SDN-Enabled Distributed Access Architecture Cable Networks

Sudhanshu Naithani, Cormac Sreenan, Ahmed Zahran
School of Computer Science and Information Technology
University College Cork
Cork, Ireland
s.naithani, cjs,a.zahran@cs.ucc.ie

***Abstract*—Cable networks are embracing Distributed Access Architectures (DAA) that push traditionally centralized network functions to the network edge. While this shift offers higher data rates, it complicates the management and configuration of the network by having functions distributed in remote nodes. Separately, SDN has evolved for enabling programmable networks of distributed switches, managed in a logically centralized manner. This paper presents an evolutionary path for SDN-based cable DAA that can overcome the aforementioned challenges and support new network services. We present an SDN-DAA architecture, implemented in a real DAA remote device, and evaluated using Mininet to demonstrate the operational benefits.**

***Index Terms*—Distributed Access Architecture, DAA, Cable networks, SDN, RMD, Mininet**

I. Introduction

Cable (television) networks have evolved to support a range of communication services, including high-speed Internet access, offered mostly over Hybrid Fiber-Coaxial (HFC) infrastructure. HFC architectures are centralized, using a head-end device to manage the network elements. The development of digital optics has enabled a shift from HFC to Digital Fiber-Coaxial, providing more reliable communication and facilitating Distributed Access Architectures (DAA) [1].

DAA has revolutionized the cable networking industry by allowing the relocation of the digital portion of the head-end to be closer to subscribers. DAA has two manifestations, namely R-PHY and R-MACPHY [2]. The former pushes the PHY layer out, while the latter pushes both MAC and PHY layers out. This relocation of layers to be near the subscriber results in higher throughput but complicates network management, testing, and maintenance operations. In this paper, we posit Software Defined Networking (SDN) as a suitable basis for a solution, taking advantage of its logically centralized method of controlling distributed network elements, and its programmability for introducing network functions that are new to DAA and of great value to cable network operators.

SDN represents a novel networking paradigm that has positively impacted network operation and management in different ways. The separation of network control and data layers represents a fundamental principle in SDN. Specifically, a network controller is implemented as a logically centralized node using high-level languages while the data layer consists of simple switching devices responsible for packet forwarding. Network services are implemented as network applications on top of the controller. Centralizing the network control is motivated by the need of minimizing the intricacy of network devices and the benefits of central intelligence. These merits have spurred the adoption of SDN by many network operators and seeded research on the application to so-called *hybrid* networks [3], including cable [4] and optical [5], where SDN can also provide a convenient interface for device configuration, in addition to network forwarding control.

In this paper, we explore the practical integration of SDN concepts into existing cable DAA, identifying and solving the attendant challenges. Specifically, the case of implementing an SDN overlay on-top of R-MACPHY devices is resolved, and deployed for evaluation in a real system prototype. Our contributions can be summarized as follows:

- The design and development of an SDN-enabled R-MACPHY capable of defining the forwarding behavior of distributed cable nodes.
- Leveraging SDN to allow DAA dynamic topology monitoring, load-balancing, and failure management.
- Evaluate the effectiveness of the proposed architecture using load balancing as an exemplar application and using both simulations and real DAA hardware.

Our project has been guided by close engagement with an industry partner, and we believe our work is valuable to equipment vendors and cable network operators in planning the evolution of their technology. To the best of our knowledge, ours is the first paper investigating SDN for DAA.

The rest of this paper is organised as follows. Section II presents relevant background and related research followed by our SDN-enabled DAA in Section III. The performance evaluation is in Section IV, with conclusions in Section V.

II. Background and Related Work

In this section, we first present an overview of SDN and OpenFlow, followed by related research.

A. Software Defined Networks and OpenFlow

SDN can be classified into three layers, i.e. data, control, and application, plus two interfaces, i.e. northbound and southbound. The application layer sits above the control layer

This publication has emanated from research supported by a grant from an industry partner and Science Foundation Ireland (Grant No. 18/CRT/6222).

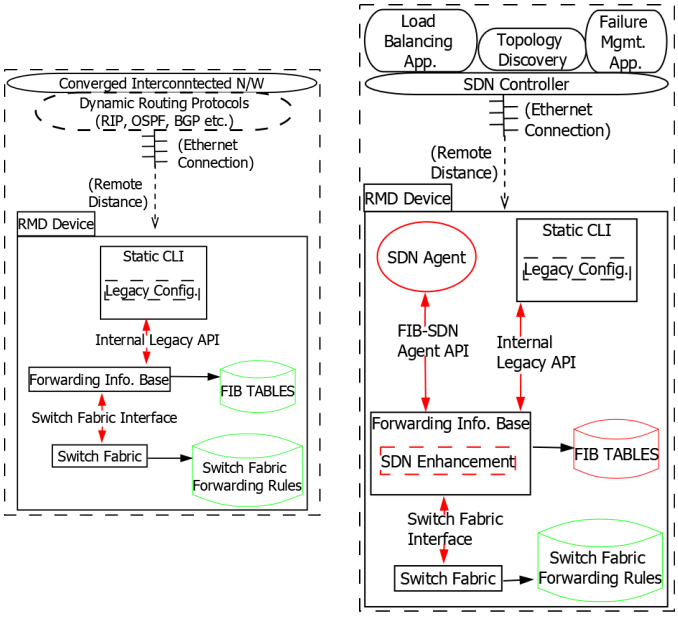
and implements network applications that define the network behavior. Typical applications include routing, load balancing, failure management, quality-of-service (QoS), and security. The data layer sits below the control layer and contains switching devices that forward packets based on its tables that are populated by the control layer through the southbound interface. The control layer is mainly composed of an SDN controller, which takes the role of network operating system, facilitating message forwarding between network applications and switching devices. Commercial controllers, such as ONOS and Ryu, facilitate key services to network applications, such as topology tracking and monitoring traffic statistics. They also feature multiple southbound interface implementations to enable operation with legacy networks. The southbound interface plays the role of a bridge between data layer devices and the SDN control layer.

OpenFlow (OF) is the most widely adopted southbound interface. Messages are divided into three categories based on their nature and direction as follows:-

- *Controller-to-Switch Messages* are sent from the controller to the switches to manage the latter's behaviour or request information. Example of these messages include Features, Configuration, FlowMod, Packet-Out, Modify-State, Barrier, GroupMod, PortMod, and all the Multipart messages (different StatRequest messages).
- *Asynchronous Messages* emanate at the switch to inform the controller regarding events and changes taking place in the data layer. There are four prime asynchronous messages namely Packet-In, Flow-Removed, Port-Status, and Error.
- *Symmetric Messages* are sent without being requested either by switch or by the controller. These messages include Hello, Echo Request/Reply, and Experimenter.

B. SDN for Cable Access Networks

The idea of using SDN in Cable Access Networks has been discussed in earlier publications. In [6], the author presented the case for using SDN in centralized cable networks, outlining the opportunities for streamlining device configuration, including for applications such as high-speed data service and layer 2 VPNs. In a similar vein, [7] gave a vision for how the OpenDaylight controller could be used to control QoS and policy management for centralized cable access networks. QoS configuration on a centralised cable network is also the focus of [4], which presents a solution to use OpenFlow with the headend, known formally as the Cable Modem Termination System (CMTS). Several architectural options are given, including using the DOCSIS standard protocol with OpenFlow, and a custom hardware abstraction layer to provide OpenFlow directly on the CMTS [8]. In contrast, our work is motivated by the more recent shift to the distributed cable architecture with R-MACPHY devices providing a more natural SDN evolution, and thereby also facilitating the introduction of network control applications that are new to cable networks, going beyond the more straightforward tasks of device and policy configuration.



(a) Traditional DAA (b) SDN-Enabled DAA

Fig. 1: DAA Cable Network Architectures

III. SDN-Enabled Distributed Cable Network

We first present the traditional Remote MACPHY (RMD) device components and functionality, followed by the equivalent for the proposed SDN-enabled RMD. For the purpose of enabling SDN, an SDN agent was developed in the remote device. The design and implementation of this agent is described, followed by the use case applications.

A. Traditional-RMD Network System

Figure 1a illustrates the key elements and functions of a traditional DAA cable network. The *Converged Interconnect Network (CIN)* represents a core component that connects remote devices, such as RMDs, and core services, such as packet engines, to facilitate different network services, including video, voice, data. A CIN could follow a single layer or spine-leaf topology. However, spine-leaf would suit large scale networks due to its scalability. Dynamic routing protocols are typically used for on-demand path calculation between network nodes. The *RMD FIB* (Forwarding Information Base) implements packet forwarding rules derived from the flow configurations defined by a legacy config. A static command line interface is typically used for configuring RMD nodes to define legacy configuration that pushes rules into the RMD FIB[1]. At lower layers, the RMD caches a subset of the FIB Table rules in the Switch Fabric Forwarding tables to allow efficient processing of packets. These rules are communicated through the Switch Fabric interface in response to the arrival

[1]Part of the static CLI configuration could be to provision routing protocols, but it is not essential - it is possible to provision a device with just static routes via the CLI. When a routing protocol is provisioned then the FIB will be populated with both the static routes derived from the CLI commands and the dynamic routes populated via OSPF/BGP, etc.

of new flows. The RMD is usually bootstrapped using DHCP (Dynamic Host Configuration Protocol) on the NSI (North Side Interface), connecting the RMD to the CIN.

B. SDN-Enabled DAA

Figure 1b shows the key components in our SDN-enabled DAA system. The SDN Controller is in the core, as part of the CIN, interacting with SDN-enabled remote devices. We use OpenFlow 1.3 at the south-bound interface. RMDs are modified to enable management through the controller. First, an SDN agent implements the required functionality for controller connection management. The agent is initialized as part of the bootstrapping process and receives the controller channel configuration (e.g., the controller IP and port) from the modified FIB. The agent solely handles all OF messages that do not involve the data plane. Examples of these messages include OFPFeaturesRequest, OFPRoleRequest, OFPDescStatsRequest, HELLO and ECHO messages. Additionally, the agent implements a bi-directional OF interpreter to translate incoming controller OF messages into RMD data plane actions, using device-specific APIs, and maps data plane messages into corresponding OF messages in the reverse direction. Examples of these messages include PacketIn, PacketOut, FlowMod, and all statistics requests and replies.

It is important to note that no changes are needed in the switch fabric (lower layers) and the modified FIB only extends the existing one. Hence, the SDN-enabled RMD can be configured using both the legacy mode and SDN controller. When running under SDN, the FIB behavior is defined by the flow configurations received from the controller. When the controller sends a message to the RMD, e.g., PacketOut or FlowMod, the SDN agent receives the message and issues the corresponding call to the modified FIB that populates the FIB tables and updates the lower layer FIB. On the other hand, when the RMD fabric layer cannot forward a packet, it queries the FIB which consults its tables. If no entry matches the packet, the packet is forwarded with its meta information to the SDN agent that sends a PacketIn OF message to the controller. Similar processing is considered for other controller-switch interaction for various OF message types.

The SDN Agent is implemented in Python and built on top of the extended RMD FIB, which has been developed in C. For the SDN Agent, RYU libraries[2] are repurposed from a switch perspective. The key modifications made in the RYU code includes extending controller-to-switch message classes with a parser method and Switch-to-controller message classes with a serialize method. The parser extracts the message content and converts it into FIB-friendly format to pass it to FIB APIs, while the serialize method coverts FIB-originated messages into OpenFlow message format for the controller.

C. Network applications

SDN Applications are implemented on top of the controller to define the network forwarding behaviour and control its operations. SDN enables ample opportunities to optimize the operation of cable networks. In this paper, we consider *topology-aware* routing, load balancing, and failure management applications, on top of the Ryu controller.

Topology Management. Our applications maintain a directed weighted graph for the network topology with the nodes representing switches and end-hosts, and edges representing links connecting various nodes. The graph is updated using different mechanisms for switch, link, and host. For switches, we leverage controller-specific events including EventSwitchEnter, EventSwitchLeave, and EventSwitchReconnected. When a switch connects to its controller, EventSwitchEnter is triggered at the controller and the switch gets added to the topology. We process other events to update the topology when the switch leaves or rejoins the network. Similarly for links, EventLinkAdd and EventLinkDelete are used to update the topology in respective event-handlers. Ryu generates these events when initialized with 'observe-links' option that employ Link Layer Discovery Protocol (LLDP) and OF messages to discover adjunct switches connected to every discovered switch port. For host discovery, we use the first packet generated from the host to update the network graph. Once this packet reaches the RMD, it generates a PacketIn event that is leveraged to extract required information (the host IP to the MAC address).

Routing and Load Balancing. We consider a hybrid routing strategy in our solution. The route of every new flow is identified when its first packet reaches one of the network switches, such as an RMD. Since, this packet does not match any OF rule in the FIB tables, the SDN agent sends this packet to the controller via a PacketIn message, that includes the packet. The controller calculates the best path between the flow source and destination using Dijkstra's weighted short path algorithm. It then sends forwarding rules using FlowMod messages to every switch in the calculated path. Hence, at every hop (switch) an OpenFlow rule is installed, in both directions, for subsequent packets. The controller also directly sends the first packet in the flow to its destination using a PacketOut message at the egress port on the first switch connected to the destination. In the case of inter-AS traffic, the internal flow endpoint, i.e., the gateway, is identified through the MAC address included in the packet embedded in the PacketIn event.

The proposed hybrid routing policy reduces both signaling load and packet delay. The signaling load is reduced as the controller is consulted only once, instead of being consulted by every switch on the path, e.g., from the leaf to the spine. Additionally, sending the PacketOut directly to the destination minimizes the packet delay as the majority of the traffic would be directed to a node residing in the core, e.g., a gateway for Internet traffic or one of the core servers. It is also important to note control packets are prioritized over data packets. Since the first packet of the every flow will be encapsulated in an OF PacketIn message, it would experience a smaller delay by skipping queuing at all switches inside the network.

To enable load balancing, we employ a monitoring applica-

[2]https://github.com/faucetsdn/ryu

tion to assign weights to every edge in our topology graph based on the corresponding link utilization. Every edge in the topology graph is assigned an initial weight of 1. The monitoring application periodically sends PortStatsRequest messages to every switch in the network. Switches reply with PortStatsReply including the number of sent and received bytes, packets, errors, dropped packets at every port. Such replies are used to calculate the link utilization relative to the link bandwidth obtained from the PortDesc message during switch configuration. The link weights are then adjusted based on the utilization. Currently we use a simple procedure in which we assign a low, middle or high weight to each link after comparing the reported utilization against fixed thresholds corresponding to low, middle and high utilization. Hence, the controller will choose the best path for any new flow based on the current network condition. Note that the controller does not need to maintain state about ongoing flows, facilitating scaling to larger numbers of flows.

Failure Management detects port/link failure through receiving PortStatus messages sent by a switch to indicate a change in any port status. The received message contains the reason for the message generation, e.g., port down, port number, port name, hardware address, current speed. Upon receiving the message, the failure management module removes the corresponding edges from the graph and deletes flows destined to the failed port in the switch flow table, using a FlowMod message. Hence, as the packets of an impacted flow arrive at the switch, it encounters a table miss and a PacketIn message is sent to the controller. The controller calculates a new route to the destination and sends a FlowMod message to all switches in the newly calculated route. Hence, subsequent packets are diverted through a new route. If the failed port goes up in the future, edges are again added in the graph with weight 1 and the link is available to be utilized again.

IV. Validation and Performance Evaluation

The proposed SDN-enabled DAA solution was evaluated using different network and traffic configurations. First, the evaluation setup is presented, and this is followed by the experimental results. We choose load balancing as the exemplar application.

A. Experimental Configuration

This section presents the experimental network configuration, network traffic load, and the performance metrics recorded for different kinds of traffic.

1) Network Setup: Figure 2 illustrates our general experimentation setup including the following two nodes:

A VM node represents a virtual machine hosting the SDN controller and Mininet[3] to emulate a three-layer spine-leaf topology. This topology is envisaged for commercial DAA networks, e.g. [9]. Layer 1 (top layer) represents the network core while the leaf layer consists of R-MACPHY devices. Both layers are connected through an intermediate switch layer that

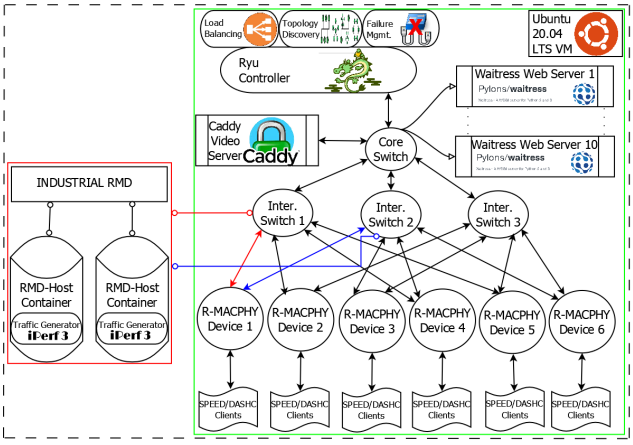


Fig. 2: Experimental Topology

offers reliability through redundant connections. In our topology the core switch is connected to three intermediate switches that are connected with six R-MACPHY Devices. Every R-MACPHY is connected to two intermediate switches. Core, Intermediate and each R-MACPHY are assigned identifiers including three digits, where the first digit of the identifier is the switch layer and the other two digits are the switch ID in that particular layer. Every Mininet R-MACPHY Device is connected to 8 hosts, named with the prefix *h1* to *h8* followed by the switch name to which hosts are connected. For example, the third host connected to the 5th leaf switch (R-MACPHY Device 305) would be named *h3sr305* (sr = switch-RMD).

RMD Node represents an experimental industrial RMD. The RMD is connected to the VM node using two links to enable hybrid experiments involving both real and emulated RMDs. In hybrid experiments, the network interfaces of these links on the VM node are added to switches 201 and 202 as shown in Figure 2. Due to a hardware limitation, only two hosts are emulated in containers in the industrial RMD and are denoted as RMD-Host containers. In simulation-based experiments, only the VM node is used, while both of the nodes are used for hybrid experiments.

2) Network Traffic: In the experiments, we describe web browsing, video streaming, and synthetic traffic.

Video traffic is based on Dynamic Adaptive Streaming over HTTP (DASH). A published video dataset (Dataset 1) was utilized [10], which is served from a Caddy server[4] connected to the core switch. In our experiments, we used 10-min H.264 videos split into two-second segments. Additionally, we use Dashc [11] as a light-weight video client that is configured to use the Logistic [12] adaptation algorithm.

Web traffic is emulated using the SPEED (a Scalable Python wEb bEhavioural moDel) [13] framework. SPEED provides a generator (speed-web-generator) and consumer (speed-web-client) of webpages. The generated content is stored on a *waitress* server[5]. To emulate real web-traffic dynamics from

[3]http://mininet.org

[4]https://caddyserver.com

[5]https://pylonsproject.org

TABLE I: Network operation for No-Load-Balancing

iperf flow	Assigned Route	Traffic Demand
1	Source → R-MACPHY Device 1 → Intermediate Switch 2 → Industrial RMD→ Destination	99.7%
2	Source → R-MACPHY Device 1 → Intermediate Switch 2 → Industrial RMD→ Destination	217%

TABLE II: Network operation for Load-Balancing

iperf flow	Assigned Route	Traffic Demand
1	Source → R-MACPHY Device 1 → Intermediate Switch 2 → Industrial RMD→ Destination	99.9%
2	Source → R-MACPHY Device 1 → Intermediate Switch 1 → Industrial RMD→ Destination	99.5%

multiple servers, we deployed ten servers, and web clients are tuned to uniformly request a web page from one of the servers.

Synthetic traffic is generated using the popular iperf utility, and used to demonstrate load balancing in the real RMD.

3) Performance Metrics: We consider different performance metrics depending on the traffic type. For web traffic, we use the Page Load Time (PLT), which represents the full web page loading delay from the moment of the request time (e.g., clicking the link). For our experimentation, if a page is not able to load within 15 seconds, the page is considered inaccessible and a timeout is produced. In the event of a timeout, PLT is set to 15 seconds.

For DASH video traffic, we record multiple metrics, including average video bitrate, number of quality switches, the number of stalls, and total stall duration. In all considered scenarios, Logistic managed to avoid stalls. Hence, we are mainly reporting visual quality metrics, i.e., bitrate and switches.

B. Load Balancing Validation

We validate load balancing performance using two iperf sessions between the industrial RMD hosts and h1sr301 and h2sr301. Mininet hosts act as iperf clients; hence, the traffic flows towards RMD hosts. The second iperf session starts 5 seconds after the first one, which is configured to saturate the bottleneck link between RMD and intermediate switches. This delay enables the monitoring element to capture high link utilization and adjust the graph weights accordingly.

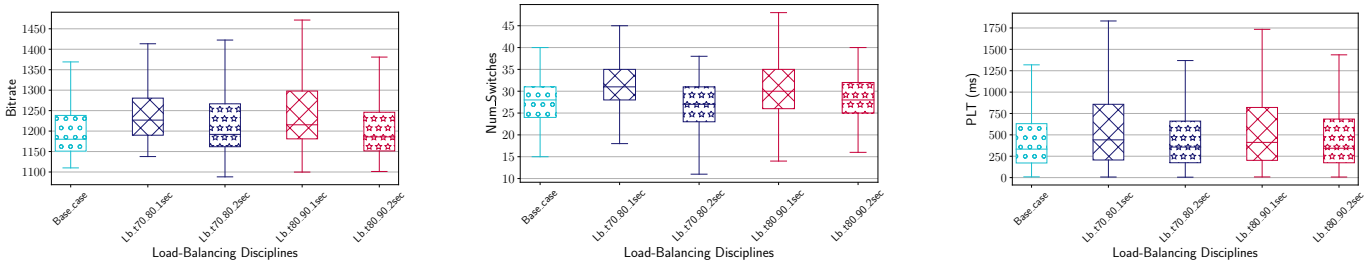
The Tables above show results when using load balancing and when not using it. In the no-load-balancing case, both iperf flows take the same path, highlighted in Blue links in Figure 2. When using load balancing, the flows take different paths, shown as Blue links for Flow 1 and Red links for Flow 2. We note the effectiveness of the SDN-enabled load balancing in avoiding an overloaded link.

C. Real Traffic Experimentation

We evaluate the video and web performance in Mininet in different scenarios with respect to link speed ranges and network load. Specifically, we consider two link rate configurations including (50,10)Mbps and (100,20)Mbps. The former rate represents links connecting intermediate and core switches, while the second value represents the rate for links connecting RMDs and intermediate switches. For the Highly-Loaded scenario, the 8 hosts connected to each RMD are split into three web clients and five video clients. For lightly-loaded scenarios, this is altered so that RMD 1 and RMD 6 run three web but no video clients. Hosts are randomly initiated with a 1s gap. For every configuration, the results are based on averages over 10 runs with randomly-generated seeds. We compare the performance of no load balancing case with the performance of load balancing with different monitoring periods (1s and 2s) and different link utilization thresholds including (70,80) and (80,90). The threshold (x,y) means that as soon as utilization on a link goes above x% the network knows that the link is about to get congested, and if link utilization is above y%, the link is congested and no new flow would be diverted via this link.

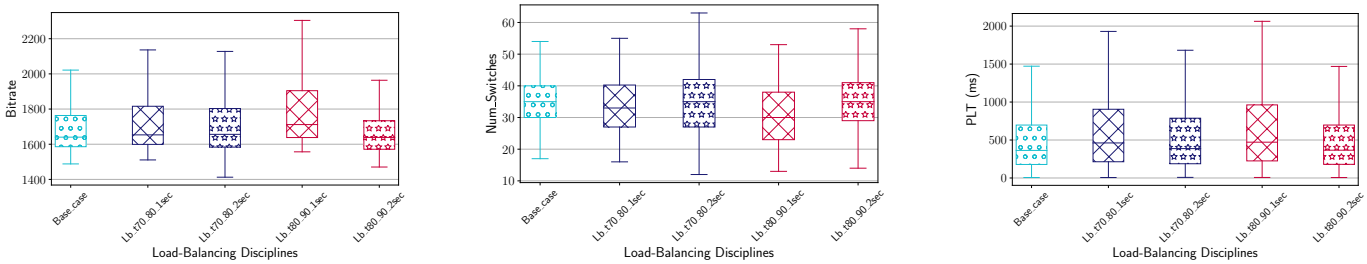
1) 50-10MB-Highly-Loaded Scenario: Figure 3 shows, using box plots, the key performance metrics for this scenario. Specifically, Figure 3a shows that load-balancing improves the bitrate for video traffic in comparison to the base case of no load balancing. For example, LB-T70-80-2sec and LB-T80-90-1sec increase the peak video bitrate by 3.64% and 7.27%, respectively, and also show an increase in the median bitrate. Figure 3b shows the average number of quality switches for video users in different sessions. The video client switches the selected quality level seeking to match the available bandwidth, so would be expected to increase when using load balancing, which would cause changes in the available bandwidth. The number of switches is lower for the less frequent 2-second monitoring because of less rerouting. Figure 3c illustrates the PLT for web traffic users, showing that with load balancing solutions, the PLT with 2-second monitoring shows no increase in the median value but using 1-second monitoring can cause a moderate increase in the median and a notable outlier of several hundred milliseconds. This is due to the system reacting more frequently and thus providing a higher degree of bandwidth changes for the HTTP sessions. It thus seems clear that the 1-second period benefits video sessions, providing a notable increase in video performance without significant impact on web outcomes, especially when using the (80,90) threshold. It is important to note that we obtain similar performance trends but a smaller performance gap for the highly loaded network with link capacity (100,20)Mbps.

2) 50-10MB-Lightly-Loaded Scenario: Figure 4 shows video and web performance for the lightly-loaded network with (50,10)Mbps links. In this case the available network capacity means that the achievable video bitrates are considerably higher, but so too is the amount of switching, as the video players adapt to try to take advantage of a wider range of usable video qualities. This leads to a moderate increase in the median PLT for web pages as shown in Figure 4c. Thus, the choice of 1 second monitoring with the (80,90) threshold is again recommended. Naturally, the lower monitoring period



(a) Video Bitrate (b) Video Quality Switches (c) Web PLT

Fig. 3: Application Performance for Highly Loaded Network with (50,10)Mbps Configuration



(a) Video Bitrate (b) Video Quality Switches (c) Web PLT

Fig. 4: Application Performance for Lightly Loaded Network with (50,10)Mbps Configuration

comes with the cost of additional messaging and analysis overheads, which would need to be considered as one element in the operation of a load balancing solution.

V. Conclusion

The paper serves to illustrate the benefits of integrating SDN and DAA cable networks, through a practical industry-guided implementation-focused study. Specifically, we present the design of an SDN-enabled Remote MACPHY Device through extending its Forwarding Information Base and implementing an SDN agent to facilitate OpenFlow interactions with the SDN controller. The presented design did not impact the legacy hardware, thus lending itself it to an evolutionary network infrastructure path. Several applications of SDN in DAA were specified, and load balancing was used as an exemplar for validation in a hybrid prototype including a real industrial RMD. Additionally, Mininet simulations were used to demonstrate application performance in different configurations. It is clearly shown that SDN-managed load balancing yields up to 7.27% of performance improvement for video traffic, without significantly impacting the web traffic. This research provides a good base on which to further develop future innovative services for Distributed Access Architecture and as a guide for companies in assessing the potential role for SDN in their cable network products and networks.

References

[1] A. Weissberger. Distributed Access Architectures (DAA) Explained, Market Assessment and GCI Deployment. Techblog.comsoc.org. Accessed: 24.03.2023. [Online]. Available: https://tinyurl.com/bdux6tt7

[2] Z. Alharbi *et al.*, "Performance comparison of R-PHY and R-MACPHY modular cable access network architectures," *IEEE Transactions on Broadcasting*, vol. 64, no. 1, pp. 128–145, 2017.

[3] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, pp. 3259–3306, 2018.

[4] A. Mendiola, V. Fuentes, J. Matias, J. Astorga, N. Toledo, E. Jacob, and M. Huarte, "An architecture for dynamic QoS management at Layer 2 for DOCSIS access networks using OpenFlow," *Computer Networks*, vol. 94, pp. 112–128, 2016.

[5] F. C. Garcia Gunning, A. Kaur, N. C. Estrada, J. Raulin, Y. Verbishchuk, and C. J. Sreenan, "Enabling high capacity flexible optical networks," in *2021 International Conference on Optical Network Design and Modeling (ONDM)*, 2021.

[6] K. Sundaresan, "SDNized cable access networks," in *Proc. INTX Spring Tech. Forum*. CableLabs, 2015.

[7] S. Patel, J. Schnitzer, M. Chowdhury, and D. Early, "SDN ground truth:implementing a massive scale programmable docsis network," in *Proc. INTX Spring Tech. Forum*. Comcast, Applied Broadband, 2016.

[8] B. Belter, D. Parniewicz, L. Ogrodowczyk, A. Binczewski, M. Stroiński, V. Fuentes, J. Matias, M. Huarte, and E. Jacob, "Hardware abstraction layer as an SDN-enabler for non-OpenFlow network equipment," in *Proc. Third European Workshop on Software Defined Networks*, 2014.

[9] CommScope. Evolution to Distributed Access Architectures (DAA). (accessed: 27.03.2023). [Online]. Available: https://tinyurl.com/3996yrkp

[10] J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Datasets for AVC (H.264) and HEVC (H.265) Evaluation of Dynamic Adaptive Streaming over HTTP (DASH)," in *Proceedings of the 7th International Conference on Multimedia Systems*, 2016, pp. 1–6.

[11] A. Reviakin, A. H. Zahran, and C. J. Sreenan, "Dashc: A Highly Scalable Client Emulator for DASH Video," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 409–414.

[12] Y. Sani, A. Mauthe, and C. Edwards, "Modelling video rate evolution in adaptive bitrate selection," in *2015 IEEE International Symposium on Multimedia (ISM)*. IEEE, 2015, pp. 89–94.

[13] D. Raca, M. Salian, and A. H. Zahran, "Enabling scalable emulation of differentiated services in Mininet," in *Proceedings of the 13th ACM Multimedia Systems Conference*, 2022, pp. 240–245.