Enabling Transparent Caching in LTE Mobile Backhaul Networks with SDN

Moises Rodrigues* † György Dán† Massimo Gallo‡

* Networking and Telecommunications Research Group (GPRT),
Federal University of Pernambuco (UFPE), Recife, Pernambuco, Brazil
† School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden
‡ Bell Labs, Nokia, Nozay, France
Email: *moises@gprt.ufpe.br, †gyuri@kth.se, ‡massimo.gallo@nokia.com,

***Abstract*—Today's mobile network architecture lacks the flexibility to efficiently handle the fast increasing amount of data traffic. SDN could potentially provide the required flexibility in future mobile architectures, however its introduction into the existing architecture is challenging.**

In this paper we propose a solution to improve the mobile backhaul's flexibility based on SDN, compatible with the current architecture, focusing on the transparent in-network caching's use case. We explore the design space in terms of function placement, and propose a scalable solution for in-network caching based on stateless DPI and stateful connection tracking. We implemented a prototype of the proposed solution to evaluate its impact on MPEG DASH streaming performance. Our experimental results show that the delay introduced by our module is less than 5ms for 99% of the packets, which is negligible in today's LTE networks, and the slight negative impact on streaming rate selection is easily outweighed by the increased flexibility.

I. Introduction

In the last decade Internet traffic significantly increased owing to the penetration of broadband access. In particular, mobile traffic grew by 70% during the last year from 1.5 to 2.5 exabytes per month. The increase is mainly driven by the rapid innovation and diffusion of mobile terminals and video related services and it is predicted that by 2020 more than 60% of Internet traffic will be originated by mobile terminals [1].

The current mobile network architecture is an IP packet switched network built around 3GPP specifications that define the Evolved Packet System (EPS) architecture, the foundation of fourth generation mobile networks. In the EPS traffic is encapsulated in GTP (GPRS Tunnelling Protocol) tunnels transporting packets from edge nodes, eNodeB, to mobile network gateways, such as the Packet Data Network Gateway (PGW), where packets are forwarded towards the global Internet. Although this architecture has been able to drive the ongoing mobile revolution, it lacks the flexibility needed to dynamically control the constantly increasing amount of mobile traffic. Indeed, GTP tunnels impose that packets traverse the whole mobile infrastructure, and transform the mobile backhaul into a passive network segment in which traffic cannot be dynamically managed.

Forced by the limited flexibility of the current infrastructure, to meet bandwidth and latency requirements despite the increasing amount of traffic in their networks, mobile network operators have been constantly increasing their network capacity. Although effective, increasing the network capacity significantly increases mobile network operators' costs as it requires the deployment of additional infrastructure. To alleviate this problem, motivated by cache-ability studies [2], [3], [4], edge caching solutions have been explored recently [5], [6], and commercial edge caching products have become available, e.g., LTE caches by ARA Networks [7], and DatE by I-Direct [8]. These solutions are, however, limited to the network edge, and since they do not allow to bypass GTP tunnels, they miss the potential benefits of in-network caching and dynamic traffic management.

While Software Defined Networking (SDN) is considered as an enabler for the emerging 5G mobile backhaul architecture, co-existence with EPS requires that an SDN-based backhaul would have to support dynamic traffic management for the existing architecture, without additional middleboxes. In this paper we propose a solution to enhance the mobile backhaul's flexibility based on SDN technology, compliant with the current architecture. In particular we propose a user-space extension to OpenFlow switches inside the mobile backhaul and show the benefits of network devices' programmability by designing and prototyping a transparent cache service. The proposed in-network caching system can be deployed on top of OpenFlow switches in order to reduce the excessive load observed in the mobile backhaul network and can be offered by the mobile network operators to Content Providers.

The main contributions of the paper are (i) design space analysis for the introduction of SDN inside the current mobile backhaul architecture through the in-network caching use case; (ii) design of a solution for transparent caching of MPEG DASH multimedia content; (iii) prototype of SDN-enabled in-network caching solution compatible with current standards and protocols; (iv) feasibility analysis of the proposed solution.

The rest of the paper is organized as follows. Sec.II introduces basic concepts of the mobile backhaul architecture and MPEG DASH streaming. Sec.III explores the design space and describes the proposed design, while Sec.IV presents our prototype implementation and its experimental evaluation. Sec.V discusses related work and Sec.VI concludes the paper.

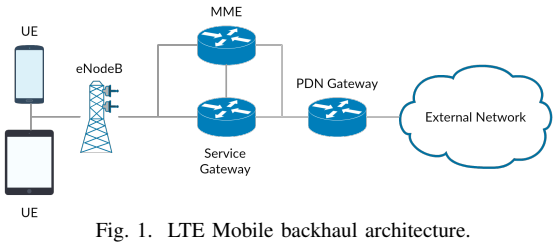


Fig. 1. LTE Mobile backhaul architecture.

II. Background

A. Mobile Backhaul Architecture

The backhaul is the part of the mobile network that ties the core, which connects to the Internet, to the antennas that provide wireless access to mobile devices. Over the years mobile backhaul evolved from a plesiosynchronous digital hierarchy (PDH) and ATM transport to an Ethernet based transport network, in which data are transmitted over IP.

Key elements of the Long Term Evolution (LTE) architecture are illustrated in Fig. 1. User equipment (UE), are connected to the network through antennas associated with at least one eNodeB. The eNodeBs provide network access to UEs by performing radio admission control, dynamic resource allocation and transforming the radio signal into digital information. At the eNodeB, the UE's traffic is encapsulated in a GTP Tunnel that is directed over the serving gateways (SGW) and terminated at the Packet Data Network Gateway (PGW). SGW's function is to route incoming UEs' traffic in the appropriate GTP Tunnel, as well as to support users' mobility driven by the Mobility Management Entity (MME). The MME is the key element of the mobile backhaul's control plane and is responsible for multiple control operations such as UEs' mapping, authentication, authorization, etc. Finally, the PGW is the termination point of the mobile backhaul, and connects the network to the Internet. Its main functions are packet filtering and marking, accounting, and IP address assignment under control of the MME.

B. MPEG DASH Streaming

MPEG Dynamic Adaptive Streaming over HTTP (DASH) [9] is a popular video-on-demand protocol adopted in 3GPP for video streaming over mobile networks. DASH video content is partitioned into one or more segments, with typical segment durations between 1s and some tens of seconds. The DASH client retrieves stream's metadata by downloading from the Media Presentation Description (MPD) file, which specifies segment information including timing, duration, available bitrates and resolutions, and the *URL*. Given the MPD file, the DASH client requests segments sequentially using HTTP and chooses the bitrate of the next segment based on the estimated download rate. A DASH client would thus generate an HTTP request up to once per second on average, depending on the segment duration. While DASH naturally lends itself to caching due to relying on HTTP, GTP tunneling in the mobile backhaul makes it challenging to implement efficient dynamic caching. In what follows we propose a solution to address this challenge leveraging SDN.

III. Transparent Caching in the Mobile Backhaul

The introduction of SDN switches would provide mobile network operators a more flexible architecture allowing them to sustain the increasing amount of traffic while reducing infrastructure's maintenance costs, as well as giving them a way to increase their revenue by introducing novel in-network services. To investigate the feasibility of SDN introduction in the current mobile backhaul architecture, in this section we explore the use case of transparent in-network caching for MPEG DASH video content.

To enable DASH content's transparent caching the proposed architecture needs to (i) intercept an HTTP GET request in a TCP segment transmitted in a GTP tunnel, (ii) decide if the request is for cacheable content, (iii) decide if the requested segment is cached and, if yes, (iv) remove the request from the GTP tunnel and the TCP connection, redirect it to the appropriate cache, and then (v) insert the data received from the cache into the GTP tunnel and TCP connection so that caching remains transparent to the UE. Finally, for charging purpose, caching architecture (vi) should offer a way for tracking requests satisfied by local caches. Next, we propose an architecture for supporting these six network functions and discuss the main design choices. The components of the architecture are shown in Fig. 2.

Function (i) requires stateless DPI, and since all UE traffic traverses the backhaul in a GTP tunnel, it has to be performed by SDN switches, as otherwise all packets would have to be transmitted to the controller. Since various SDN switches are expected to include a DPI engine in the near future (e.g., Open vSwitch [10]), we assume DPI is available.

To decide whether the request is for cacheable content, function (ii) matches the destination socket of the request (extracted from the GTP tunnel) against a list of known content server sockets, maintained in the Content Server Directory (CSD). Function (ii) can be implemented using *stateless* DPI, although the entries in the CSD may change over time.

To support function (iii) we maintain a Content Location Directory (CLD), which holds content placement information i.e., the address(es) of the cache(s) serving the segment, and a reference to the DASH MPD file. Notice that obtaining the MPD file of a content is straightforward, and allows optimizations as we discuss later. The CLD can be either centralized (at or near the the PGW) or distributed, and the information it contains does not change due to flow or packet arrivals, hence function (iii) can be considered stateless.

To support functions (iv) and (v) we designed a splicing network function that transparently extracts and reinserts segments into a TCP connection, and packets into the GTP tunnel. Unlike the DPI, the CSD and the CLD, the splicing function is stateful. Finally function (vi) requires that operations (iv) and (v) are registered by the centralized controller in order to keep pricing consistent when traffic flows are served from the in-network caches.

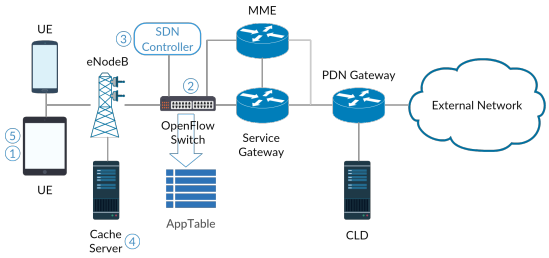


Fig. 2. Transparent caching for LTE network architecture.

A. Design and Function Placement

While function (i) based on stateless DPI must be performed in the switch, several alternatives exist for the placement of functions (ii)-(v), which we will discuss next.

Controller-based: SDN switches use a stateless DPI engine to perform function (i) and redirect the identified requests to the central SDN controller. Functions (ii)-(vi) are then performed in the controller. Since splicing (function (iv) and (v)) is located in the controller, video segments are delivered from the caches via the controller to the switch and charging (function (vi)) is centralized at the controller. While this solution provides a centralized view of the network, is easy to manage, and requires simple stateless DPI in the switches, it does not scale well and has a single point of failure. The single point of failure problem can be partly solved by using a distributed controller, such as the one proposed in [11]. However, a distributed controller increases complexity, and a significant amount of packets has to be forwarded via the controllers, hence the solution is not scalable.

Switch-Controller Hybrid: SDN switches use a stateless DPI engine to perform functions (i) and (ii). If the requested content is cacheable, the packet is redirected to a controller, where functions (iii)-(vi) are performed. In this way only requests for cachable video content are forwarded to the controller, and the DPI on the switch only needs to hold the CSD. However, the amount of traffic traversing the controller remains significant, leading to limited scalability.

Switch-based: SDN switches perform functions (i)-(v) by using stateful L5-L7 packet inspection, similar to [12]. Function (vi) is then performed locally at the switch that regularly provide statistics on charging to the controller. With this solution, the SDN switch still exchanges messages with the Controller (using the CLD to find the best cache to serve the request, in the case of a centralized CLD). However, since the request rate per DASH client is relatively low (up to 1/sec on average), this solution scales well, especially with a distributed CLD. Note that since (iv) and (v) are implemented in the switch, video segments are delivered from the caches via the local SDN switch, not via the controller.

While currently available SDN switches do not have the required features for the implementation of the *switch-based* design, various switches will include DPI and stateful connection tracking (e.g., Open vSwitch [10]) in the near future. In the rest of the section we describe the components of the *Switch-based* design, assuming that DPI and connection tracking are feasible.

B. Switch-based transparent caching for LTE

The proposed solution for SDN enabled transparent caching in the mobile backhaul relies on a local *AppTable* in the user space of the SDN switch, similarly to [12]. In this solution, the SDN switch has rules with 'goto AppTable' actions instead of 'fwd to controller'. If functions (i), (ii) identify a request for cacheable content, the request is redirected to the *AppTable* .*AppTable* entries provide the location of the best cache if the same content segment has been previously requested. Otherwise, the request is forwarded to the CLD (located in the switch or in the controller). If the segment is cached, the CLD returns the location of the best cache (and other information, as described later), and the request is served in the switch's *AppTable* scope using functions (iv) and (v).

AppTable Design

The *AppTable* is an Openflow table in which matching can be performed on L5-L7. *AppTable* entries are of the form

$$<\text{IPaddress}><\text{port}><\text{segmentpath}> \rightarrow <\text{action}>$$

where *IPaddress* and *port* are web server's IP address and port number. The *segmentpath* field is a string of the format "GET path/to/cachedvideosegment", and *action* is the address of the cache where the video segment is cached.

The *AppTable* is initially empty, and populated in response to requests. Every time a request is forwarded to the CLD, and the requested DASH segment is identified as cached, the CLD returns the names of all cached segments listed in the requested segment's MPD file, together with corresponding cache locations. The returned segment names are then added to the *AppTable*. As an effect, upon receiving the request for the first DASH segment of a media file, the *AppTable* is populated with the cache locations for the subsequent segments of the file, which helps keep the load of the CLD low. Although the *AppTable* is populated dynamically, its use does not need per-flow state to be maintained. An existing *AppTable* entry can be invalidated and discarded upon removal of a segment from a cache, the CLD triggers this.

GTP and TCP splicing Functions (iv) and (v), i.e., the removal and insertion of packets into the GTP tunnel and the TCP connection, are based on splicing. TCP splicing is a technique for enhancing L7 proxy performance, which allows the proxy to forward TCP segments received from one endpoint to the other, avoiding application layer segment processing [13]. It is important to note that our use of splicing is different from that of regular proxies. Proxies intercept the SYN from the client and send a new SYN to the server to establish a connection, as implemented in TCPSP [14]. Unlike a regular proxy, we do not splice all TCP connections traversing the switch, but only connections that are requesting content cached by local caches. Therefore, the SYN sent by the client remains unchanged. If a request for cached content is identified in an established TCP connection, the segment

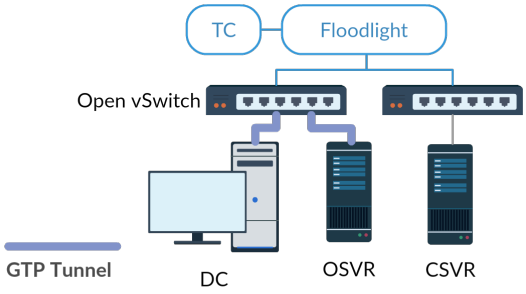


Fig. 3. Experimental testbed topology.

containing the request to the local cache is redirected through splicing. The response sent by the cache is then spliced into the connection between the client and the server. A similar solution is used for the GTP tunnel. Notice that functions (iv) and (v) require per-flow state to be kept.

IV. Prototyping and Experimental Evaluation

In this section we present a prototype-based evaluation of the transparent SDN-based in-network caching solution. In order to evaluate the feasibility of the proposed architecture we implemented a proof-of-concept prototype. Due to the lack of DPI support in OpenFlow, we choose to implement the solution based on a centralized controller. This solution allows us to assess the worst-case impact of the proposed architecture on the download performance of DASH clients, not considering scalability.

A. Prototype Implementation

Our prototype implementation is based on Open vSwitch v2.0.2 [15], and Floodlight v1.1 [1] running on Ubuntu 14.04. The prototype consists in a Floodlight controller's module we identify as the Transparent Caching (TC) module. The TC module performs GTP dissection, payload inspection to detect cacheability (search for HTTP GET method for a locally cached content), acts as a local CLD, and does GTP and TCP splicing, i.e., it implements functions (i)-(v).

TCP Splicing: To implement TCP splicing, upon identifying a request for cached content, we buffer the segment corresponding to the GET method, and we initiate an active open from the module to the local cache by sending a SYN segment. We add the newly initiated connection to the list of spliced connections, and we set its state to Sync, which represents the beginning and the end of a spliced connection. While in this state, if a SYN-ACK is received from the local cache then the module acknowledges its receipt and then sends the buffered GET method to the local cache. Once the local cache acknowledges the segment of the GET method with a TCP ACK, the state of the connection is set to Connected and data start to be spliced. Since after this point everything is spliced, flow control, congestion control and error control are taken care of by the two sessions' endpoints (the DASH client and the cache server). While the connection is in the Connected state, the only TCP segment that needs to be evaluated is the FIN segment. If either the DASH client or the local cache sends a FIN segment then the state of the connection is changed to Sync, Disconnecting and Disconnected after FIN-ACK and ACK messages are sent, or if the connection is inactive for a 2 minutes Maximum Segment Lifetime. When a connection is considered as Disconnected the module removes it from the list of spliced connections.

GTP support: In order to enable GTP in Floodlight we developed two subclasses of the *BasePacket* class found in net.floodlightcontroller.packet, one for GTP-C (GTP Control plane, used for signaling) and the other for GTP-U (GTP User plane, used to transport data packets) packets. The implemented subclasses allow us to handle GTP packets in the developed module, i.e., remove and insert them from/into the GTP tunnel. Removal and insertion of packets require to store the context for each new client opening a GTP tunnel, which consists of the MAC and IP addresses, and GTP information, such as flags and Tunnel Endpoint Identifier (TEID).

B. Experiment Methodology

The topology of our experimental platform is shown in Fig. 3. A GTPv1 tunnel is established between a DASH Client (DC) and the Origin server (OSVR), traversing an OVS switch. The GTP tunnel is created using the *ggsn* and the *sgsnemu* tools in OpenGGSN[2]. The DC establishes TCP connections over the GTP tunnel to the OSVR in order to download DASH segments. When the DC requests a segment over the TCP connection, the Transparent Cache (TC) module detects the request, and if the content is stored by the cache server (CSVR) then the module establishes a TCP connection between itself and the CSVR. Once the TCP connection is established, the TC module starts to splice the TCP connection and the GTP tunnel between the DC and the OSVR.

In order to evaluate the impact of the TC module and of GTP, we considered four different scenarios. The first one is a baseline scenario, in which GTP tunnels are not used, and Floodlight is equipped with a dummy module that takes packets as input and puts them unmodified as its output ('dummy'). In the second scenario GTP tunneling is used and Floodlight uses the dummy module ('GTP'). In the third scenario GTP tunneling is not used and Floodlight is equipped with the TC module that performs all functions but GTP splicing ('TC'). In the fourth scenario GTP tunneling is used and Floodlight is equipped with the TC module that performs TCP and GTP splicing ('GTP+TC').

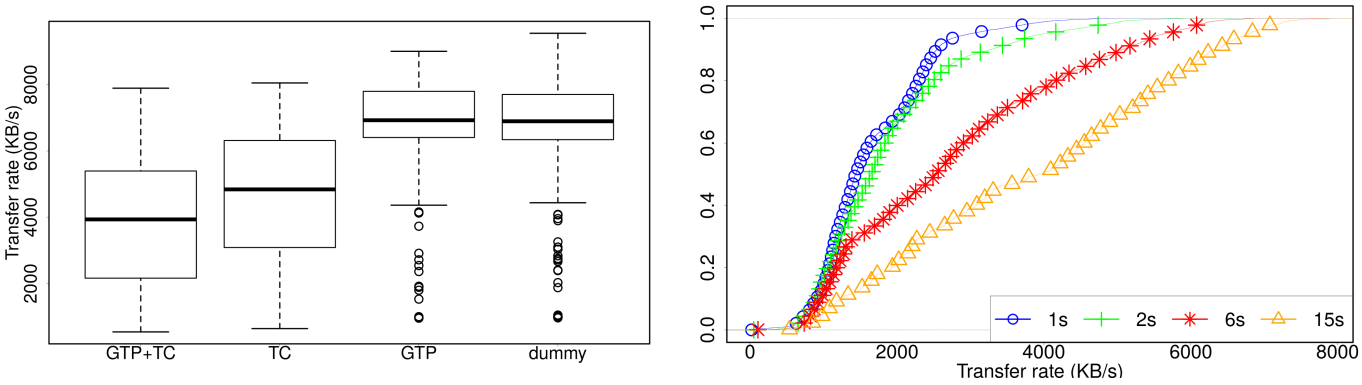
C. Throughput Performance

In the first set of experiments we consider a single DC that downloads DASH segments to evaluate the impact of the TC module on the achievable download rate. We used DASH video segments of lengths between 1s and 15s worth of video content and bitrates between 150 and 8000kbps. For each scenario, we used Apache *ab*[3] as the DC, and downloaded 7293 video segments twice, consecutively.

[1]http://www.projectfloodlight.org/floodlight/

[2]http://openbsc.osmocom.org/trac/

[3]https://httpd.apache.org/docs/2.2/programs/ab.html



(a) Box plot of bitrates for 15s length video segments.

(b) CDF of bitrate for 1s-15s segments for GTP+TC scenario.

Fig. 4. Achieved download rates with and without TCP and GTP splicing during sequential segment download.

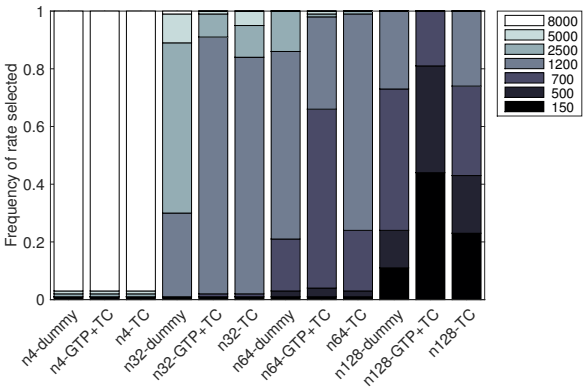


Fig. 5. Segment bitrate selection frequency for three scenarios and $n = 4, 32, 64, 128$ simultaneous DASH clients. Bitrates in the legend in kbps.

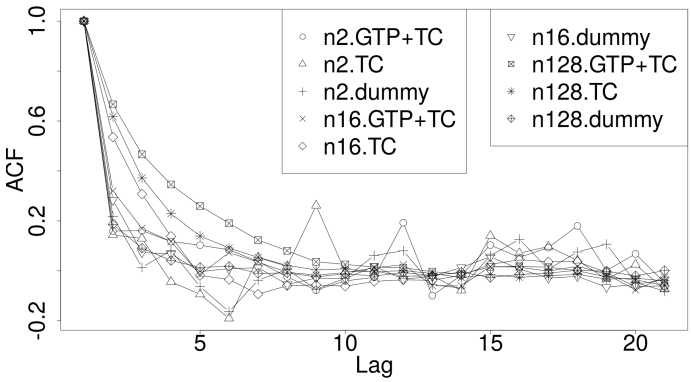


Fig. 6. Autocorrelation function (ACF) of segment download rates for three scenarios and $n = 4, 32, 64, 128$ simultaneous DASH clients.

Fig. 4(a) shows the box plot of the achieved bitrates for the four scenarios in which the DC download 15s segments. We observe a decrease of about 30% in download rate when comparing 'GTP+TC' to the 'dummy' scenario, which is due to TCP and GTP splicing, mainly the delay introduced by the TCP connection establishment from the module to the CSRV. The performance decrease is, however, not that significant considering the complexity of GTP and TCP splicing.

Fig. 4(b) shows the CDF of the achieved bitrates for the 'GTP+TC' scenario for variable DASH segment. It is worth noticing that download rate for small segments is lower, which is due to TCP slow start and to the delay introduced by the additional connection establishment for TCP splicing.

To evaluate the overhead introduced by the TC module we measured the additional latency it generates. For both the 'GTP+TC' and the 'TC' scenarios we found that the additional delay was less than 5ms on average, negligible compared to the typical round trip time of LTE networks, i.e., 50ms.

D. DASH Streaming Performance

In the second set of experiments we consider multiple DCs simultaneously streaming DASH content to evaluate the impact of the TC module on the achieved streaming rate, and the impact of the module on the rate selection algorithm of the DCs, two factors that determine the user perceived QoE. We use 6s length DASH segments and bitrates between 150 and 8000kbps. To perform this set of experiments, we implemented an instrumented DASH client that uses the bitrate calculation and selection algorithm of DASH-js[4]. In order to compensate for Floodlight's multi-threaded packet processing, which can result in out-of-order packet deliveries, we increased the TCP receive buffers to allow for segment reordering.

Fig. 5 shows the bitrate selection of the DASH clients for the 'dummy', 'TC' and 'GTP+TC' scenarios for $n = 4, 32, 64, 128$ simultaneous DCs. For $n = 4$ the highest bitrate (8000kbps) is chosen almost exclusively in all three scenarios, hence the TC module has no significant impact on the rate selection algorithm. For 32 or more simultaneous clients we observe that 'TC' and 'GTP+TC' typically results in one level lower bitrate chosen most frequently (e.g., 700kbps instead of 1200 kbps for $n = 64$).

To gain insight into the rate changes of the DASH clients, Fig. 6 shows the autocorrelation function (ACF) of the download rates achieved by the DCs for the same scenarios and for $n = 2, 16, 128$ simultaneous DCs, as a function of the lag in terms of segments. The ACF shows an exponential decay for all scenarios and number of clients, which shows that the bitrates are stationary both with and without the TC module.

[4]https://github.com/dazedsheep/DASH-JS

The experimental results show that the the proposed solution slightly decreases the streaming performance. Note, however, that in lack of support in the available switch implementations we implemented all functionality in the controller, and the performance of the proposed switch-based solution would be better. Furthermore, in our evaluation scenario the OSVR is located as close to the DCs as the CSVR. In practice, the round trip time to the OSVR may be significantly higher than to the CSVR, which again would favor the proposed solution.

V. Related work

Recent works have shown the benefits of in-network caching for mobile networks [2], [3], [6]. While these works show the potential benefits of RAN level caching, they do not consider the implementation of dynamic caching and the impact of GTP tunneling between the SGSN/SGW and the GGSN/PGW, which is the focus of our work.

An SDN-based solution providing cache-as-a-service was proposed in [16]. The proposed solution consists of two entities, a coordinator and a cache node. The coordinator receives and redirects requests according to information retrieved from Content Providers about cached contents. The proposed architecture does not consider a mobile network scenario and does not implement real-time cacheable content detection. SDN-based transparent caching in the RAN was proposed in [17]. The proposal relies on deep packet inspection (DPI) to identify requests for cacheable content, triggering a content-dependent IPv6 address assignment to the request. The prefix is later used to forward the request to the appropriate cache. Authors in [18] propose an MME within an SDN controller to support content delivery through Content Centric Networks (CCN). The solution proposes to use an IPv6 header extension to identify CCN packets at specific Gateway, placed after the P-GW, and redirect requests to the appropriate cache. Also in the context of CCN, [19] proposes an SDN-based extension of the CONET architecture, which is composed of ICN Clients, ICN Servers and a Name Routing System (NRS). Packet forwarding is done based on content name in a "Lookup-and-Cache" fashion where each forwarding node contacts the NRS for routing information. Content is identified using tags within an ICN/Openflow domain, eventually removed by edge nodes whenever traffic is leaving the ICN domain. The solution was implemented on the OFELIA testbed using Openflow v1.0.

Unlike all previous works, we propose a scalable solution for SDN-enabled dynamic caching compatible with current LTE backhaul technology, and supports dynamic request redirection to caches based on locally managed lookup tables that can proactively be populated based on meta information such as the MPEG DASH MPD for minimizing the control traffic.

VI. Conclusion

In this paper we investigated the feasibility of introducing SDN in the current mobile backhaul architecture through the use-case of transparent in-network caching. We analyzed the design space for the introduction of SDN in the mobile backhaul for in-network caching, and proposed a scalable solution compatible with the existing backhaul architecture. We made a controller-based prototype implementation of the proposed solution that demonstrates that the overhead of the network functions needed in order to enable in-network caching in the mobile backhaul through SDN switches (and controllers) is negligible. Furthermore, our results show that the proposed solution has a minor impact on the streaming clients' behaviour.

Acknowledgements

This work has been partially funded by EIT Digital through the Information-aware data plane for programmable networks project, by the Swedish Foundation for Strategic Research through the MODANE project, and by CNPq - Brazil (206501/2014-5).

References

[1] "CISCO Visual Networking Index 2014-2019," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf.

[2] B. Ramanan, L. Drabeck, M. Haner, N. Nithi, T. Klein, and C. Sawkar, "Cacheability analysis of HTTP traffic in an operational LTE network," in *In Proc. of WTS*, 2013.

[3] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Scalable mobile backhauling via information-centric networking," in *Proc. of IEEE LANMAN*, 2015.

[4] C. Imbrenda, L. Muscariello, and D. Rossi, "Analyzing cacheability in the access network with HACkSAw," in *Proc. of ACM ICN*, 2014.

[5] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Comm. Mag.*, vol. 52, no. 8, 2014.

[6] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5G systems," *IEEE Comm. Mag.*, vol. 52, no. 2, 2014.

[7] "ARA Networks, LTE cache," http://aranetworks.com/solutions/mobile_edgeCDN.

[8] "I-Direct, Data-at-the-Edge," http://www.idirect.net/Altobridge.aspx.

[9] "Information technology - dynamic adaptive streaming over http (dash) - part 1: Media presentation description and segment formats," http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=65274, iSO/IEC 23009-1:2014(E).

[10] F. Baudin, "OpenvSwitch L7 matchers & conntrack metadatas," http://www.openvswitch.org/support/ovscon2014/17/1100-OVS-L7-matchers-v1-1.pptx.

[11] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. of USENIX INM/WREN*, 2010.

[12] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware data plane processing in SDN," in *Proc. of ACM HotSDN*, 2014.

[13] A. Cohen, S. Rangarajan, and H. Slye, "On the performance of tcp splicing for url-aware redirection," in *Proc. of USENIX USITS*, 1999.

[14] "TCPSP - tcp splicing for the linux kernel," http://www.linux-vs.org/software/tcpsp/index.html.

[15] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proc. of USENIX NSDI*, 2015.

[16] 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 *Proc. of ICCCN*, 2014.

[17] M. Kimmerlin, J. Costa-Requena, and J. Manner, "Caching using software-defined networking in lte networks," in *Proc. of IEEE ANTS*, 2014.

[18] R. Haw, C. S. Hong, and S. Lee, "An efficient content delivery framework for sdn based lte network," in *Proc. of ACM ICUIMC*, 2014.

[19] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, "Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed," *Computer Networks*, vol. 57, no. 16, 2013.