Future Generation Computer Systems 56 (2016) 295–302

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Assessment of SDN technology for an easy-to-use VPN service

Ronald van der Pol [a,*], Bart Gijsen [b], Piotr Zuraniewski [b,d], Daniel Filipe Cabaça Romão [c], Marijke Kaat [a]

[a] SURFnet, Moreelsepark 48, 3511 EP Utrecht, The Netherlands
[b] TNO, Anna van Buerenplein 1, 2509 JE Den Haag, The Netherlands
[c] University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
[d] Department of Applied Mathematics, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland

HIGHLIGHTS

- We describe the architecture of an OpenFlow based multi-domain on-demand L3VPN.
- We give implementation details of the developed demonstrator.
- An easy to use web portal allows end-users to set-up and manage multi-domain VPN.
- Community Connect (CoCo) service can enable integrated resource management solutions.
- We performed functional and non-functional tests in a physical and virtual testbed.

ARTICLE INFO

Article history:
Received 25 February 2015
Received in revised form
4 August 2015
Accepted 11 September 2015
Available online 21 September 2015

Keywords:
SDN
OpenFlow
OpenDaylight
Mininet
VPN
eScience

ABSTRACT

This paper describes how state-of-the-art SDN technology can be used to create and validate a user configurable, on-demand VPN service. In the Community Connection (CoCo) project an architecture for the VPN service was designed and a prototype was developed based on the OpenFlow protocol and the OpenDaylight controller. The CoCo prototype enables automatic setup and tear down of CoCo instances (VPNs) by end-users via an easy to use web portal, without needing the help of network administrators to do manual configuration of the network switches. Users from the research community, amongst others, expressed their interest in using such an easy-to-use VPN service for on-demand interconnection of their eScience resources (servers, VMs, laptops, storage, scientific instruments, etc.) that may only be reachable for their closed group. The developed CoCo prototype was validated in an SDN testbed and via Mininet simulation. Using the calibrated Mininet simulation the impact was analysed for larger scale deployments of the CoCo prototype.

1. Introduction

The advent of Software Defined Networking (SDN) is creating innovation opportunities for a wide range of use cases. A specifically promising application is the opportunity to create connectivity solutions in a more flexible way than with current Internet technology. To assess this opportunity and the maturity level of state of the art SDN technology, such as the OpenFlow protocol, the OpenDaylight controller and the Mininet simulation environment, an easy-to-use VPN service was developed and validated in the CoCo project. The CoCo project, that ran from October 2013 until March 2015, was one of the *open calls* projects funded by the European GN3plus project [1].

The demand for an easy-to-use VPN service has, for example, been identified in the eScience community. In eScience research the importance of networked services and facilities such as medical and genome databases, scientific instruments, visualization facilities, storage and cloud computing is increasing. However, current Internet services and facilities do not always provide the solutions that are required to meet the security and privacy requirements. It may require significant security configuration effort by network administrators as well as by users. Reducing the required configuration actions has been a long term endeavour for Internet researchers and significant steps have been taken. For example, in most intranet environments users already feel as if secure com-

* Corresponding author.
E-mail addresses: ronald.vanderpol@surfnet.nl (R. van der Pol), bart.gijsen@tno.nl (B. Gijsen), piotr.zuraniewski@tno.nl (P. Zuraniewski), d.f.romao@uva.nl (D.F.C. Romão), marijke.kaat@surfnet.nl (M. Kaat).

http://dx.doi.org/10.1016/j.future.2015.09.010
0167-739X/

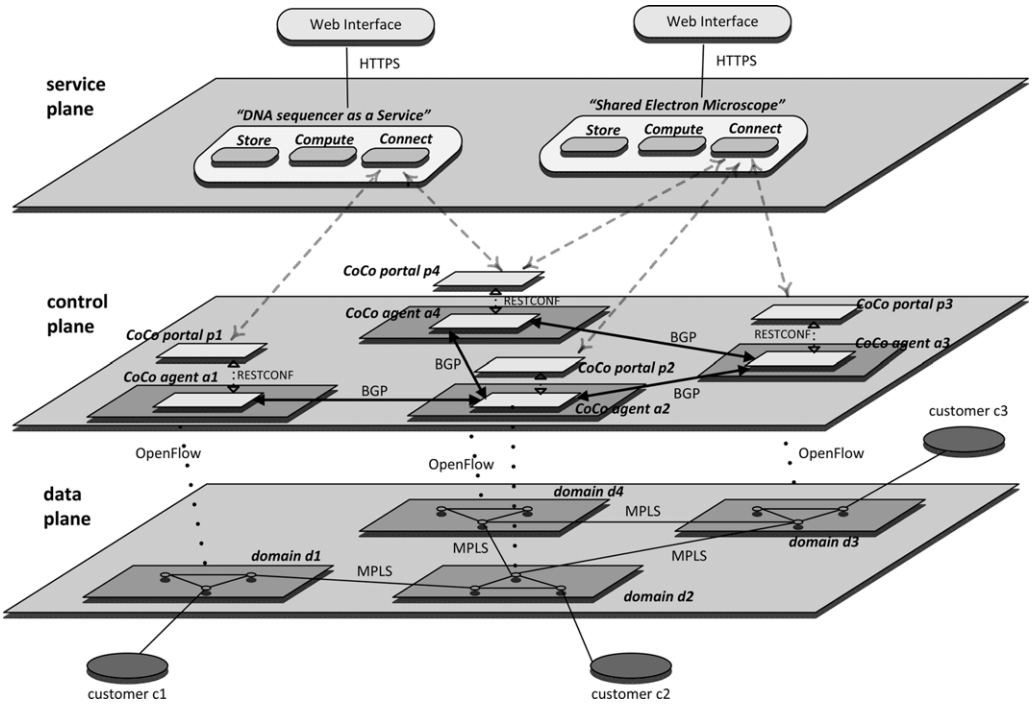

Fig. 1. CoCo Layered architecture.

munication comes out of the box and with HTTPS related technology secure client–server communication on the Internet has become child's play. A little more skilled user will be able to apply file encryption for securely sending its content to his peers. However, here is where current, user-friendly secure communication between end-users stops. More generic secure communication technology, such as Virtual Private Networks (VPNs) between multiple end-users, involves manual processes at one or more Network Operation Centers. This lack in secure communication technology leaves a variety of interesting applications out of scope. The need exists in particular in the eScience community as will be illustrated by the use case in Section 2.

The prototype developed in the CoCo project demonstrates how the OpenFlow protocol and the OpenDaylight controller can be used to create a new type of user configurable, on-demand, multi-domain and multipoint-to-multipoint VPN service. After a one-time initial general set-up of the CoCo service by the network administrator end-users will be able to set up and manage CoCo instances via an easy to use web portal, without having to rely on further manual intervention of the administrator. The programming interfaces of the CoCo prototype can also be exposed as API's to other applications, such that these other applications can automatically set up and tear down CoCo instances. The architecture for the CoCo service is presented in Section 3.

The CoCo prototype is developed on the SDN testbeds of SURFnet and the University of Amsterdam, that are equipped with OpenFlow switches and an OpenDaylight controller. During the development of the prototype several challenges in state of the art SDN technology were encountered, such as missing features that still need to be developed and bugs that need to be fixed. The solutions that were found for the CoCo prototype are described in Section 4.

In order to validate the CoCo prototype an automated test environment was developed for performing user-level experiments. In addition to the experimental validation in the SDN testbed a Mininet-based simulation environment was created. Mininet uses Open vSwitch as software OpenFlow switches, but in the test environment the same CoCo code was used and it also used OpenDaylight. The only difference was that software switches were used instead of hardware OpenFlow switches. With this setup Mininet simulations were run for validating the scalability of the CoCo prototype, beyond the boundaries of the number of OpenFlow switches and CoCo connected sites in the testbed. The results from these experiments and the scalability analysis are included in Section 5.

In Section 6 a discussion of the results is presented, including recommendations for following research. Finally, in Section 7 concluding remarks are presented about the ability of state of the art SDN technology to create connectivity solutions in a more flexible way than with current Internet technology.

2. Representative CoCo use case

A particular use case demonstrating the innovative power of the CoCo service is the *DNA sequencer as a Service* [2]. DNA sequencers are increasingly important instruments for scientists in the genomics science field. These sequencer instruments and the specific bioinformatics solutions required for the storage, processing and transport of their output are very expensive and get outdated relatively quickly, due to the current rapid developments. Therefore, research organizations can only justify such investments if the (re-)utilization of the sequencers and bioinformatics solutions is sufficiently high. The opportunity to strongly improve this return on investment by offering scientists from multiple institutes a DNA sequencing *as a* ,Service has been identified as a key innovation in the genomics research field.

Fig. 1 presents an overview of a technical solution for a DNA sequencer as a Service. The DNA sequencer as a Service is an example of a service plane extension of the CoCo service. The service plane is not part of CoCo, but can further facilitate the ease-of-use for the end user by incorporating the CoCo service in the control and data plane into an integrated service. Fig. 1 illustrates how authorized genomics experts can access a web interface of the DNA sequencer as a Service to set up the connectivity required for conducting an experiment.

Currently, the automation of DNA sequencing and processing is increasingly being applied via workflow management solutions.

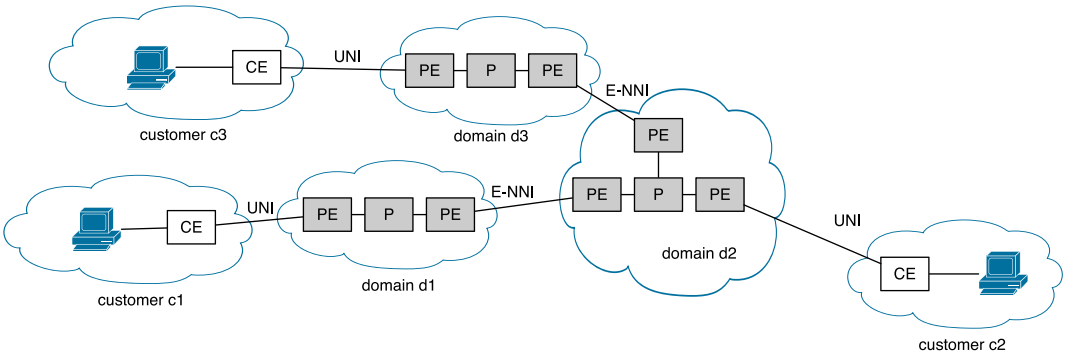

Fig. 2. Interdomain data plane forwarding.

At their back-end such platforms will manage and interface to the resources that are used to execute these processes. These resources include storage, processing and network connectivity. In this use case the CoCo service is foreseen to provide management services for the connectivity resources. In particular, once the workflow management solution needs to establish connectivity to a DNA sequencer in a different domain it can be programmed to use the CoCo portals in the relevant domains to create the required multi-domain connectivity. This way the genomics expert's actions can be limited to setting up the experiment, without logging-in on multiple systems and having to bother about establishing the required connectivity paths over and over again.

3. CoCo overall architecture

The exploration of use cases, such as the DNA sequencer as a Service, resulted in a number of design principles for the VPN service. The CoCo service should be a multipoint-to-multipoint VPN service, applicable in a multi-domain environment. A CoCo instance can be set up between several, up to tens of endpoints. The configuration of a CoCo instance is done by the end-users themselves and the time required to activate (changes to) a CoCo instance should take no more than 10 s. In other words, the CoCo service should be highly scalable, fast and very easy to use. The systems providing the CoCo service must be able to support multiple CoCo service instances simultaneously, i.e. end-user equipment can be part of several, different CoCo instances at the same time. Setting up and using the CoCo service should be affordable as only limited portions of research budgets are usually available for connectivity. The CoCo architecture should be easy to incorporate in the existing research and campus network infrastructures.

The CoCo prototype is developed using existing state-of-the-art open source SDN frameworks and open standards are used where possible. The CoCo prototype will consist of multiple domains and each domain has an OpenFlow based infrastructure. The data plane (forwarding plane) consists of OpenFlow switches. The switches in one domain are controlled by an OpenDaylight SDN controller and each domain runs its own CoCo agent. A CoCo agent is an extension to the OpenDaylight controller and adds specific CoCo functionality to the OpenDaylight controller. Fig. 1 shows the inter-domain architecture of CoCo.

OpenDaylight [3] is a community-driven open source SDN controller framework hosted at the Linux Foundation [4]. It is currently seen as one of the major open source SDN controllers. We have chosen to use OpenDaylight in the CoCo project because of this large and growing user and developer community.

A CoCo agent has several tasks. One task is to control the OpenFlow switches in its domain by doing topology discovery and configuring flow forwarding rules on the switches. The other task is in the inter-domain control plane of CoCo. The CoCo agents use the BGP protocol to exchange VPN and end-point information with each other. BGP is only used for exchanging information. BGP does not do any forwarding (FIB) manipulations. The forwarding tables in the OpenFlow switches are controlled via OpenDaylight.

The core of the network is based on MPLS label forwarding and is implemented over several domains. MPLS is only used as encapsulation. The forwarding paths are calculated by and provisioned from the OpenDaylight controllers. As such, no data plane label distribution protocols such as the Label Distribution Protocol (LDP [5]) are used.

In the control plane of each domain there is a single CoCo portal. End users can login to this portal directly, or via a service plane portal, and they can setup or tear down CoCo instances. CoCo instances are set up by choosing end-sites from a list and entering prefix and port based VLAN information for each site. End users can join multiple CoCo instances simultaneously. The web portal distributes the prefix and VLAN information to the CoCo agents in the various domains.

3.1. CoCo data plane forwarding

The CoCo networks core consists of OpenFlow switches that have either a Provider Edge (PE) function or a Provider (P) function. The PE switches connect to either Customer Equipment (CE) via a UNI interface or to PE switches in other domains via an E-NNI interface, as shown in Fig. 2. The P switches are internal core network switches. The terminology and concepts are specified in RFC 4026, *Provider Provisioned Virtual Private Network (VPN) Terminology* [6].

MPLS based forwarding is used in the core of the network in order to keep the forwarding tables small by aggregating all IP prefixes that are behind one PE OpenFlow switch. Two MPLS labels are used. The outer MPLS label is used to identify the PE switch to which a packet must be sent. The inner MPLS label is used to identify a particular CoCo instance (VPN). PE switches take care of tagging the user traffic received from Customer Edge (CE) equipment with the proper MPLS labels. When sending traffic from the core network to the CE switch, the PE switch removes the MPLS labels. Note that a site can be present in multiple CoCo instances at the same time. The inner MPLS labels identify the separate CoCo instances.

3.2. CoCo control plane

The CoCo agents are responsible for topology discovery within a domain and do intra- and inter-domain path calculation. The intra-domain path calculation is based on the domain topology only. The inter-domain path calculation is based on BGP path information that is exchanged with neighbouring domains. In the inter-domain case each CoCo agent configures forwarding entries on the OpenFlow switches in its own domain that form a path

between two PEs in that domain. This can be either two PEs directly connected to the source and destination CE, or it can be a PE that is connected to the E-NNI port to the inter-domain link that has been chosen by the BGP path selection process towards a CE in another domain (see Fig. 2). For simplicity, we start with one shortest path between each pair of PEs. At a later stage we can easily extend it with backup paths.

BGP is used to exchange information between the CoCo agents similar to what is described in RFC 4364 *BGP/MPLS IP Virtual Private Networks (VPNs)* [7]. There are BGP peering relationships between neighbouring CoCo agents. The CoCo agents also work with the concept of transit. For example, CoCo agent *a1* has a peering relation with CoCo agent *a2* only (see Fig. 1). CoCo agent *a1* exchanges information with CoCo agents *a3* and *a4* via CoCo agent *a2*, which in this case acts as a transit agent.

For each PE in its domain, a CoCo agent sends the following information to its CoCo BGP peers:

- *VPN-IPv4 address family (12 bytes) (RFC 4364* [7]*)*
  The 12 bytes consist of an 8 byte Route Distinguisher (RD) and a 4 byte IPv4 address. An RD is encoded as a 2 byte ⟨Type⟩ and a 6 byte ⟨Value⟩. We will use Type 2 RDs that consist of a 4 byte AS number followed by a 2 byte value. This value is managed by each domain, so by each CoCo agent. The CoCo agent manages a list of free values and assigns a unique value to each CoCo instance.
- *VPN-IPv6 address family (24 bytes) (RFC 4659* [8]*)*
  The 24 bytes consist of an 8 byte Route Distinguisher (RD) and a 16 byte IPv6 address. The RD will be the same value as for IPv4.
- *Next hop (VPN-IPv4 route with* RD = 0*)*
  The next hop in the route announcement should point to the PE that has the prefixes behind it. The CoCo agent assigns unique IPv4 addresses (from 10.0.0.0/24) to each PE in its domain to be used as next hop.
- *MPLS label to reach that PE (RFC 3107* [9]*)*
  This is done in the NLRI (Network Layer Reachability Information) by using an AFI (Address Family Identifier) of VPN-IPv4 and a SAFI (Subsequent Address Family Identifier) of 4. The NLRI is encoded as one or more triples of the form ⟨length, label, prefix⟩. The length is in bits and includes prefix and label(s). Each label is encoded as 3 octets, where the high order 20 bits contain the label value, and the low order bit contains *Bottom of Stack*. The prefix field contains address prefixes followed by enough trailing bits to make the end of the field fall on an octet boundary.
- *CoCo instance identifier (2 bytes) encoded in Route Target*
  A Route Target is sent via BGP Extended Communities (8 bytes) (RFC 4360 [10]) and is structured the same as a Route Distinguisher. We will use a Type 2 RD again with a 4 byte AS number and a 2 byte value. The value identifies the CoCo instance and will be used as inner MPLS label in the data plane for all traffic belonging to that CoCo instance.

Section 7 of RFC 4364 describes how PEs learn routes from CEs. This can be done by static configuration or by running a dynamic routing protocol, such as OSPF or BGP. When using OSPF, an OSPF instance per VPN is needed. BGP provides more control to the customer by using suggestions for route targets and/or using extended communities.

4. CoCo prototype

The CoCo prototype was developed on SURFnet's SDN testbed and an OpenFlow testbed set up by the University of Amsterdam. In the SURFnet testbed a Pica8 Pronto 3920 switch [11] was used, running PicOS 2.4 in Open vSwitch (OVS) Mode. Four OpenFlow bridges were configured on the switch. In the testbed of the University of Amsterdam a Pica8 P3290 switch was used running PicOS 2.1.3 with two bridges configured.

The CoCo architecture was designed to operate both in a single-domain and a multi-domain environment. The current CoCo prototype supports L3 VPNs only. There are many scalability and operational challenges in a L2 VPN service, especially in a multi-domain case. In Section 6.1 we describe the challenges of L2 VPNs.

4.1. CoCo data plane implementation

During development of the CoCo prototype it appeared that currently not all OpenFlow switch software supports the MPLS functions required for CoCo. Therefore, we use VLAN based port services on the UNI interfaces. This means that traffic between CE and PE is VLAN tagged and the VLAN ID maps to a particular CoCo instance. The customer is responsible for putting traffic of nodes in the correct VLAN. The PE matches on that VLAN ID and on the destination IPv4 prefix. Before forwarding the packet the PE switches add the two MPLS labels corresponding to the destination PE and CoCo instance. The CoCo agent that installs the flow forwarding rules on the PE switches needs to know what the destination PE for each IP prefix is. The CoCo agent also needs to know about the mapping between customer VLAN ID and CoCo instance and the IP prefixes that the customer uses in that particular CoCo instance. In the initial implementation of the CoCo prototype, the person adding a site to a CoCo instance configures this manually via the web portal.

4.2. CoCo control plane implementation

For the CoCo prototype the OpenDaylight SDN controller is used. As mentioned in Section 3, OpenDaylight is a community-driven open source SDN controller framework hosted at the Linux Foundation. The OpenDaylight project started in April 2013. The first version of OpenDaylight (Hydrogen [12]) was released in February 2014. The current release cycle for major releases is six months. Helium-SR2 [13] was released on January 30, 2015. During development of the CoCo prototype OpenDaylight Helium (SR1 and SR3) was running on VMs with Ubuntu 14.04, with the following features: odl-dlux-all, odl-openflowplugin-all and odl-restconf.

OpenDaylight consists of several modules, such as the controller platform, southbound plugins for protocols like OpenFlow 1.0 [14] and 1.3 [15], OVSDB [16], and NETCONF [17] and modules that offer services for OpenStack integration, a GUI, a DDOS protection module, etc. The controller platform is the central module and it has a Model-Driven Service Abstraction Layer (MD-SAL) based on YANG [18] models. It also has basic network service functions, such as a topology manager, a statistics manager, a switch manager, a forwarding and routing module and a host tracker. Most of these modules provide REST based northbound interfaces.

CoCo uses the northbound REST interfaces to retrieve the topology of the OpenFlow network and to manipulate forwarding entries on the OpenFlow switches. Parts of the VPN service project, which is part of the OpenDaylight Lithium release and due to be released in June 2015, will be used for the interaction with BGP.

To exchange the prefix reachability information between PE and CE switches we have chosen for static configuration for simplicity. The user enters the configuration data that is needed (e.g. IP prefixes) on the CoCo agent web portal when adding a site to a VPN.

4.3. CoCo portal

The CoCo portal is the user interface for the CoCo service and it is part of the CoCo agent. The CoCo agents are implemented as a web application running on Tomcat and use a Model-View-Controller

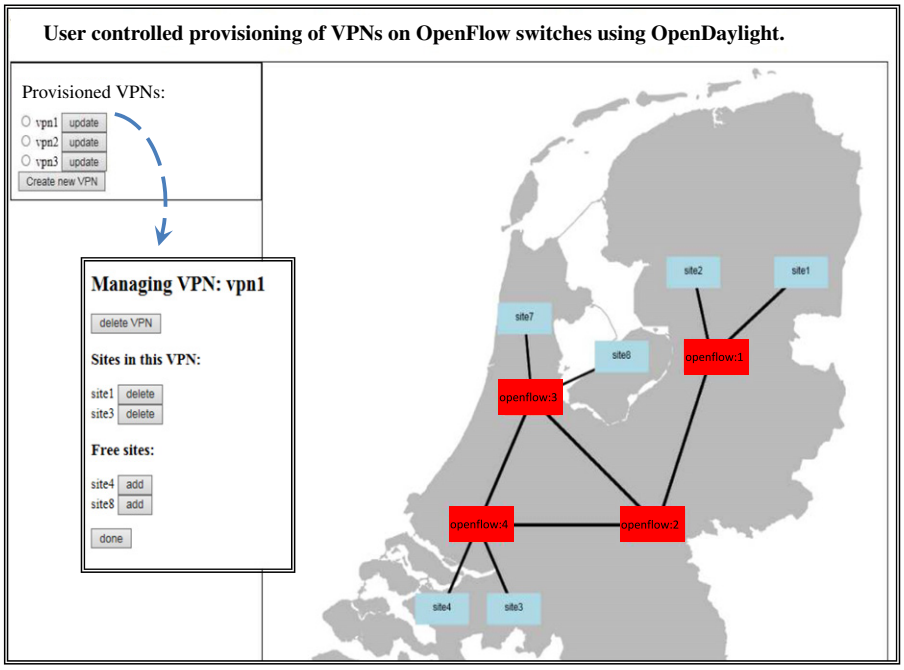

Fig. 3. CoCo portal screenshot.

architecture. The prototype is developed on a VM running Ubuntu 14.04 and Tomcat version 7.0.55. The development was done using Eclipse [19] with maven [20] and Java 1.7.

The user connects to the CoCo agent website with a browser and gets an overview of the network topology and the configured VPNs. See Fig. 3 for a screenshot of the portal.

The CoCo agent retrieves the topology information from the OpenDaylight controller via REST calls. The configured VPNs are stored in a MySQL database. On the portal the user can create a new VPN, add sites to it, delete sites from it and delete the complete VPN. But a user can only manipulate VPNs if he has the proper credentials. After logging in he has access to VPNs and sites in his group(s) only.

When a new VPN is created, the information is stored in the MySQL database. When a VPN is deleted, the information is deleted from the MySQL database. When sites are added to a VPN, the MySQL database is updated and OpenFlow forwarding entries are generated and sent to the OpenDaylight REST interface. OpenDaylight takes care of configuring the OpenFlow switches. A similar process takes place when removing sites from a VPN.

5. CoCo validation

We have performed a number of tests to verify interaction with the CoCo portal from the user perspective. Since we wanted to test various configurations (e.g., several different sets of sites participating in a CoCo instance), we have developed an automated test environment.

5.1. Test instruments

We have used AutoHotkey [21] to emulate interaction of an MS Windows user with the CoCo portal. In addition, we have developed several bash scripts for testing the connectivity (or the lack thereof) between virtual machines (VMs, Ubuntu 14.04) that are connected to our testbed and that can participate in CoCo instances. For this connectivity test each VM uses the nmap tool to send ICMP Echo Requests to all other VMs. To speed up this phase we have used the parallel-ssh tool to initiate tests on each of the VMs simultaneously. To facilitate the tests, we have used a 'Command & Control' (C&C) machine which can accept the instructions from MS Windows hosts and send the instructions towards the VMs. In this way we can emulate the behaviour of a large number of CoCo end users. To assure a proper sequence of the events during a test, bash scripts are started from a single command issued from the AutoHotkey script (i.e., from an MS Windows system) by using the plink [22] tool which connects to the C&C machine.

The tests with the CoCo prototype are performed in SURFnet's SDN testbed. In order to extend the scope of the test results beyond the limited scale of the testbed we also developed a Mininet [23] simulation environment. Mininet is a software emulator allowing the creation of a virtual SDN network with a configurable number of OpenFlow switches. These switches can either use Mininet's internal OpenFlow controller or use an external SDN controller, e.g. OpenDaylight. Moreover, Mininet switches use OVSDB (Open vSwitch Database, [16]) which is recognized as the de facto management protocol standard for SDN [24]. We used Mininet version 2.2.0 to emulate the OpenDaylight controller and the CoCo prototype software in the Mininet environment. Configuration and validation scripts to setup a Mininet-based simulation for the CoCo service have been published by the CoCo project [25].

5.2. Experiments

As an input, the AutoHotkey script is fed with the type of action to be tested (e.g., setting-up the CoCo instance, adding a site to a CoCo instance or disabling a CoCo instance) along with appropriate arguments (e.g., a list of the sites being members of the CoCo instance to create) and the number of test repetitions. In the CoCo project a number of test cases were performed to validate (a) the functionality of the CoCo prototype, (b) the ease of use and (c) its scalability.

The ease of use test starts with opening the CoCo portal in a web browser, clearing all existing CoCo instances and verifying that there is no connectivity (yet) between the VMs under test. After assuring that each VM can only reach itself, the next step is to select other sites on the CoCo portal, to submit the request to

the OpenDaylight controller and check the connectivity again. In order to get an indication for the CoCo instance activation times we measured the elapsed time, and repeated this test sequence about ten times.

The measurement results from these repeated tests indicated a very limited variability in the elapsed activation times. The coefficients of variation (ratio of standard deviation to the average value) being in the order of 0.02 and therefore we can suffice with reporting the average elapsed times. The elapsed time between pressing the *done* button (i.e., sending an order to actually create a CoCo instance) until a successful connectivity measurement result took approximately 3.3 s. This includes 1 s 'sleep' time in the script and 1.45 s for the nmap full-mesh connectivity test. In general, we can conclude that the CoCo instance is ready in roughly 1 s. In total, the described session (from opening the web browser with the CoCo portal until closing it after completing all the mentioned tests and actions and including some artificially introduced sleep-time) took approximately 13 s. Excluding the initial CoCo instance clearing and the test of its effect the session time was below 10 s. According to Nielsen's [26] de facto usability criteria for interactive web applications this CoCo instance activation time will be perfectly acceptable from a user's point of view.

This validation of elapsed CoCo instance activation times was performed on VMs that are connected to four CoCo sites in the testbed. To analyse the impact of elapsed activation times in larger scale SDN networks we used the Mininet simulation environment.

5.3. Scalability tests

While in principle it is possible to perform large scale tests using a physical testbed, it may require some tedious tasks such as rewiring when topology changes. Using a virtual testbed is more convenient, offering rapid generation of different setups. We have used the Fast Network Simulation Setup (FNSS) toolchain [27] to generate various networks which in turn can be easily exported to Mininet. Any topology created in this way is then automatically retrieved by the OpenDaylight controller as described in Section 4.2. To function correctly, CoCo needs additional information stored in its MySQL database such as MPLS labels or sites configuration. We populate this database using a custom Python script which processes Mininet topology information along with a MySQLdb [28] module for the actual export. Finally, for these tests we have decided not to use AutoHotkey to simulate user clicks needed to add sites to a VPN, but rather supply the required configuration directly to a little helper function in CoCo itself. Therefore, our focus is on CoCo instance setup time understood as described in the previous section, i.e., as a time which elapses from sending an order to actually create a CoCo instance. A successful creation is each time confirmed by performing connectivity tests as described Section 5.1.

For our experiments we have chosen to simulate a two-tier data centre topology: switches are organized in two tiers (core and edge) with each core switch connected to each edge switch while each host (in our case: CoCo site) is connected to exactly one edge switch. We keep the number of core switches equal to two while increasing the number of edge switches.[1] Each of the edge switches connects two sites and one of them is selected to participate in a CoCo instance. In this setup we therefore expect a quadratic complexity with the number of sites as paths between each pair of sites need to be calculated and inserted into the appropriate switches. We have used Ubuntu 14.04 virtual machine with 2 cores clocked at 3.5 GHz and 32 GB of RAM.

The results are summarized in Table 1 and Fig. 4, confirming quadratic dependency. We have performed 10 repetitions for each selected number of nodes using Java *System.nanoTime()* to measure the elapsed time. The first observation is that in each test a first run always lasted roughly half of the typical runtime, despite clearing configuration after each run. We have left it to future work to investigate where this difference comes from. We have decided to treat the first value as an outlier and remove it from further processing. Therefore mean (and its plot in Fig. 4), median, standard deviation and coefficient of variation in Table 1 are calculated for runs 2–10. Since the obtained times were stable we decided that 9 effective repetitions are sufficient. For number of sites less than 10 the setup time is less than 10 s. *On the other hand, for 20 participating sites our system needed about 1 min to converge.* We have observed that for 15 and 20 sites the controller sometimes failed to install a flow, resulting in lack of connectivity between certain sites. The OpenDaylight console showed exceptions related to concurrent execution, but were not able to find a complete explanation. We plan to repeat our tests using the next release of OpenDaylight (Lithium) and verify if this particular problem still persists. We conclude that a virtual testbed is capable to verify various scenarios using the same tools as in a production network. The added value is the speed of creating new network topologies and ease of prototyping at minimal (or no) investments.

Table 1
Elapsed times for various network configurations. Result of the first run considered as outlier and removed in further calculations. Times in ms.

No. of sites	2	4	6	8	10	15	20
Run							
1	252	981	2159	5128	6689	14579	28617
2	428	1788	5427	9429	12675	29586	52150
3	469	1897	4752	8456	12900	29278	57020
4	426	2213	4426	8255	12593	30893	52709
5	459	2083	4281	7763	12640	29299	53875
6	407	1915	4663	8330	12085	30186	56715
7	472	1843	4642	8109	12660	28916	54152
8	533	1787	4188	7683	13106	29562	54673
9	415	2179	4508	8005	13159	32962	56703
10	402	1795	5610	7622	12620	29125	54052
mean	446	1944	4722	8184	12715	29979	54672
median	428	1897	4642	8109	12660	29562	54152
stdev	40	160	461	520	301	1198	1677
CV	0.09	0.08	0.10	0.06	0.02	0.04	0.03

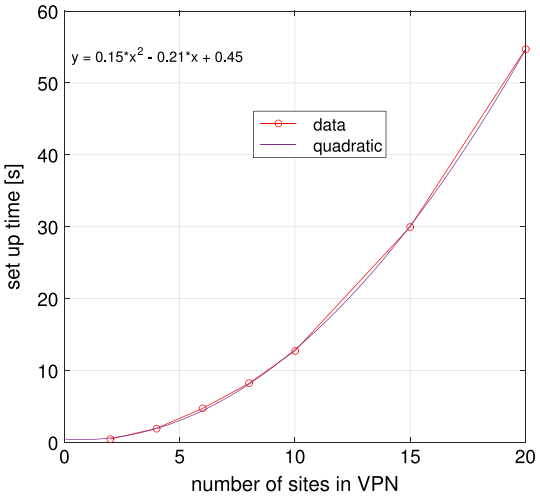

Fig. 4. Setup time vs. number of sites.

6. Discussion

The underlying objective of developing the CoCo prototype was to assess the maturity level of state of the art SDN technology. To this end our experiences during the CoCo development are

[1] Note, that we do not aim to replicate an optimal network architecture here with acceptable oversubscription ratios, etc. but rather want to verify the VPN setup process itself.

discussed in this section. These experiences are distinguished in (a) the limitations of SDN technology that were encountered and (b) challenges that are yet unresolved for the CoCo prototype.

It should be noted that this is a reflection of the state of the art during the development in the period of 2014 and the beginning of 2015. Of course, the technology has not stopped evolving since then. Nevertheless, we regard it valuable to determine the maturity level for that period of time.

6.1. L2 VPN challenges

The presented CoCo prototype offers a L3 VPN service. We explored the extension of the CoCo prototype to an easy to use, on-demand L2 VPN service. Creating such a service with SDN technology requires the implementation of a number of basic L2 functions. In order to implement these functions some scalability challenges need to be addressed, especially in the multi-domain case. A L2 VPN service requires two major additional features:

- MAC learning (or an equivalent mechanism) is needed at the PE edges.
- The network must support broadcast.

As opposed to the case for a L3 VPN service, there is no routing and all nodes at all sites in a L2 VPN reside in the same L2 network. Therefore, they share the same IP prefix and each node must get a unique IP address from that prefix.

One option is to use DHCP(v6). This requires either the provisioning of addresses from one big pool, or to divide the available address space over the CoCo sites. The latter option is more difficult to manage, as it is usually hard to predict how many addresses are needed at each site.

For IPv6 there is the option to use SLAAC (Stateless Address Autoconfiguration) [29]. This requires a mechanism for ICMPv6 prefix announcement and discovery. However, this traffic does not need to traverse the backbone, as it is only needed at the edges. It is probably best to implement this at each site by using a proxy mechanism that intercepts discovery messages before they enter the backbone.

When an Ethernet frame coming from a CE device reaches a PE edge device there are two possibilities. When the destination MAC address is known, the frame can be forwarded on the proper interface. When the destination MAC address is not known, a L2 VPN service would usually broadcast that frame towards all destination PEs. This requires setting up broadcast support in the backbone network. A technology like VPLS implements this with setting up a full mesh of circuits between all PEs. This has clearly scalability issues. Recent work in the IETF L2VPN working group tries to solve this scalability problem with the EVPN specifications (RFC 7432 [30]). With EVPN, MAC/IP address information is explicitly exchanged between PEs via the BGP protocol. This way there are no unknown MAC destinations anymore.

When a host in one site wants to communicate with a host in another site, it needs to know its IP address. This is done via the ARP and ND (Neighbor Discovery) protocols. These messages are broadcast (multicast) and usually the destination node responds with a reply. In a L2 VPN service this requires broadcast (multicast) support in the backbone. Another option is to use proxy ARP (and ND). At the edge the ARP (ND) requests are intercepted and a proxy (that knows the destination IP address) replies to the sending node. This way the ARP (ND) messages never enter the backbone.

6.2. Mininet test environment

In traditional communication networks the use of a separate test and development environment, besides the operational production network, is common practice. The advent of SDN offers the opportunity to reduce the complexity and manual labour in the physical test environment by using a simulation environment that has identical control plane software installed. As mentioned before, the CoCo service is intended to be easy to use by its end-users, however, rolling out such a new service will require some initial efforts from the network administrators. Fortunately, there are possibilities to validate and tweak the CoCo service in a separated environment before deploying it in a production network. One of the options is to use Mininet, which can run on a single PC. It should be noted however, that Mininet is a software emulator that has certain limitations. For example, limitations with respect to attainable speed transfers, such as identified in [31]. In addition, Mininet is a relatively new simulation environment with a growing community and feature set. In our tests we have encountered certain problems with for example port numbering (resolved in version 2.2.0) or somewhat crude sshd support (improving it is high on the official Mininet project ideas list). Despite such limitations and minor issues Mininet can be regarded as a good option (but not the only one, see for example [32]) for experimenting with SDN, complementing the facilities in a physical testbed.

6.3. Further CoCo development

The prototype of the CoCo service presented in this paper still lacks a number of features. For example, the implementation of authentication (and other AAA functions) is still very basic and the necessary extensions to make the prototype applicable over multiple domains are currently under development. The absence of these features in the presented prototype are not due to limitations in SDN technology, but are due to the higher priority that we gave to implementing the features that are included in the current CoCo prototype.

Resolution of conflicting CoCo requests is also a feature that is still missing. For example, it may occur that multiple users are adding or deleting CoCo sites and devices to the same CoCo instance at the same time. For such potentially conflicting user actions there is no mechanism in the current prototype that enforces the correct processing of these actions. However, we anticipate that existing concurrency mechanisms can be re-used to handle conflicting requests in the CoCo prototype.

7. Conclusion

We were able to implement a working Community Connection (CoCo) prototype. This prototype has been successfully validated and a Mininet setup was used for scalability tests. During the development we discovered that the SDN technologies we used are still rapidly evolving. Implementing the CoCo components gave us a lot of insight into the maturity of SDN, its possibilities and weaknesses.

Acknowledgements

This paper has been produced with the financial assistance of the European Union. The contents of this document are the sole responsibility of SURFnet, TNO and UvA and can under no circumstances be regarded as reflecting the position of the European Union.

The authors acknowledge help from MichałGoliński in CoCo code development.

References

[1] CoCo on Demand Community Connection Service for eScience Collaboration. http://www.geant.net/opencall/SDN/Pages/CoCo.aspx.
[2] Intermediate Use Cases for the Community Connect (CoCo) Service. http://www.geant.net/opencall/SDN/Documents/CoCo_intermediateusecases_deliverable.pdf.
[3] OpenDaylight. http://www.opendaylight.org/.
[4] Linux Foundation. http://www.linuxfoundation.org/.
[5] L. Andersson, I. Minei, B. Thomas, LDP Specification, RFC 5036, 2007. http://tools.ietf.org/html/rfc5036.
[6] L. Andersen, T. Madsen, Provider Provisioned Virtual Private Network (VPN) Terminology, RFC 4026, 2005. http://tools.ietf.org/html/rfc4026.
[7] E. Rosen, Y. Rekhter, BGP/MPLS IP Virtual Private Networks (VPNs), RFC 4364, 2006. http://tools.ietf.org/html/rfc4364.

[8] J. De Clercq, D. Ooms, M. Carugi, F. Le Faucheur, BGP-MPLS IP Virtual Private Network (VPN) Extension for IPv6 VPN, RFC 4659, 2006. http://tools.ietf.org/html/rfc4659.
[9] Y. Rekhter, E. Rosen, Carrying Label Information in BGP-4, RFC 3107, 2001. http://tools.ietf.org/html/rfc3107.
[10] S. Sangli, D. Tappan, Y. Rekhter, BGP Extended Communities Attribute, RFC 4360, 2006. http://tools.ietf.org/html/rfc4360.
[11] Pica8 Pronto 3920 datasheet. http://www.pica8.org/documents/pica8-datasheet-64x10gbe-p3780-p3920.pdf.
[12] OpenDaylight Hydrogen. http://www.opendaylight.org/hydrogen.
[13] OpenDaylight Helium-SR2. http://www.opendaylight.org/software/downloads/helium-sr2.
[14] OpenFlow Switch Specification version 1.0.0, December 31, 2009. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.
[15] OpenFlow Switch Specification 1.3.4, March 27, 2014. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf.
[16] B. Pfaff, B. Davie (Eds.), The Open vSwitch Database Management Protocol, RFC 7047, 2013. http://tools.ietf.org/html/rfc7047.
[17] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network configuration protocol (NETCONF), RFC 6241, 2011. http://tools.ietf.org/html/rfc6241.
[18] M. Bjorklund (Ed.), YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF), RFC 6020, 2010. http://tools.ietf.org/html/rfc6020.
[19] Eclipse. https://eclipse.org/ide/.
[20] Maven integration for Eclipse. https://maven.apache.org/eclipse-plugin.html.
[21] AutoHotkey (AHK). http://ahkscript.org/.
[22] Plink. http://www.chiark.greenend.org.uk/~sgtatham/putty/.
[23] Mininet. http://www.mininet.org/.
[24] https://wiki.opendaylight.org/view/OVSDB_Integration:Design.
[25] Mininet simulation of CoCo prototype. https://github.com/rvdpdotorg.
[26] Jakob Nielsen, Usability Engineering, Academic Press, Inc., ISBN: 0-12-518406-9, 1993.
[27] L. Saino, C. Cocora, G. Pavlou, A Toolchain for simplifying network simulation setup, in: Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, SIMUTools 2013, Cannes, France, March 2013.
[28] A. Dustman, MySQLdb: a Python interface for MySQL. http://mysql-python.sourceforge.net/.
[29] S. Thompson, T. Narten, T. Jinmei, IPv6 Stateless Address Autoconfiguration, RFC 4862, 2007. http://tools.ietf.org/html/rfc4862.
[30] A. Sajassi, R. Aggarwal, N. Bitar, A. Isaac, J. Uttaro, J. Drake, W. Henderickx, BGP MPLS-based Ethernet VPN, RFC 7432, 2015. http://tools.ietf.org/html/rfc7432.
[31] What are Mininet's limitations? https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what-are-mininets-limitations.
[32] ns-3 vns-3-dev documentation: OpenFlow switch support. http://www.nsnam.org/docs/release/3.13/models/html/openflow-switch.html.

Ronald van der Pol, M.Sc. He has been working in the field of Education and Research Networks for more than twenty years. Since 2012 he is working on network innovation projects at SURFnet. His former employers include VU University Amsterdam, SURFnet, NLnet Labs and SARA. His current interests are in new network technologies and how these can be applied to next generation networking. This includes Software Defined Networking, OpenFlow, multipathing, load balancing and other forms of traffic engineering and optimization, network management and monitoring, transport protocols, carrier Ethernet and end-to-end performance of demanding applications. He holds masters degrees in both Physics and Computer Science and is a frequent speaker at networking conferences.

Bart Gijsen, M.Sc. After receiving his M.Sc. degree in both computing science and mathematics Bart joined KPN Research in 1996 as a researcher in the field of performance analysis of Internet technology based information and communication systems. From 2003 till present day Bart is employed at TNO as a senior innovator in the expertise group Performance of Networks and Systems. One of his focus areas is quantitative modelling and impact prediction of Internet security and stability. Among the 20+ journal and conference publications about performance and robustness of information and networking technology was his work on 'A Global Reference Model of the DNS', for which he received the best paper award. Since 2010 Bart is also part-time director of the Dutch ICT Innovation Platform 'Critical ICT infrastructures'.

Piotr Zuraniewski, Ph.D. He received his M.Sc. degree in mathematics (Best Graduate Award) in 2003 from AGH, Poland, where he further worked as a research assistant, being involved in several FP6/FP7 research projects. In 2011 he defended his Ph.D. thesis entitled 'Stochastic modelling and control of communication networks' at the University of Amsterdam. He is now with the Performance of Networks and Systems group of TNO, working as a research scientist. His research interests are Data Science, performance evaluation, anomaly detection especially in the context of new network technologies like SDN of NFV. Dr. Zuraniewski is a co-author of several refereed scientific papers, published in international journals. Since 2004 he is recognized as a Cisco Certified Network Professional (CCNP).

Daniel Filipe Cabaça Romão, B.Eng. He received his bachelor in electronics, telecommunications and computer engineering in 2012 from ISEL, Lisbon. Prior to his current job, Daniel was involved in ULOOP(FP7), and he researched secure bootstrapping methods for connecting to wireless networks at Caixa Mágica, Lisbon. Currently he is a part-time Scientific System Engineer for the System and Network Engineering research group at the University of Amsterdam. In his current position, he is responsible for network and system administration as well as developing extensions for cloud management systems and participate in research projects. Also, he is pursuing a Master degree in System and Network Engineering. His research interests are: SDN/NFV, Cloud Computing, and security of networks.

Marijke Kaat, M.Sc. She received a master's degree in Computer Science from the University of Amsterdam. She has been working for over twenty years in the field of Education and Research Networks. Currently, she is a Network Manager at SURFnet, with a focus on network innovations and sustainability. She participated in several global and European standards and operational fora (IETF, RIPE) and GÉANT research projects. Her current interests are new network technologies and architectures such as SDN, RINA and NDN, but also new developments in network management, monitoring and measurements and the sustainability and greenness of networks. She is also a part-time lecturer and researcher in the System and Network Engineering group at the University of Amsterdam.