SDCCN: A Novel Software Defined Content-Centric Networking Approach

Sergio Charpinel, Celso Alberto Saibel Santos
Magnos Martinello and Rodolfo Villaca
Federal University of Espirito Santo (UFES)
Vitoria/ES, Brazil
{scjunior, saibel, magnos}@inf.ufes.br, rodolfo.villaca@ufes.br

Alex Borges Vieira
Federal University of Juiz de Fora (UFJF)
Juiz de Fora/MG, Brazil
alex.borges@ufjf.edu.br

***Abstract*—Content Centric Networking (CCN) represents an important change in the current operation of the Internet, prioritizing content over the communication between end nodes. Routers play an essential role in CCN, since they receive the requests for a given content and provide content caching for the most popular ones. They have their own forwarding strategies and caching policies for the most popular contents. Despite the number of works on this field, experimental evaluation of different forwarding algorithms and caching policies yet demands a huge effort in routers programming. In this paper we propose SDCCN, a SDN approach to CCN that provides programmable forwarding strategy and caching policies. SDCCN allows fast prototyping and experimentation in CCN. Proofs of concept were performed to demonstrate the programmability of the cache replacement algorithms and the Strategy Layer. Experimental results, obtained through implementation in the Mininet environment, are presented and evaluated.**

I. Introduction

The engineering principles and architecture of today's Internet were created in the 1960s and '70s to solve resources sharing problems. These resources were shared among expansive and scarce devices. The resulting communication model was a communication in pairs [1]. Despite patches made over decades, the base of the communication model has not practically changed.

One of the perspectives of Content Delivery Networks (CDNs) is that these networks will carry 60% of Internet's traffic in 2019 and video traffic will correspond to more than 80% of the Internet traffic in 2019 [2]. The report points out that the user is interested in the content ("what") while today's communication is still based on location ("where").

In this context, Content-Centric Networking (CCN) emerges as a new abstraction for the current communication, providing addressing, routing and security based on the content and native cache provided by the network.

Although CCN is a "clean-slate" architecture, it inherits from TCP/IP the lack of flexibility and the difficulty of experimentation in production networks. The cache replacement algorithm and the Strategy layer are not flexible enough to be modified at runtime or even allow the execution of different strategies for multiple nodes without manually modifying CCN routers code. The protocols are also "hardcoded" in the routers, requiring considerably more effort to evolve the architecture. And finally, router management and configuration in a manual and expensive task for develloppers.

This paper presents a novel approach using Software Defined Networks aiming to add flexibility to CCN routers. Our SDCCN (Software Defined Content Centric Network) approach has three major contributions: (i) it supports programmable forwarding and caching in CCN; (ii) eliminates the need for mappings between content names and identifiers, by natively processing content names with varying sizes through the POF (Protocol Oblivious Forwarding) protocol and (iii) provides a simulated prototyping environment based on Mininet for carrying out experiments and innovative solutions in CCN.

The rest of this paper is divided in the following way: Section II discusses and compares some related work to the proposed SDCCN. Section III shows the architecture of the SDCCN solution. In Section IV implementation details are shown. Section V introduces the experimental results obtained in SDCCN scenarios. Finally, Section VI concludes the paper and presents proposals for future work.

II. Related Work

COntent NETwork (CONET) [3] extends the CCN approach in various aspects, such as integration with IP, routing scalability, use of new mechanisms for transport, inter-domain routing and integration with the SDN architecture using OpenFlow [4]. To work around the problem of processing variable length content names, CONET maps content name to an identifier (TAG) and uses this TAG for routing content packets. This work does not include cache programmability and content name mapping to TAG breaks the aggregation of hierarquical names in routing.

Nguyen et al. [5] proposes an SDN architecture for CCN for implementation of "off-path caching", which consists on caching only the most popular contents. The control plane gets the most popular contents, defines in which nodes they should be cached and install forwarding rules. SDCCN provides a more flexible approach for implementing "off-path caching".

Vahlenkamp et al. [6] proposes an SDN solution for gradual integration of Information-Centric Networking in IP networks. They use a network address publicly routable per domain,

called ICNP, for each SDN node and an ICN identifier that will be used in the UDP or TCP ports fields. This solution also envolves content name mapping, bringing all of the mentioned drawbacks. Cache programmability is not discussed.

Our approach mainly differs from the previous works because it simultaneously: (i) eliminates the need for mappings between content names and identifiers, processing natively content names with varying sizes; (ii) provides programmable cache replacement algorithms; (iii) allows proactively installation of content in caches; (iv) allows implementation of CCN protocols in the network elements in an agnostic way; (v) provides a simulated environment based on Mininet for prototyping programmable CCN in an easy and scalable way.

III. SDCCN Architecture

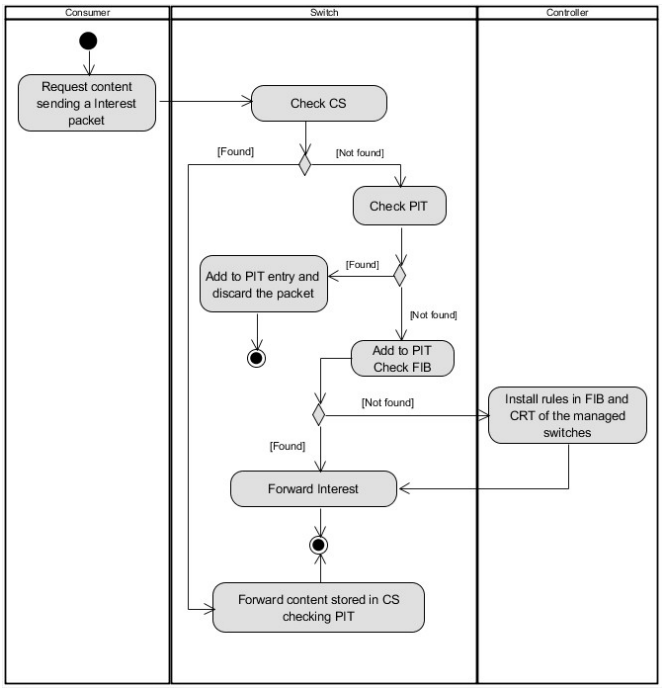
The SDCCN architecture is composed by users (producers and consumers of content), CCN switches (we will call they routers as well) and a CCN controller (logically centralized). There are two types of packets: Interest packets, which are used to request content, and Content packets, used to provide content. Consumers request content sending Interest packets on the network. Switches forward packets and keep in cache the most popular content. Controllers manage the switches programming the forwarding of the Interest packets and the Content Storage. Content providers answer Interest requests sending Content packets and also disclose content names prefixes they produce contents on.

A SDCCN switch is composed by: (i) a FIB (Forwarding Information Base) to store forwarding rules related to the Interest packet forwarding; (ii) a Cache Rules Table (CRT) for storing cache rules that inform what content should be stored in cache; (iii) a Content Store (CS) for storing content; and (iv) a PIT (Pending Interest Table) used to group Interest requests that have not received a corresponding content yet and to forward Content packets.

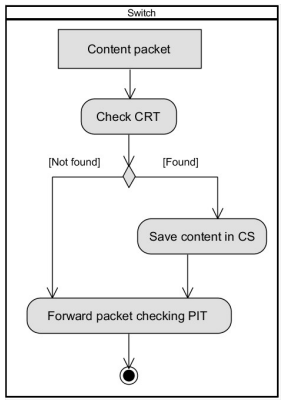
The controller uses well known SDN protocols and is able to modify SDCCN tables, request information about them, modify cache replacement policies on the fly, allocate different caching policies to different nodes and install more efficient forwarding strategies, since it has a complete view of network.

A. Forwarding

The activity diagram shown in Figure 1(a, b) gives an overview of the packet forwarding in SDCCN. A host requests content sending a Interest packet for the content on the network. When a switch receives this packet, it first checks whether the requested content is stored in its CS. If the switch CS has the content, it returns the corresponding Content packet (data) to the switch port that the Interest packet was received. Otherwise, it checks if there is any pending Interest in its PIT and, if so, it adds the interface id to the correponding PIT entry and discards the packet. Otherwise, it forwards the Interest packet to the next switch consulting its FIB and stores a new entry in its PIT containing the content name and the interface id from where the pending Interest was received. If the switch does not find a matching entry in the FIB, it queries



(a) Interest forwarding.



(b) Content forwarding.

Fig. 1. Activity diagram of packet forwarding in a SDCCN network.

the controller which determines how the forwarding should be done and installs forwarding rules in the FIB of the switches.

Once the Interest packet has reached a switch with a valid content in cache, or reaches the producer, a corresponding Content packet is sent back. The content packet is forwarded following PIT entries and therefore it follows the same route of the corresponding Interest packet. Switches also check its CRT to verify whether this content must be stored in the CS.

The Strategy layer is responsible for defining Interest packet routing based on optimized choices, such as sending packets to different ports simultaneously in order to achieve shorter response time. Moreover, it is also responsible for resending timed out Interest packets using different strategies. In the original proposal of the CCN [1], it is recognized that there is not a routing strategy that is best for all cases. Thus, the original intention was to add in each entry of the FIB a

program written in a specialized language in packet forwarding decisions which would determine how the routing would be done. The Strategy layer definition brings the forwarding programmability concept and SDCCN uses an already known and SDN protocol to implement it, providing a more reliable and stable solution.

B. Cache

Several studies have been made to find the best solution for cache in CCN ([7], [8]). Many of these studies propose the use of different replacement cache policies for nodes located on the edges and nodes located in the core of the network. However, in CCN networks changes in replacement policies are costly, once they need manual firmware updates. The SDN approach facilitates this implementation because caching replacement algorithms are defined in the control plane. Also, the controller has an overview of the network topology and content distribution can provide more customized and dynamic solutions. Moving cache decisions to the control plane through a standardized interface facilitates experimentation and comparison of different cache policies aiming to improve network performance in content retrieval.

A SDCCN switch will only store a content in its CS if it has a rule in its CRT. CRT also supports the use of wildcard rules. When the CS becomes full or reaches a configurable threshold, the switch notifies the controller by sending a control message. The controller may remove content stored in the CS following any custom logic, allowing the definition of cache replacement algorithms in the control plane.

The controller can also proactively install contents in the CS, that is, without even receiving a corresponding Interest packet. This approach is very useful in some situations, such as when we can predict certain content will have great chances of being asked by multiple nodes in a network, like the release of a blockbuster content, or in face of user mobility when transmitting a twitcasting in a popular event.

IV. SDCCN Implementation

Data plane and Control plane communicate through the Protocol-Oblivious Forwarding (POF) [9] protocol. POF allows routing content names of variable sizes. Moreover, the data plane does not need to recognize any frame format. Processing flows is similar to OpenFlow, however in OpenFlow the search is done using known protocols fields. In POF, the search is made comparing bytes of a field, defined by the tuple {offset, size}, with bytes presented in tables rules. POF also supports all messages of the OpenFlow protocol.

By using POF, CCN messages can be implemented in a agnostic way in network elements. In other words, you can change the format of the CCN packets without changing the code of CCN switches. Messages such as neighbor discovery, repository synchronization, etc., may have its format changed without modifying the data plane.

In the data plane, POF switch has been extended to support SDCCN messages. In addition to the existing FIB, the CS, CRT and PIT have been added to the POF implementation. We further discuss the current format of these tables in Sections IV-A and IV-B.

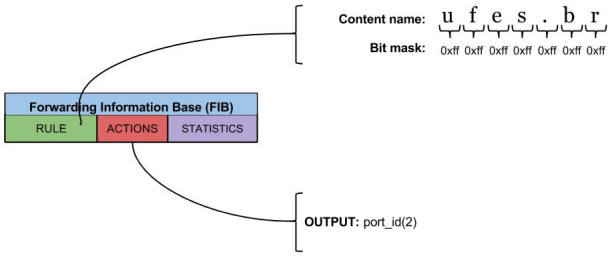
We developed an environment for rapid prototyping and experimentation in CCN based on Mininet [10]. It is possible to create CCN nodes running the official CCNx project [11] implementation. The current implementation supports all routing and cache messages described in Sections IV-A and IV-B.

A. Forwarding

A set of messages and actions of POF protocol are extended and used to provide programmable routing. A new message was added to POF for implementation of Interest retransmissing, managed by the Strategy layer. The message INTEREST_INFO is used by switches to signal to the controller that an Interest request was timed out.is detailed below:

The current PIT format can be shown in Figure 2(a). Source ports of pending Interests are store in the “ports” field as well as its number in “nports” field. The “created_date” field is used for the pending Interest timeout.

(a) PIT format.



(b) FIB format.

Fig. 2. PIT and FIB format.

POF flow table was used for FIB implementation. As shown in Figure 2(b), each FIB entry consists of rules, actions and statistics data like bytes and packets counters. The rule has a content name and a bit mask that will indicate whether the corresponding bit of the content name will be considered during forwarding lookup (byte “0xff”) or not (byte “0x00”). Each rule implies in one or more actions, such as packet output to some port or write bytes to packets.

B. Cache

New messages have been added to the CCNx implementation to support cache programmability. New messages are:

- CACHE_MOD: Message used to modify the CRT (add, remove and modify cache rules).
- CS_MOD: Message used to modify the CS (add, remove and modify content in cache).
- CACHE_FULL: Message used by the switch to indicate to the controller that a CS is full or nearly full.
- CACHE_INFO: Message used by the controller to query CRT data.

- CS_INFO: Message used by the controller to query CS data.

The current formats of CS and CRT are illustrated in Figure 3. In CS, "created_date" and "last_update" fields are used for cache replacement algorithms and can be retrieved using CS_INFO message. In CRT, "cs_mod" field is used for proactive installation of content in CS, as following: upon receiving a CS_MOD message with ADD command, the switch sends an Interest packet to the content specified in the message. To store the content in the CS, the switch creates a CRT entry with "cs_mod" equals to 1. When the switch receives the corresponding Content packet and checks CRT to verify if the content should be saved, it will find an entry with "cs_mod" set to 1 and will remove this entry after saving the content in CS, avoiding saving undesirable contents in future.

Content Store (CS)					
content_name	index	created_date	last_update	content_size	content

(a) CS format.

Cache Rules Table (CRT)					
content_name	index	exact	idle_timeout	hard_timeout	cs_mod

(b) CRT format.

Fig. 3. Current format of the CS and CRT.

V. Proof of Concept and Experimental results

The proof of concept is structured in two parts. The first part demonstrates the programmability of packet forwarding and caching.

In the first experiment, two different cache replacement policies are implemented and programmed directly in the control plane. Then, the results of cache hit ratio of both are compared in different situations with different content popularity characteristics. The second experiment shows the Estrategy Layer operation, exemplified by the use of dynamic forwarding policies.

The second part is dedicated to experimental results of the implemented prototype, with focus on the exploration of different scenarios involving the proactive installation of contents in caches, cache aggregation and load balancing.

A. Proof of Concept development

1) Cache replacement policies implementation: In SDCCN architecture, cache replacement policies are defined in the control plane. This enables the implementation of customized or specific policies for certain scenarios, such as the definition of different policies for nodes located at different areas of the network (core and edge for example) and the change in real time the cache replacement policies. Moreover, the controller has an overview of the network and of contents cached in its managed switches. It may use these informations to create more advantageous cache replacement policies.

The cache replacement policy decides which content will be removed from CS. So the controller needs to retrieve information about cached contents, run some algorithm that decides which contents shall be removed and remove these contents. When a Content arrives on the switch, it checks if its CS utilization is above the configured limit. If so, sends a CS_FULL message to the controller, signaling the high utilization. The controller requests CS data by sending a CS_INFO message. When it gets the reply, it runs a cache replacement algorithm and removes contents using the CS_MOD message.

The experiment aims to evaluate implementation of cache replacement policies in the control plane. The topology is composed by two hosts (`h1`, `h2`) and one SDCCN switch. Links between the SDCCN switch and hosts were defined with delay of 10ms. Host `h1` runs ccnpingserver application and host `h2` makes ccnping [12] requests in CCN network. Due to link delays, a ccnping request passing through two hosts must have a RTT around 40ms. If the switch had the content of the corresponding ccnping request stored in its CS, then the RTT must be around 20ms.

The CS was limited to 50, 100 and 200 entries and the limit for sending warning alerts (OFPCFAC_WARN) was set to 5 entries. Forwarding rules have been added to `h1` FIB for forwarding Interests with the prefix "ccnx:/UFES" to `h2`. Caching rules have been added to `h2`'s CRT for storing any content with the prefix "ccnx:/UFES", which was also used as prefix for ccnping. The names of the generated ccnping content followed the pattern "ccnx:/UFES/[N]", where [N] is a positive integer randomly generated. This number was generated using a Zipf-like distribution with α equal to 0.6, 0.7, 0.8, 0.9 and 1.0 (as suggested in [8]).

In the beginning of the experiment, the CS is empty and the CRT has an entry to store ccnping contents. So each content passing through the switch is stored in its CS.

For each cache replacement algorithm and each CS limit configuration, 1000 requests were generated. The experiment was repeated 5 times for average and confidence interval (confidence level of 95%) calculation. Table I shows the cache hit rate for each value of α for FIFO and LRU algorithms respectively. We can see that the LRU algorithm achieved a slightly higher hit rate for all values of α. However, the impact on hit ratio of the parameter α (content popularity) was more significant than the cache algorithms.

When CS was limited to 50 and 100 entries, the difference between the cache algorithms was around 12% for all values of α. With CS limited to 200 entries, this difference decreased to approximately 9%. The average increase with the variation of α from 0.6 to 1.0, for both cache replacement algorithms, also tends to decrease when increasing the cache hit ratio (either when α increases or when CS capacity increases). Figure 4 shows the number of requests that had RTT inside the intervals of 0–30ms for all values of α and all CS configurations. Note that the performance of the LRU algorithm was better, as expected for this scenario. A little variation on ($\alpha = 0.6$ to $\alpha = 1.0$) increases substantially the number of requests with small RTTs (hits).

These results show that the impact of the content popularity can be more effective in performance gain than the cache replacement policy. However, we emphasize that the main

TABLE I
COMPARISON OF AVERAGE RTT AND CACHE HIT RATIO FOR LRU E FIFO POLICIES.

(a) CS limited to 50 entries.

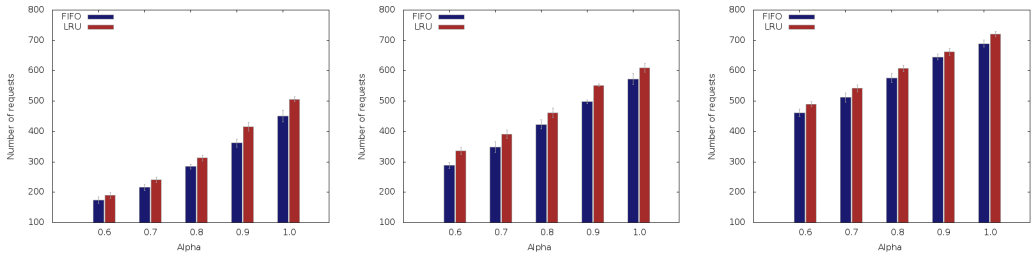
α	Hit ratio – FIFO	Hit ratio – LRU
0,6	0,173 ± 0,010	0,189 ± 0,009
0,7	0,215 ± 0,009	0,240 ± 0,008
0,8	0,284 ± 0,008	0,312 ± 0,009
0,9	0,362 ± 0,013	0,415 ± 0,014
1,0	0,451 ± 0,019	0,506 ± 0,008

(b) CS limited to 100 entries.

α	Hit ratio – FIFO	Hit ratio – LRU
0,6	0,287 ± 0,009	0,335 ± 0,011
0,7	0,348 ± 0,016	0,391 ± 0,014
0,8	0,423 ± 0,015	0,461 ± 0,015
0,9	0,498 ± 0,005	0,552 ± 0,003
1,0	0,573 ± 0,017	0,609 ± 0,014

(c) CS limited to 200 entries.

α	Hit ratio – FIFO	Hit ratio – LRU
0,6	0,461 ± 0,011	0,489 ± 0,008
0,7	0,512 ± 0,015	0,542 ± 0,010
0,8	0,576 ± 0,014	0,607 ± 0,009
0,9	0,645 ± 0,009	0,661 ± 0,010
1,0	0,689 ± 0,011	0,720 ± 0,008



(a) CS limited to 50 entries. (b) CS limited to 100 entries. (c) CS limited to 200 entries.

Fig. 4. RTT comparison for FIFO (blue) and LRU (red) cache policies.

contribution of this experiment is the dynamicity the SDCCN architecture gives to build cache replacement policies without reprogramming the switches.

B. Implementation of Strategy Layer forwarding policies

The Strategy Layer is responsible for Interests routing. In SDCCN, the Strategy Layer is composed by the instructions, actions and events that are used in the communication between the data plane and the control plane. Its operation is defined in the control plane in order to incorporate all of the benefits of the packet forwarding programmability presented in SDN architectures. In addition, the Strategy Layer takes advantage of the overview offered to the control plane and can make smarter decisions,

The Strategy Layer encompasses all decisions involving the Interest routing, such as dynamic and static routing and node-to-node Interest retransmission. Upon receiving an Interest in the network, the switch forwards the packet following FIB entries. These entries are defined by a routing algorithm, which runs in the control plane. If the switch has not received a corresponding Content after waiting a maximum waiting time, set previously, it signals the controller sending a INTEREST_INFO message. The controller runs a routing algorithm and sets the new routing, modifying FIB entries through FLOW_MOD messages. After, the switch resends the Interest packet, following the new FIB rules installed.

C. Experimental results

Content caching programmability supported by SDCCN architecture brings numerous benefits compared to the original CCN architecture, among which we highlight: (i) the possibility of proactive caching; (ii) a global view for cache aggregation; (iii) native support for load balancing.

Proactive caching refers to the ability of the controller to request the caching of a particular content in a switch without an Interest has gone through this switch. The controller can install content via a direct connection to the switch or requesting the switch to make a request for such content on the network (sending an Interest packet on the network).

The decision of where to cache a content on the network is difficult because it may depend on several factors such as delay of links, size of the CSs and their occupation, network topology. The closer the end-user is to the content, the shorter the response time. However, the closer the content is to the end user, the greater the chance that this content is duplicated on different nodes. SDCCN makes this decision much easier because the control plane has information of the network topology and can request data about the tables of each managed node.

Load balancing is natively supported by SDCCN due to POF protocol features used. As the search in POF is done using byte and offset, it is possible to load balance packet forwarding by content name prefixes and even chunks. The same can be done to balance content storage. As an example, the controller may create rules so that only one switch store pair chunks and another odds only. Thus Interest requests to the same content may be divided between the two switches, dividing the load on different links and cache in different switches. The control plane can also offer an API for applications to request load balancing services, enabling the network to offer customized and open solutions for applications.

1) Proactive caching : To illustrate the benefits of proactive content caching on the network using SDCCN, we set up a scenario based on the experiment “Content Distribution Efficiency” [1]. This experiment evaluates the content sharing performance by comparing the total time to recovery simultaneously multiple copies of a 6MB file. Client nodes are connected to a switch with a link of 1 Gbits and the file server is connected through a link of 10 Mbits to the same switch.

Figure 5(a,b) illustrates the network topology. The node `h1`

acts as a content generator and the other nodes download concurrently that content. Initially all nodes are connected to the switch `s1`. Six rounds were carried out similarly to the experiment "Content Distribution Efficiency", explained as follows: in Round 1, only one node expresses interest for the content; in Round 2, two interests in the same content simultaneously; in Round 3, three interests in the same content simultaneously and so on.

The topology is configured for three architecture models: (i) CCNx, using the testing environment created but using CCNx nodes in place of switches; (ii) SDCCN, using the experimental environment created with CCNx nodes and SDCCN switches; (iii) TCP/IP, in a simulated network supporting the TCP/IP stack. In CCN environments was used CCNx 0.8.2 and `ccnr`, `ccnputfile` and `ccngetfile` tools to transfer a 6MB file between the producer and consumer nodes.

Besides the inclusion of a new switch and the addition of the SDCCN architecture in the comparison, other modifications were made to the experiment to explore the proactive content caching in a mobility scenario. CS contents have not been erased between rounds, keeping content in cache for subsequent requests of Interest. Before the last round, consumers lose the connection to the switch `s1` and connect to the network just by connecting to `s2`. In CCNx network, `s2` CS will start empty and requests should pass through the 10Mbps link. In SDCCN environment, the controller can anticipate the migration and proactively orders the installation of contents on the CS of the switch `s2`. It is worth mentioning here that it does not necessarily required to make up a copy of the whole content. Since the content is a file that is being requested sequentially (in pieces, or chunks, according to the CCN architecture), if prior to mobility the user is consuming the piece x, it is possible to order `s2` to proactive cache the piece $x + N$. As the pieces have limited size, the copy time of a piece between a cache and another will be very small, which enables your copy during the mobility event.

For each round the cumulative time of content download requested by all parties was calculated, that is, from the beginning of the downloads of the round until the last Interest of the round is satisfied. The experiment was repeated 5 times for averaging and confidence interval (confidence level of 95 %). The comparison between the download time and number of simultaneous downloads for TCP/IP, CCNx and SDCCN environments can be seen in Figure 5(a,b). In TCP/IP enviroment, contents have to take a longer way for all requests, contents will traverse the 10 Mbps link to access the producer `h1`. Thus, the higher the number of simultaneous downloads, more this link will become saturated and greater the time required for downloading. In CCN networks, during the first request contents are cached in CS of `s1`. Thus, the remaining requests for that content passing by `s1` do not go through the 10Mbps link. As the 1Gbps link doest not get saturated, the download time remains almost constant.

In the last round, the download time increased in CCNx environment because the content that was in `s1` is no longer accessible by the 1Gbps link. So the content must go through the 10Mbps link. In SDCCN environment, the download time remained constant because the controller proactively commands the copy of the contents of the cache in `s1` to `s2` (Figure 5 (c)).

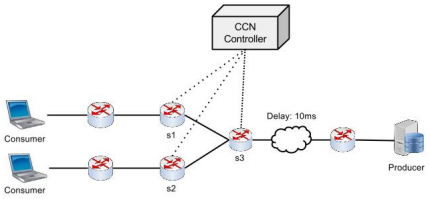


Fig. 6. Scenario considered to evaluate cache aggregation in SDCCN.

We can see that for the first request SDCCN solution achieved a slightly worse response time than the original CCNx. This is because the SDCCN switch is based on POF switch, which lacks some optimizations for content routing.

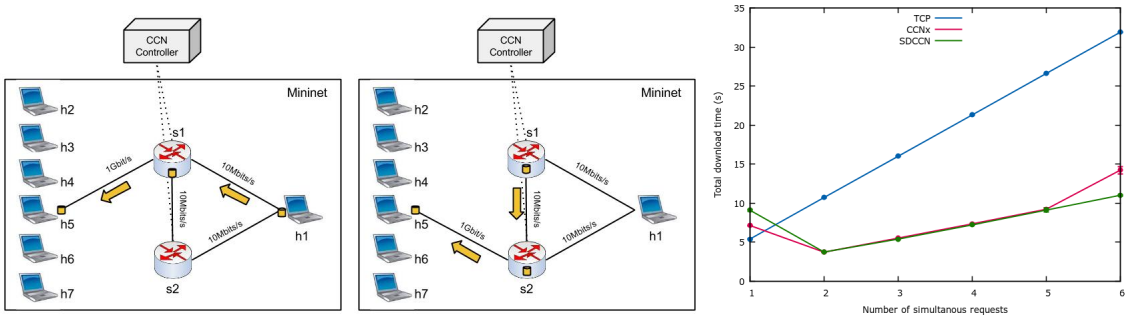
2) Cache aggregation: In SDCCN architecture, cache aggregation in central or common nodes can be done in a very efficient way due to two features of the architecture: overview of the network and cache programmability. SDCCN provides an overview of the CCN network in the control plane. The controller has knowledge of the network topology and the content of the tables of all switches it manages. The cache programmability enables the management of its CS and CRT tables programmatically at runtime.

To demonstrate how cache aggregation can be explored to improve networking cache efficiency, an experiment was carried out whose topology is illustrated in Figure 6. Two consumers request content simultaneously from a single producer. All links have delay of 1ms except the link that connects the switch `s3` to the producer's switch, which has a 10ms delay. The three switches `s1`, `s2` and `s3` are managed by the controller and cache content on the network. The other switches only forward packets.

In this experiment two cache replacement algorithms were set. In the first algorithm, the three switches cache all of the contents generated by the Producer and each one performs LRU independently, i.e. without being aware of the others. The second one has an aggregation scheme. Only `s1` and `s2` cache contents generated by Producer passing over the network. Switch `s3` only caches content when requested by the controller, acting as an aggregator of common contents to the two other switches. The cache replacement algorithm of these switches works as follows:

1) `s3`'s CS data is queried and contents in `s3` are removed from `s1` and `s2`.
2) Contents in `s1` and `s2` that aren't in `s3` are removed from `s1` and `s2` and added to `s3`.
3) If CS utilization is still beyond the alert limit (for this experiment, 45 entries), use LRU to remove contents.

The controller implements a scheme of "copy" of the CS of the switches, so that they does not need to make extra CACHE_INFO requests. The switch `s3` uses the LRU algorithm to replace content in cache. This algorithm involving



(a) Initial setting of the created scenario. (b) The setting of the created scenario after the migration of the clients to `s2`. (c) Comparison of download time and number of simultaneous downloads between TCP/IP and CCN networks.

Fig. 5. Topology and Results for comparison of the download time in TCP/IP, CCN and SDCCN networks.

content aggregation via runtime queries is impossible to be implemented in the CCN architecture without the addition of new protocols, which would involve changes in switch code.

`Ccnping` and `ccnpingserver` tools for content generation between producer and consumer nodes. The CS of the switches was limited to 50 entries and the threshold was set to 45 entries. The limit of 50 entries represents 10% of the maximum number of contents and was fixed because the variation of α has the same purpose (frequency of generation of popular content). The prefix was defined as "ccnx:/UFES" and it was used with `ccnping`. Forwarding rules were added to the FIBs and CRT. Content names have the same pattern (ccnx:/UFES/[N]), where [N] is a positive integer randomly generated using a Zipf-like distribution with α parameters equal to 0.6, 0.7, 0.8, 0.9 and 1.0).

Figure 7 shows the comparison of the hit ratio of the network cache for each value of α for both algorithms (LRU with aggregation and without aggregation). The LRU algorithm with aggregation obtained a considerably higher hit ratio for all values of α. However, the difference tends to decrease when α increasing. In Table II, the network cache hit ratio increases 19.12% on average, reaching up to 34% when α equals to 0.8.

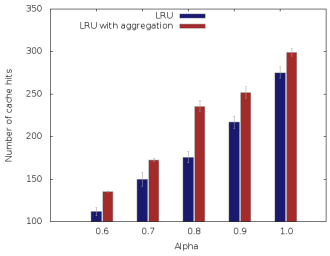


Fig. 7. Comparison of the number of cache hits between LRU with and without aggregation.

3) Load balancing: The goal of this experiment is to show how SDCCN architecture is able to support this kind of service, and the question here is not about the discussion of optimal strategies of load balancing. In CCN features common Interests are aggregated in the PIT during routing process. So it is interesting that common requests are routed to the same path, increasing the possibility of aggregation in the PIT and also the chance to get content from cache; and Content packet goes through the same path of the corresponding Interest packet. Thus, by balancing the Interest routing, the content delivery will also be balanced.

TABLE II
INCREASE OF THE AVERAGE RTT OF THE LRU WITH AGGREGATION WHEN COMPARING WITH SIMPLE LRU.

α	Hit ratio – LRU	Hit ratio – LRU with aggregation
0,6	0,223 ± 0,009	0,271 ± 0,001
0,7	0,300 ± 0,015	0,345 ± 0,004
0,8	0,351 ± 0,013	0,471 ± 0,012
0,9	0,433 ± 0,014	0,503 ± 0,013
1,0	0,550 ± 0,013	0,598 ± 0,008

From these observations, an experiment was set up in order to demonstrate the facilities offered by SDCCN architecture to provide load balancing services. Figure 8 illustrates the environment set up for this experiment. The network consists of four consumers who request contents of a single producer. Consumers are connected to a load balancer switch that distributes Interest packets among four switches. All switches are controlled by the same controller but only the four core switches are able to cache content.

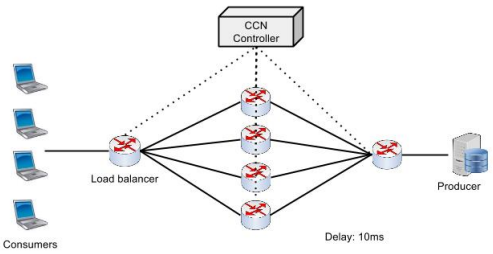


Fig. 8. Architecture of the created scenario to demonstrate the facilities of load balancing in SDCCN.

Two simulations to transfer a single file between consumers and the producer were conducted. Each link has a delay of 10ms. The file was divided into 160 chunks of 8 bytes each.

Each consumer requests 3 chunks in parallel (operation similar to a TCP window size of 3). CSs were limited to 50 entries to fit all chunks in cache when divided equally by 3 switches and the alert threshold was set to 49 entries. The switches applied LRU algorithm for cache replacement.

For each simulation two load balancers were set up. The first is a simple load balancer, which forwards each Interest request related to file transfer to a different port (using round-robin). This balancer does not require any intelligence and functionality of the SDCCN architecture. The second one divides chunks between the 4 core switches, maximizing the chances of aggregation in PIT and CS. The division of chunks is made throught the routing of Interests, which was programmed as follows: requests for chunks ending in 0, 4 and 8 go to the first switch; requests for chunks ending in 1, 5 and 9 go to the second one; requests for chunks ending in 2 and 6 go to the third one and, finally requests for chunks ending in 3 and 7 go to the fourth one.

The second balancer was implemented in SDCCN architecture without any modification to the data plane due to the characteristics of the POF protocol. The bit mask functionality was used, allowing only the content name prefix and the last digit of the chunk to be evaluated during the FIB search. The implementation of this balancer in the CCN architecture would involve reprogramming CCN routers.

In the first experiment, consumers sequentially downloaded the file located initially only in the producer. Table III presents the average of the file transfer time between each consumer and producer and the same average but excluding the first transfer, as during this transfer there will be no cache hit.

From Table III we can also observe that there was a decrease of 34% in the total transfer time when considering the first transfer and 45% when disregarded this transfer. The cache hit ratio of the first balancer was very low (approximately 0.175) because the same chunks were routed to different paths in many cases. As the second balancer forwards chunks to the same path and all the chunk were divided between four core switches, the cache hit ratio was in 100%.

TABLE III
COMPARISON BETWEEN THE LOAD BALANCERS DOWNLOAD TIME FOR THE SEQUENCIAL FILE TRANSFER.

Load balancers	Average transfer time (s)	Average transfer time without first request (s)	Hit ratio
First	4,439 ± 0,029	4,332 ± 0,042	0,175 ± 0,020
Second	2,925 ± 0,003	2,358 ± 0,004	1

In the second experiment, all transfers were made simultaneously. The total transfer time was calculated as the interval from the beginning of the transfers until the end of the last transfer. The first balancer obtained a slightly higher transfer time (approximately 3.86%) than the second one. Thus, the benefit of the aggregate common Interests in PIT is lost, increasing the number of contents sent by the producer and thereby increasing the response time.

VI. CONCLUSION

Content-Centric Networking (CCN) promotes a better abstraction for the Internet use. However, one important problem in the current Internet architecture was incorporated: experimentation is still costly as they often involves reprogramming the code of switches, cache decisions and routing strategies. SDCCN represents a step foward in this direction. In this approach, control plane has a logically centralized controller and the data plane is composed of CCN switches. The controller program the routing tables and cache strategies. Content can also be dynamically installed on a CCN switch to optimize its delivery. In this paper, we also presented a experimental environment created from Mininet to provide easy and fast prototyping experimentation of SDCCN scenarios. Several scenarios based on this environment were built to demonstrate the benefits of the SDCCN architecture.

For future work, we plan to add security messages to the implementation and provide a complete and otimized solution. We also intend to study packet fragmentation strategies, to propose new algorithms for inter-domain routing, to provide implementations of the intra-domain algorithms currently used (such as NSLR [13]) and to propose smarter cache strategies, exploiting the advantage that the controller has a global view of the network topology. Finally, we pretend to experiment the solution with real hardware, such as Mikrotik/Routerboard switches.

REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th ACM CoNEXT*, 2009, pp. 1–12.

[2] Cisco, "Cisco visual networking index: Forecast and methodology, 2014–2019," 2014.

[3] A. Detti, N. Blefari Melazzi, S. Salsano, and M. Pomposini, "CONET: A Content Centric Inter-networking Architecture," in *Proceedings of the ACM SIGCOMM ICN*, 2011, pp. 50–55.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[5] X. Nguyen, D. Saucez, and T. Turletti, "Providing CCN functionalities over OpenFlow Switches," *Research Report*, 2013.

[6] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling Information Centric Networking in IP Networks Using SDN," in *Proceedings of the IEEE SDN4FNS*, Nov 2013, pp. 1–6.

[7] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie, "Optimal Cache Allocation for Content-Centric Networking," *Proceedings of the IEEE Intl. Conference on Network Protocols*, 2013.

[8] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proceedings of the IEEE INFOCOM*, 1999, pp. 126–134.

[9] H. Song, "Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane," in *Proceedings of the Second ACM SIGCOMM HotSDN*, 2013, pp. 127–132.

[10] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the 9th ACM SIGCOMM Hotnets*, 2010, pp. 19:1–19:6.

[11] (2015) Official implementation of the ccn model. [Online]. Available: https://www.ccnx.org/.

[12] (2015) Named data networking github repository. [Online]. Available: https://github.com/NDN-Routing/ccnping/

[13] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "NLSR: Named-data Link State Routing Protocol," in *Proceedings of the 3rd ACM SIGCOMM ICN*, 2013, pp. 15–20.