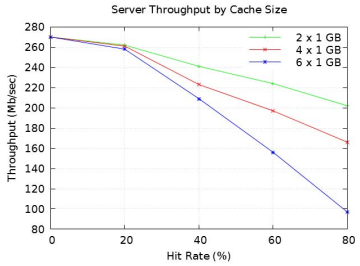
(b) Sum of transferred bytes at each network interface of all switches within the network topology.



(c) Sum of transferred packets at each network interface of all switches within the network topology.



(d) Server throughput at different cache hit rates and responses' sizes, with 2 caches storing 1GB of data.



(e) Server throughput at different cache hit rates and cache capacities, with responses of 3200 kilobytes.

Fig. 6. Experiment results

The third experiment is a derivation of the second. The objective is to check what happens to the server throughput when the hit rate varies, and the available cache storage capacity varies. It is expected that the throughput decreases as the hit rate and the storage capacity increases, since bigger storages provide fewer cache evictions.

Experiment 1 has used caches with 1GB of capacity. Experiment 2 has used two caches of 1GB of capacity. Experiment 3 varies the capacities from 2GB to 6GB. The caches were configured to use the Least Frequently Used (LFU) cache eviction policy and an in-memory approach in all experiments.

VI. Results

Figure 6a shows the clients average delay when requesting contents from a server. It is possible to notice that ContentSDN is effective on improving the users' QoE. It was possible to check an improvement on the delay of nearly 75%, independently of the content's size. The result for 3200 kilobytes contents shows an improvement of almost 80%, which we believe that is a consequence of less network congestion when using ContentSDN. Furthermore, it is possible to observe that the confidence interval error decreases as the responses' sizes increases when there is no caching into the network. This effect could be explained by the network congestion increase and the fairness of TCP, which is used as transport for HTTP. When there is caching, this behavior could not be reproduced, which indicates that the current implementation of ContentSDN needs to be improved to follow the same fairness pattern, even though there is a significant improvement on users' QoE.

Figures 6b and 6c shows the amount of bytes and packets that were transferred after the execution of experiment 1. As expected, both metrics were reduced when using ContentSDN, which can be explained by the reduced proximity of the content from the requesting clients. It can also explain the improvement in the users QoE. It is also evident that ContentSDN improves the efficient use of network resources, helping the networks to be more effective and reducing operational costs.

Figure 6d and 6e shows what happens to the network throughput at the target server when the underlying network is served by ContentSDN at different hit rates, content sizes and cache capacities. As expected, when the hit rate increases and the cache capacity increases, the throughput at the server decreases, indicating that ContentSDN decreases the number of requests that reach the destination server. In Figure 6e it is easy to observe that the overall cache capacity is an important variable to be considered when deploying ContentSDN or any caching solution of the network, since less capacity incurs in more evictions and more accesses to the destination servers. At a hit rate of 80%, for instance, the server's throughput decreases 52% when the capacity is increased just 4GB. Therefore, when the cache capacity is fully loaded, the eviction rate starts to increase together with the server's throughput, indicating that small caches may be useless for high workloads. These results are also good indicators that real-world deployments should use a policy that determines only specific types of content to be cached, avoiding the waste of important

cache capacity.

An interested result is that ContentSDN always follows the same improvement patterns on user metrics (QoE) and network metrics (server throughput and transferred bytes) independently of the content's sizes. As ContentSDN stores the content near to any client at high speeds, the performance result against different content sizes is negligible, i.e., the transmission time of a small or a big content is similar.

VII. Conclusion and Future Work

This paper has shown the ContentSDN, a content-based transparent caching architecture over SDN. The work showed that ContentSDN architecture provides highly available and scalable caching of named content for SDN networks, independently of specific underlying transport and application protocols. The work has also shown that the caching mechanisms are driven by business policies and can be deployed in networks of any nature, using a NFV approach and a microservice-based architecture. Moreover, the current implementation of ContentSDN supports the HTTP protocol, which is the main protocol used for content delivery over the Internet at the moment. The experimental evaluation has shown that ContentSDN fulfills its role, helping to improve the Quality of Experience and multiple network metrics.

As a future work, we intend to improve the naming scheme. In a content-based network, a same content may have different names, wasting memory and storage resources in caches due to duplicate files. A hash mechanism can give indications that what requests refers to identical content, to avoid unnecessary copies.

The ContentSDN considers that all content is public and there is no restriction in distribution, making it unusable to deliver paid content, such as some VoD. The content should be encrypted and the VoD provider can give a temporary key to the subscriber in order to decrypt the content. After this period, the key is changed and the subscriber needs to renew the key.

References

[1] J. S. Turner and D. E. Taylor, "Diversifying the internet," in *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, vol. 2. IEEE, 2005, pp. 760–766.

[2] Cisco, "Cisco visual networking index: Forecast and methodology, 2014-2019 white paper," Last accessed, Sep 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[3] J. F. Gantz and D. Reinsel, "The expanding digital universe: A forecast of worldwide information growth through 2010." IDC, 2007.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "Devoflow: cost-effective flow management for high performance enterprise networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets '10. New York, NY, USA: ACM, 2010, pp. 1:1–1:6.

[6] R. Guerzoni *et al.*, "Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper," in *SDN and OpenFlow World Congress*, 2012.

[7] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, July 2012.

[8] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named data networking project," Xerox Palo Alto Research Center-PARC, Tech. Rep., 2010.

[9] V. Jacobson, M. Mosko, D. Smetters, and J. Garcia-Luna-Aceves, "Content-centric networking," *Whitepaper, Palo Alto Research Center*, pp. 2–4, 2007.

[10] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, "Supporting information-centric functionality in software defined networks," in *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 6645–6650.

[11] A. Ooka, S. Ata, T. Koide, H. Shimonishi, and M. Murata, "Openflow-based content-centric networking architecture and router implementation," in *Future Network and Mobile Summit (FutureNetworkSummit 2013)*. IEEE, 2013, pp. 1–10.

[12] X. N. Nguyen, D. Saucez, and T. Turletti, "Efficient caching in content-centric networks using openflow," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 1–2.

[13] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, and L. Tassiulas, "Pursuing a software defined information-centric network," in *Software Defined Networking (EWSDN), 2012 European Workshop on*. IEEE, 2012, pp. 103–108.

[14] A. Chanda and C. Westphal, "A content management layer for software-defined information centric networks," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 47–48.

[15] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service: leveraging sdn to efficiently and transparently support video-on-demand on the last mile," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*. IEEE, 2014, pp. 1–9.

[16] M. Ciosi *et al.*, "Network functions virtualization," in *SDN and OpenFlow World Congress*, 2012.

[17] M. P. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013, pp. 1009–1016.

[18] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory tcp: Connection migration for service continuity in the internet," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002, pp. 469–470.

[19] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 3, pp. 375–385, 1996.

[20] A. Balamash and M. Krunz, "An overview of web caching replacement algorithms," *Communications Surveys & Tutorials, IEEE*, vol. 6, no. 2, pp. 44–56, 2004.

[21] J. Wang, "A survey of web caching schemes for the internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36–46, 1999.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets '10. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.

[23] D. Erickson, "Floodlight Java based OpenFlow Controller," Last accessed, Aug 2012. [Online]. Available: http://floodlight.openflowhub.org/

[24] B. Augustin and A. Mellouk, "On traffic patterns of http applications," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec 2011, pp. 1–6.

[25] N. L. Van Adrichem, C. Doerr, F. Kuipers *et al.*, "Opennetmon: Network monitoring in openflow software-defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–8.