Emulation of SDN-Supported Automation Networks

Peter Danielis, Vlado Altmann, Jan Skodzik, Eike Bjoern Schweissguth, Frank Golatowski, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany, Tel./Fax: +49 381 498-7277 / -1187251
Email: peter.danielis@uni-rostock.de

***Abstract*—Software-defined networking (SDN) is a principle for the flexible configuration of networks, which recently has aroused an increasing interest of researchers and companies. Today, the operation and configuration of networks as well as their adaptation to changing requirements represent a major challenge if legacy network management protocols are used due to their inability to provide network-wide configuration. Compatibility issues with legacy protocols or even proprietary protocols have further contributed to hardly reconfigurable and inefficient networks and thus prepare the way for the new concept called SDN. SDN implementations are mainly in the data center today. Implementations will find their way into broader networking applications over the next few years. However, even automation networks with highest QoS requirements can benefit from SDN support by achieving significantly more devices that meet these requirements. Therefore, this paper addresses the emulation of SDN-supported automation networks to examine possible design options. Limitations and possibilities when using the popular and widely accepted SDN emulator Mininet are analyzed.**

I. Introduction

Software-defined networking (SDN) is a novel approach for the flexible configuration of networks, which recently has attracted growing interest of both researchers and companies. If network operators want to implement high-level network policies today they need to configure each individual network device separately with low-level and often manufacturer-specific commands [1]. Compatibility issues with legacy network management protocols like Simple Network Management Protocol (SNMP) or proprietary protocols have further contributed to hardly reconfigurable and inefficient networks. SDN was developed to allow for the easy configuration and operation of networks as well as their adaptation to changing requirements. In this regard, OpenFlow represents a protocol that has been widely accepted for the run time configuration of network hardware by means of an SDN controller [2]. It aims at easing the transition from research to practical applications of SDN as it is an open standard that is supported by the network hardware of many manufacturers.

So far, SDN implementations can predominantly be found in data center environments. Undoubtly, SDN will gain increased entry into broader networking applications in the very near future [3]. We already want to go a step further by introducing SDN into automation environments with strict Quality of Service (QoS) requirements such as hard real-time (RT) capability [4]. We derive our motivation from the fourth industrial revolution, which is currently taking place. Its objective consists in paving the way for the Internet of Things into the factory resulting in the so-called Smart Factory or even an Industrial Internet of Things, Data, and Services. The US-American company General Electric (GE) recently initiated a comprehensive research initiative called "Industrial Internet" [5] and in Germany the term "Industry 4.0" has been coined [6]. GE forecasts for the future that there will be more intelligent devices, which have to be connected to interact with each other dynamically. Therefore, improved automation structures are necessary to manage the high complexity arising from the increasing number of devices. Furthermore, the management of a network consisting of thousands of devices is a technical challenge requiring tools and technologies to be able to meet this challenge. As soon as the number of devices increases from 100 to 1,000 or even 10,000, the device networking technology must be ready for this magnitude.

To conclude, devices have to be interconnected in industrial facilities to communicate with each other and prospectively, their number will strongly increase in the described Internet of Things, Data, and Services while still requiring RT behavior. In this regard, Industrial Ethernet (IE) systems are the latest communication technology, which provides guarantees for the arrival of an Ethernet frame and its delivery time and thus represent a RT capable device networking technology for automation networks [7]. However, the established IE system solutions can especially not fulfill the future challenges in terms of scalability and flexibility as they are right now [8]. In this respect, future large-scale automation networks can benefit from SDN support by achieving more parallel communication and thus SDN support allows significantly more RT capable devices in an automation network.

Therefore, this paper addresses the emulation of SDN-supported automation networks to examine possible design options. Experiments are carried out with the popular and widely accepted SDN emulator Mininet. Mininet's limitations and possibilities are analyzed when emulating automation networks with hard RT capability. To the best of our knowledge, our publication is the first paper, which analyzes the impact of SDN-support on automation networks by means of emulation.

The remainder of this paper is organized as follows: Section II explains SDN basics. Section III analyzes how SDN is suited to build SDN-supported automation networks. Section IV addresses the emulation with Mininet using the example of the SDN-supported RT Ethernet automation network called HaRTKad and evaluates results. Section V contains a comparison with related work. The paper concludes in Section VI.

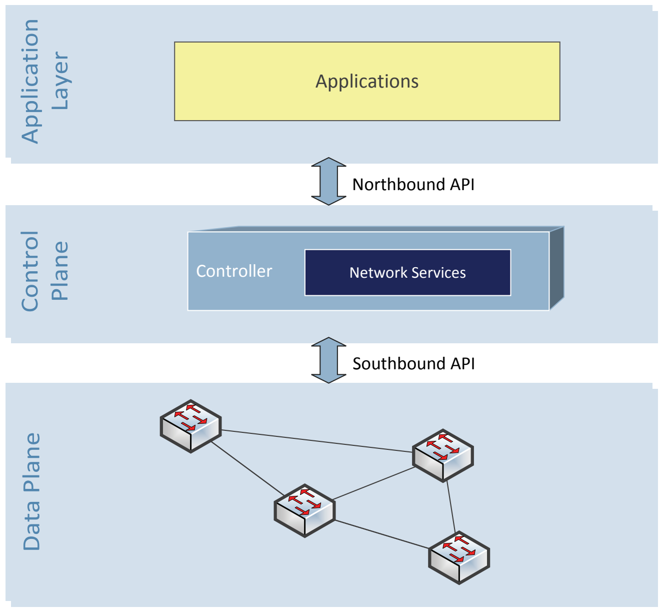

Fig. 1. Relationship between control plane, data plane, and applications [1].

II. Basics

A. The SDN principle

In today's computer networks, there is a variety of different components with dedicated tasks, which are designed to ensure an efficient and transparent communication between connected devices. These components are, e.g., switches, routers, firewalls, and load balancers. Each of these network devices has its own control logic to analyze packets and decides how they should be changed and forwarded. The control logic of all network devices can be regarded as control plane while the hardware for packet forwarding and modification is denoted as data or forwarding plane. In the typical established networks, the logic of the control plane is distributed throughout the network by implementing it in the network devices and it is directly connected to the hardware of the data plane. Contrary, the two basic characteristics of SDN are the separation of control plane and data plane and the merging of the control logic of the control plane in a controller software that controls devices of the data plane. The controller (also referred to as network operating system) communicates with the devices of the data plane via a well-defined API and can also provide an API to communicate with the applications running on hosts [3], [9]. This concept is apparent from Figure 1. According to this depiction, the APIs are also called southbound API (towards data plane) and northbound API (towards the applications) [1]. It should be noted that the communication with the applications via the northbound API is not mandatory for a working SDN. However, depending on the application a communication between controller and application can bring considerable benefits and is therefore considered an important research topic [10]. One of the currently most noted southbound APIs is the OpenFlow protocol.

B. OpenFlow

With OpenFlow, researchers can test new protocols in common networks with realistic size [2]. Hardware manufacturers can support OpenFlow on their devices without significant changes to their hardware as the implementation of OpenFlow switches imposes requirements that are already met by conventional switches and routers [3]. The prevalence of OpenFlow results from these properties. The two core components of OpenFlow are the specification of the OpenFlow-enabled switches and the definition of the API for communication between OpenFlow switches and controller software.

Switch: The classification and processing of packets in OpenFlow-enabled switches is done on the basis of so-called flows. A flow is a particular connection in the network, which can be specified with different granularity. The entire flow definitions are stored in a flow table. An entry in the table consists of a packet header (defines the flow), actions to be taken (processing of packets of this flow), and statistics (previous number of packets of this flow, the time since the last packet of the flow, etc.). For each incoming packet, the flow table is searched for a matching entry. These flow tables can be implemented using Ternary Content Addressable Memory (TCAM). Because of this hardware-based search, the packet processing based on flows is performing well enough in order to not limit the speed of the Ethernet connections. In addition, TCAMs are already used in conventional switches and routers and thus the rapid adoption of the concept by hardware manufacturer is enabled (see [2]).

Controller: If no suitable flow entry exists for an incoming packet in an OpenFlow switch, it is forwarded to an OpenFlow controller. The OpenFlow controller then decides about the further processing of the packet. It can either instruct the switch how to handle the package (if necessary carry out modifications, forward to a specific port, discard) or it can create a matching flow entry for the packet, which determines the processing for both the current and future packets of the flow. Issuing a request to the controller results in a relatively high delay since the controller is typically implemented in software. For the network to provide high performance, it is therefore desirable to create flow entries (see [2]).

C. Mininet

Structure and functionality: Mininet allows to emulate virtual networks with SDN and OpenFlow capabilities under Linux [11] and has been developed by Brandon Heller [12]. A detailed description of the objectives, the operation, and the limitations of Mininet can be found in [13], [12], [14]. Mininet provides a convenient framework to facilitate working with virtual networks. It provides a command line program, which can generate corresponding networks. Another feature of great importance to carry out experiments is Mininet's Python API. Through the Python API, own topologies can be defined and the configuration of individual virtual hosts, switches, and Ethernet connections can be carried out. Mininet relies on the emulation of a network. Thereby, many advantages of simulators and hardware-based test environments are combined while compensating their weaknesses: Contrary to simulators, a comprehensive abstraction of system components is avoided as an abstraction could lead to functional differences compared to real hardware. Instead, virtual software components are used in the emulation whose behavior is identical to the real hardware from a functional perspective. This allows to run real code on the network-connected virtual hosts in the same way as

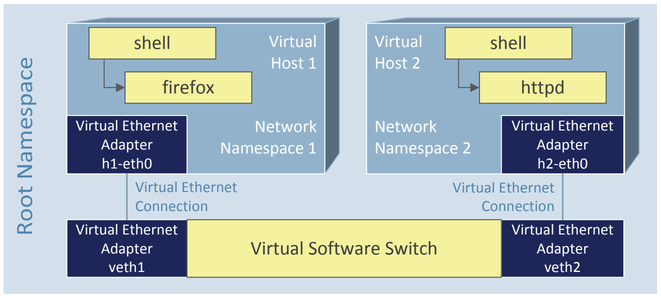
Root Namespace
shell
firefox
Virtual Host 1
Network Namespace 1
Virtual Ethernet Adapter h1-eth0
Virtual Host 2
shell
httpd
Network Namespace 2
Virtual Ethernet Adapter h2-eth0
Virtual Ethernet Connection
Virtual Ethernet Connection
Virtual Ethernet Adapter veth1
Virtual Software Switch
Virtual Ethernet Adapter veth2

Fig. 2. Network design with Mininet [17].

one would use it in a hardware-based test setup. Accordingly, the traffic pattern used in the emulation equals that of the real test environment. However, in contrast to a real testbed and similar to a simulator, the advantages of high flexibility (e.g., in terms of network topology) and low costs are retained. The emulation is enabled by options, which are included in Linux by default such as the virtualization of switches, Ethernet adapters, and hosts. For a better understanding, a simple example of the concept is apparent from Figure 2.

- Virtual Ethernet adapters and connections: The ip link command allows the generation of virtual Ethernet connections with a virtual adapter at each end.
- Virtual switches: In Mininet, the Linux package Open vSwitch is used to create virtual OpenFlow-enabled software Ethernet switches. The Mininet software currently only supports OpenFlow version 1.0.0 so that this version has been used as a reference [15]. One of the adapters of the virtual Ethernet connection is assigned to the switch.
- Virtual hosts: A resource-saving measure for the virtualization of network hosts are namespaces. This is a Linux concept to provide each process with a network stack, which is independent of the rest of the system. That is, for the corresponding processes independent routing and firewall settings are possible and only network adapters explicitly assigned to the respective network namespace are visible [16]. After creating a network namespace, a shell process is started in this namespace, which represents the virtual host. All programs called from there are then in the same network namespace (e.g., firefox and httpd in Figure 2). Despite their own view of the network resources, a virtual host has full access to the file system of the Linux system and can start programs in the network namespaces, which are needed in experiments to generate the desired traffic pattern.

Boundaries and restrictions: The fundamental problem of Mininet (and emulators in general) is to emulate the correct time sequence of events and thus to correctly reproduce the performance characteristics of a system. This problem has already been mentioned as the most important limitation of Mininet in [14] and is analyzed in more detail in a subsequent paper [13] by the same authors. The reason of the problems with temporal correctness is the operation of a virtual network on a single physical PC. In a real network, switches and hosts operate in parallel and independently. In contrast, in the emulation originally parallel operations are performed sequentially due to the limited number of CPU cores. In addition, computationally intensive processes on a host can affect the performance of the entire emulated network as the scheduler of the operating system no longer adequately provides the software switches or other hosts with sufficient computing time to perform the processing and forwarding of packets.

Accordingly, the time response of the system may change significantly. Therefore, recent research works such as [13] have formulated a main criterion that must be met so that an experiment can still provide useful results in the emulation. The *criterion* is that the experiment must be limited by network resource constraints such as bandwidth or latency but not by the CPU or memory bandwidth [13]. In addition, there are *two important conditions*, which must be met so that an experiment carried out with the emulator can produce correct results compared to those of a real testbed:

1) Virtual network accuracy: All packets need to be transmitted always at a certain point in time in order to ensure the set bandwidth in the emulation (e.g., a 1514 byte packet needs to be transmitted every 121.1 μs over a 100 Mbit/s link). However, the scheduler and CPU performance of the emulating system may not be sufficient to ensure too high transmission rates for all virtual connections. If that is the case all virtual connections must be limited to a certain bandwidth to provide sufficient time for scheduling and CPU processing on the virtual hosts.
2) Virtual host accuracy: All virtual hosts must always show a small fraction of idle CPU time as this indicates that the experiment is not CPU limited, i.e., a virtual host is not starved of CPU resources.

For a variety of experiments, results were generated, which are comparable to those of a real testbed. Many experiments had to be scaled down in terms of bandwidth in the network as the hardware setup comprised 1 Gbit/s or even faster connections, which would have violated the first condition stated above [13]. However, in this paper we will show that the criterion and conditions stated above are still not sufficient to allow for a precise emulation of automation networks.

III. Towards SDN-Supported Automation Networks

In this section we investigate how SDN and OpenFlow can be used to optimize networks for a particular application. First, the general options for application-oriented control of a network is discussed before evaluating, which advantages and disadvantages the use of SDN in automation networks entails.

A. *Application-oriented SDN controller:*

The OpenFlow controller is responsible to find correct routes between connected devices and to use the network topology as efficiently as possible in a network. Found routes should ideally be installed in the flow tables of the switches as the network is otherwise slowed down by forwarding all packets to the controller. The communication with the switch is done through the standardized OpenFlow protocol. In addition, however, there are no specifications for the implementation of the controller. The complexity of the controller can thus range from layer 2 switches through traditional layer 3 routers to customized controllers. Thereby, the behavior of the network can be adapted in a simple and inexpensive manner by the user. Special hardware or support from the manufacturer of networking hardware are not required. The controller can be programmed taking into account the existing network topology and the expected traffic patterns. Generally, there are two possible approaches for the optimization of a controller to a specific application:

1) Static scenario: Network topologies and application traffic patterns are known at the time of programming the controller. Efficient controller routing algorithms can be implemented with this knowledge without requiring any communication with the application.
2) Dynamic scenario: Both the topology and the traffic patterns change over time. In order to ensure an optimal routing (e.g., in terms of bandwidth or latency), the controller must automatically detect the topology and communicate with applications to synchronize routing and transmission behavior.

In particular, the communication between the application and the controller has great potential but there is neither a standardized nor widely used northbound APIs yet. The Open Networking Foundation is working on the standardization of northbound APIs [18] but currently it is still unclear whether an appropriate standard similar to the OpenFlow protocol will gain acceptance since the communication required between controller and application is very specific for the respective application.

B. *SDN and automation networks:*

Common RT capable Ethernet automation networks use time slots in order to allow individual network stations the exclusive access to the entire network. This is to prevent concurrent data transfers leading to unexpected delays in switches, which is inacceptable for an RT system. In those systems, parallel communication during one time slot is not possible today because there is no instance, which has knowledge of topology and routes used and could guarantee conflict-free parallel communication. In this regard, an automation network for RT data transfer can greatly benefit from the support of SDN as SDN allows parallel communication during one time slot without causing unexpected delays. Parallel communication within the time slots could be used both to increase the bandwidth and to shorten the cycle time (the time until all of the devices were allowed to send once). Alternatively, more devices may be included in the network without deteriorating the properties in terms of RT capability. Thereby, the potential benefit depends on topology and controller implementation, which will be detailed in the following section.

IV. Emulation of an SDN-Supported RT Ethernet Automation Network with Mininet

In this section, we describe our investigation of using Mininet and OpenFlow for the example of our developed RT Ethernet automation network called HaRTKad (A Hard Real-Time Kademlia Approach) [19]. The objective was to evaluate the feasibility and potential benefits of SDN in automation networks for RT data transmission. Since for an RT data transmission, the adherence to maximum transmission times must be guaranteed especially the measurement of RTTs in the network is of interest. Therefore, in Mininet a variety of measurements using the ping tool was conducted to collect relevant data for a HaRTKad network controlled by OpenFlow. For a RT system, especially the maximum RTT as well as the fluctuation of the measured values are of importance to evaluate the reliability of the system regarding the RTT. Moreover, the impact of simultaneous communication of multiple devices on the RTT (multiple transmitting devices within a network) was investigated and optimization opportunities by means of various topologies and routing procedures were analyzed.

A. *HaRTKad: An exemplary automation network*

A system developed by us called HaRTKad represents an example of a fully decentralized RT Ethernet automation network [19]. Like common RT capable Ethernet networks, it uses time slots to grant individual devices the exclusive access to the network. During the time slots, the devices periodically exchange process data with each other. The coordination of exclusive access is carried out by means of a peer-to-peer (P2P) system and without a central control. Moreover, in this system short-term interruptions of the RT communications are provided in the form of a maintenance phase, which can be used by the SDN controller for runtime configuration of the SDN-enabled switches.

B. *POX controller framework*

Since the built-in controller in Mininet does not provide the necessary functionality for our tests, a controller based on Python network operating system (POX) was used in the experiments. POX is a framework for developing OpenFlow controllers in the programming language Python [20], [21]. While Mininet provides fast and easy access to a virtual network as test environment, POX allows the rapid implementation of new controllers for this network. For this, in POX basic functions needed by any OpenFlow controller are already implemented. Furthermore, POX can be extended by the desired functionality (e.g., new routing algorithms) with self-written Python modules. The following modules were used in the practical part of this work:

- openflow.discovery discovers the network topology.
- openflow.spanning_tree must be used together with the openflow.discovery module. It uses information obtained during the discovery to establish a spanning tree.
- forwarding.l2_pairs makes OpenFlow switches behave like common layer 2 switches (flooding for unknown destination MAC addresses, targeted forwarding for known destination MAC addresses).

C. *Test arrangement and procedure*

To perform the measurements, the behavior of the HaRTKad system had to be emulated in Mininet using the Python API. In the performed measurements, the traffic pattern comprises ICMP echo request and reply packets to emulate the

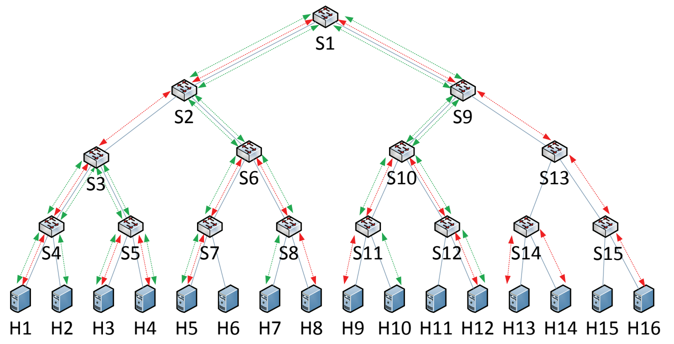

Fig. 3. Binary tree topology with routes of two different traffic patterns. Red arrows show the routes of non-overlapping paths (see Table I no. 1-5). Green arrows show the routes of overlapping paths (see Table I no. 6-7). ICMP echo requests and responses take the same path.

process data exchange in HaRTKad system. The traffic pattern specifies, which virtual hosts simultaneously send an ICMP echo request to which virtual target hosts. Using this traffic pattern, the training phase is started. In this phase, pings are sent once resulting in the installation of the appropriate entries in the flow table of OpenFlow switches. In order to ensure that during the measurement no (relatively slow) communication between switches and controllers is necessary, no timeouts are set for these entries. Finally, the measurement is performed. In this case, all ICMP echo requests of the traffic pattern are triggered simultaneously (as far as this is possible in software) and the measured RTT is logged. For a series of measurements, this step is repeated 500 times and from the collected data, the minimum, maximum, and average RTT as well as the standard deviation are calculated.

All virtual Ethernet connections are limited to a speed of 1 Mbit/s by means of the Linux traffic control. The payload of the ICMP echo requests is set to 1472 bytes. With this configuration, measurement values for the RTT as high as possible shall be generated so that influences of the scheduler and the inaccuracy of the time measurements are below the RTT by orders of magnitude. Also, a CPU limitation of the emulation for the virtual networks is avoided by configuring slower Ethernet connections. Tests have been carried out for four topologies: single switch, binary tree, mesh grid, and FatTree topology. Due to space constraints, solely results for the binary tree and FatTree topology are presented.

Binary tree topology: The following measurements were made for a Binary tree topology (see Figure 3) with a total of 15 virtual switches and 16 virtual hosts. The forwarding.l2_pairs module of the POX framework was used for reactive installation of flow entries. The two traffic patterns comprise (1) the simultaneous transmission of several ICMP echo requests from hosts with non-overlapping paths as well as (2) the simultaneous transmission between hosts with overlapping paths. ICMP echo responses take the same path like ICMP echo requests in the opposite direction. As expected, the series of measurements with this topology showed that the RTT is proportional to the number of the traversed switches and that the parallel communication over non-overlapping paths does not cause any delay (see Table I no. 1-5) while any overlapping of the paths causes delays (see Table I no. 6-7). Moreover, it becomes apparent that this topology allows very few combinations of parallel communication since no different selectable paths exist between hosts. For an extensive parallel communication between hosts, a better topology is required.

TABLE I. BINARY TREE TOPOLOGY EVALUATED WITH TWO TRAFFIC PATTERNS: (1) NON-OVERLAPPING PATHS (NO. 1-5) AND (2) OVERLAPPING PATHS (NO. 6-7).

No.	Src. Host	Dst. Host	Max. RTT [ms]	Mean RTT [ms]	Std. Dev. [ms]	# Traversed Switches
1	H3	H4	1.26	0.090	0.094	1
2	H13	H14	1.359	0.095	0.107	1
3	H9	H12	4.672	0.146	0.228	3
4	H5	H8	6.449	0.156	0.307	3
5	H1	H16	9.144	0.270	0.556	7
6	H1	H4	38.301	3.430	5.418	3
	H2	H3	30.516	3.500	5.435	3
7	H7	H10	32.139	3.045	4.959	7
	H5	H12	40.628	3.768	5.563	7

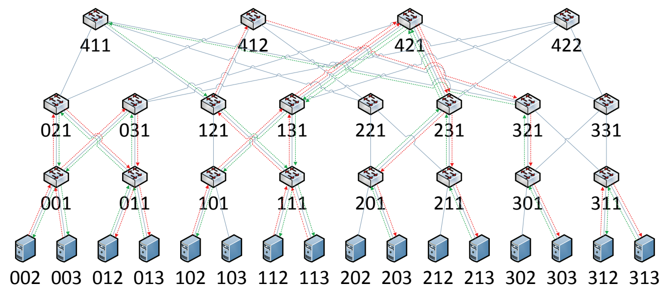

Fig. 4. FatTree topology with six sending virtual hosts and random RipL-POX routing (see Table II). Red arrows show the routes of ICMP echo requests. Green arrows show the routes of ICMP echo responses.

FatTree topology: The measurement series shows how efficient parallel communication is if a suitable topology and a controller that can exploit the potential of the topology are used. The FatTree topology is hierarchical and has been designed for the efficient networking of data centers. It provides the following benefits [22]:

1) The network shall provide enough bandwidth to make the total bandwidth of the attached hosts completely usable (realized by redundant paths and same bisection bandwidth at all levels of the tree).
2) Similar and inexpensive switches shall be used throughout the network.
3) The network should be easily scalable.

An appropriate topology for 16 hosts is apparent from Figure 4. The labels for hosts and switches are based on [22]. For emulating a FatTree topology in Mininet, the ”Ripcord Lite” (RipL) Python library was selected [23]. To exploit the potential of the topology, the POX module ”Ripcord Lite for POX” (RipL-POX) was used [24]. This module provides routing methods to utilize redundant paths. For the tests, Ripl-POX was configured so that flows are reactively installed and the used paths are selected randomly from a previously calculated set of valid paths. It is therefore to be expected that redundant paths are used but due to the random selection not the best routing is always achieved. For this topology, one traffic pattern with randomly selected paths was used resulting in non-overlapping paths (see Table II no. 1-4) and overlapping paths (see Table II no. 5). Thereby, ICMP echo requests and responses took another way in two test cases (no. 2 and 4 in Table II). Like in case of the binary tree topology, the RTT is proportional to the number of the traversed switches, parallel communication over non-overlapping paths does not cause any delay (see Table II no. 1-4), and any path overlapping leads to delays (see Table II no. 5).

TABLE II. FatTree topology with six sending hosts and randomly selected paths resulting in non-overlapping paths (no. 1-4) and overlapping paths (no. 5).

No.	Src. Host	Dst. Host	Max. RTT [ms]	Mean RTT [ms]	Std. Dev. [ms]	# Traversed Switches
1	312	313	3.619	0.100	0.190	1
2	003	012	8.21	0.143	0.373	3
3	002	013	2.43	0.148	0.168	3
4	113	303	3.852	0.187	0.208	5
5	102	213	30.249	2.803	4.583	5
	112	203	37.513	3.047	4.843	5

D. Results evaluation

Non-overlapping routes: For all investigated topologies and traffic patterns, the average RTT of below one millisecond for non-overlapping routes results from the fact that the traffic control does not cause any delays. As bandwidth and delay of the underlying virtual Ethernet adapters are only dependent on the available computing power, the recorded values only depend on the scheduling of the processes involved in packet forwarding. Since the scheduler is not optimized for latency-dependent processes and during series of measurements not exactly the same scheduling is performed for each sent ping, the measured values fluctuate. Moreover, all measurements with parallel and non-overlapping communication routes do not show significant changes in the measured properties. In contrast to the absolute values of RTT and the fluctuations, this behavior is consistent with what one would expect from a real hardware testbed with switches: packets forwarded via different ports do not interfere. Furthermore, one can observe that packets over several switches require a larger average RTT and standard deviation (see Table I no. 1-5 and Table II no. 1-4) as more processes are involved in the transmission of packets. This also corresponds to the behavior of a hardware testbed, in which each traversed switch contributes an additional delay (the exact time delay is implementation dependent).

Overlapping routes: For all measurements with overlapping routes, both the influence of scheduling and bandwidth limitation is relevant. If ICMP echo requests meet in a switch the, which arrived at the switch later, is delayed (see Table I no. 6) as it has to wait until the traffic control allows the packet to be sent. The added delay of 12 ms (explanation see following Section IV-E) for the second packet corresponds to the expected delay of a corresponding hardware testbed very well. As it is stochastic, which of the two ICMP packets arrives at a switch first, it can be assumed that two hosts measure an RTT increased by 12 ms for 50 % of their sent packets. Therefore, both hosts should measure an average RTT of 6 ms. The measured values are slightly lower due to the influence of the scheduler. The maximum values are subject to the random impact of the scheduler again. Only by traffic control, a maximum delay of about 12 ms is caused. The standard deviation is therefore generally not meaningful in the measurements with overlapping paths (see Table I no. 6-7).

Results summary: Overall, it becomes apparent that Mininet cannot emulate the time behavior of a real network but the impact of parallel communication is still visible. The measurements for binary tree and mesh grid topology also show that in addition to the topology the controller is important for efficient parallel communication. Although the mesh grid topology, in contrast to binary tree topology, offers many alternative routes they cannot be used by the controller. For this reason, the FatTree topology was chosen and operated with the RipL-POX controller. The concept of the FatTree topology to use cheap similar switches throughout the network perfectly matches the objectives of SDN and OpenFlow. Moreover, the redundant paths intended for increasing the bandwidth are also well suited to implement parallel communication. As the routes shown in Figure 4 and the measurements listed in Table II demonstrate, the RiPL-POX controller can use the existing paths effectively. Only two of the six performed pings are delayed due to overlapping paths (see Table II no. 5). Since RipL-POX was configured to select random routes, the results can differ for repeated executions of the test. Moreover, as RipL-POX has no routing method, which dynamically adjusts the chosen routes to the traffic pattern and avoids overlapping paths effectively, this controller is not yet sufficient for applications that require a guarantee for low delay (i.e., RT applications). Nevertheless, the great potential of the topology could be demonstrated with this experiment.

E. Comparison of Mininet results with real hardware testbed

In order to understand the results, one must consider how a real hardware testbed behaves with respect to RTT and latency and how this behavior is emulated by Mininet. An ICMP echo request with a payload of 1472 bytes including ICMP (8 bytes), IP (20 bytes) and Ethernet (14 bytes) headers is 1514 bytes in size. Including frame check sequence (FCS, 4 bytes), preamble (7 bytes) and start frame delimiter (SFD, 1 byte), there are 1526 bytes to be transferred physically. With a 1 Mbit/s Ethernet connection, the calculated transmission time equates to 12.208 ms. Since the ICMP echo reply packet has the same size and transmission time, the RTT should theoretically equate to at least 24.416 ms. In addition, there are further delays through switches and the cable. The propagation delay depends on cable lengths (negligible here); the delay through switches is implementation-dependent. For simplification, cut-through switching is assumed here so that there are almost no delays caused by the switches. For traffic patterns with non-overlapping routes, a theoretical average RTT of 25 ms would be expected. Here, the standard deviation of RTT in a real network should be very low and the maximum value should only slightly deviate from the mean value. The reason for this is that the packet processing for known flows is completely performed in hardware in typical switch implementations and is therefore deterministically predictable. The impact of the network on the RTT should therefore almost be constant. In the case of two overlapping routes, the packet, which arrives at the switch first, should not be delayed while the second experiences a delay of about 12 ms (transmission time of the first packet). Assuming that it is stochastic, which of the packets arrives first, an increase of the mean RTT of 6 ms and of the maximum RTT of 12 ms is expected compared to the situation with non-overlapping paths. As apparent from the experimental results, this behavior is not adequately modeled in Mininet. The basic differences from the previously described theoretical behavior of a real network are:

- Too low average RTT (< 1 ms) for non-overlapping paths (see Table I no. 1-5 and Table II no. 1-4).
- Increase of the mean RTT in the case of two overlapping paths is lower than the expected 6 ms (see Table I no. 6-7 and Table II no. 5).
- Very large standard deviations of all measurements.
- Maximum values are significantly different from the mean value, random outliers.

In the emulation, there are two important factors responsible for these differences: The scheduling of the processes and the operation of the traffic control. Both the scheduling and the traffic control need to be optimized to develop an RT capable Mininet called MininetRT to enable the timely exact emulate automation networks with hard RT capabilites.

Impact of the scheduler: The scheduling may affect the measurement in two ways. On the one hand, it can happen that packages are delayed as the process, which is responsible for the further processing of the packet, is not executed immediately upon arrival of the packet. On the other hand, it is possible that two hosts, which transmit simultaneously and introduce a delay for one of the packets due to overlapping paths, are executed with too long time interval and the mutual influence of the packets thereby decreases or even disappears. The operation of the traffic control has an impact on the measurements as it is responsible for emulating both the bandwidth and the delay properties of 1 Mbit/s Ethernet connections. Thus, the influence of traffic control and scheduling is very different. The traffic control determines whether the emulation in general has a realistic behavior in terms of RTT while the scheduling only causes random measurement errors.

Impact of the traffic control: In Mininet, the bandwidth of virtual Ethernet connections is configured with the Linux traffic control tool. A detailed description of the operation and capabilities of Linux traffic control can be found in [25], [26]. Each network adapter is assigned its own FIFO queue. Thereby, tokens are collected in a bucket and one token corresponds to the send permission for one byte. The bucket is continuously filled with tokens at a configurable rate. If a frame is sent it is checked whether enough tokens are available in the bucket (according to the frame length in bytes). If so the corresponding number of tokens is removed from the bucket and the frame is passed to the associated Ethernet adapter without any delay for transmission. Otherwise, the packet will be delayed until enough tokens are available in the bucket. This explains why the average RTT for all traffic patterns without overlapping routes amounts to less than 1 ms although one would theoretically expect higher delays when using 1 Mbit/s connections. Moreover, it follows from the described mechanism that the buckets have to be so small that they are empty after sending a packet and have to wait for new tokens.

V. Related Work

A. EstiNet OpenFlow network simulator and emulator

The EstiNet OpenFlow network simulator and emulator [27], [28] is another project, which aims at providing a fast and easy test option for OpenFlow networks and thus is comparable to Mininet. Similar to Mininet, EstiNet allows conducting experiments in virtual networks using an OpenFlow controller and host applications. It provides a simulation mode for maximum accuracy and an emulation mode, which allows for a faster realization of experiments at the cost of a small loss in accuracy. The underlying principle differs significantly from Mininet and is intended to better provide reproducible and accurate results. In [28], RTTs were measured for different sized OpenFlow networks and the results of the two EstiNet modes were compared with results from Mininet. The author shows that Mininet provides less reliable results and partly shows an inexplicable behavior, which did not occur in our measurements. The possibility to be able to provide better results in terms of RTT makes EstiNet an interesting tool for measurements. However, the test setup in [28] seems to be chosen unfavorably as static delays have been added to the virtual Ethernet connections, which are intended to simulate the propagation delay of packets while the transmission time of packets is neglected. This is exactly the opposite approach to our approach as we take into account transmission times for the consideration of real delays in parallel communication.

B. KVM-supported RT data transmission in OpenFlow nets

A work of the University of Tokyo [29], which has already dealt with RT communication over OpenFlow networks, proposes an extension of the Linux KVM (Kernel-based Virtual Machine) module by an RT capable virtual Ethernet adapter called RTvNIC, which allows virtual machines a RT communication over an OpenFlow network. The treatment of all virtual network adapters is expanded by one Earliest Deadline First (EDF) packet scheduler, which prefers packets from RTvNICs to packets from normal virtual Ethernet adapters. Apart from this advanced system for virtual Ethernet adapters, a new component is provided in KVM, which communicates with the OpenFlow controller. This component shall request the reservation of bandwidth from the OpenFlow controller for all paths required by RTvNICs. The authors have compared a working implementation of their system in a virtual environment with a system without RTvNICs and notice a significant improvement of RTT. Because of the implementation, is to be assumed that only soft RT is achieved. Guarantees to meet time deadlines are not possible so that the system does not provide hard RT capabilities, which are necessary in automation networks.

C. Achieving end-to-end RT QoS with SDN

The work in [4] aims at creating an end-to-end RT time QoS communication service based on SDN. A deterministic network model is introduced to compute optimal paths and a simulation-based performance analysis validates the concept for one selected network scenario. The authors' idea improves the average link utilization from 30 % to more than 60 % in comparison to traditional QoS reservation approaches. Although the contribution underlines the feasibility of SDN-supported automation networks, it is limited to one topology (ring topology) and should be extended to various topologies. Moreover, it should be possible to communicate the requirements of RT applications to the controller. Furthermore, a future implementation of the approach should prove that it can be realized as online routing algorithm in practice.

VI. Conclusion

In this publication, we have assumed that SDN and OpenFlow can bring immense benefits in the field of automation networks. OpenFlow is an important component in the SDN environment and has already established as well-usable southbound API as the protocol is supported by both hardware manufacturers and many controller frameworks. Moreover, SDN provides another significant opportunity to use automation networks more efficiently by implementing the controller and developing a northbound API in an appropriate way.

We used the popular and widely accepted Mininet emulator to carry out experiments by emulating an exemplary automation network. Recent research works have formulated criteria that must be met so that Mininet experiments can provide timely exact results in the emulation. However, this paper has shown that these criteria are not sufficient to allow for a precise emulation of automation networks with hard RT capabilities. This publication reveals the reasons for that, which further works of the research community can use as starting points, to develop an RT capable MininetRT by means of optimized scheduling and traffic control. Overall, we believe that SDN-supported automation networks offer enormous potential by allowing parallel communication of multiple devices during a time slot and still guaranteeing hard RT.

Acknowledgement

This work is partly granted by the Research Fund Mecklenburg-West Pomerania, Germany, as well as the European Social Fund.

References

[1] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

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

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[4] J. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 70–76.

[5] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," General Electric, Tech. Rep., November 2012.

[6] Communication Promoters Group of the Industry-Science Research Alliance, acatech - National Academy of Science and Engineering, "Recommendations for implementing the strategic initiative INDUSTRIE 4.0," Industrie 4.0 Working Group, Tech. Rep., April 2013. [Online]. Available: http://www.plattform-i40.de/finalreport2013

[7] M. Felser, "Real time ethernet: Standardization and implementations," in *IEEE International Symposium on Industrial Electronics*, 2010, pp. 3766–3771.

[8] P. Danielis, J. Skodzik, V. Altmann, E. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Sept 2014, pp. 1–8.

[9] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Palo Alto, CA, USA, White paper, Apr. 2012. [Online]. Available: http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[10] SDxCentral, "What are sdn northbound apis?" March 2015. [Online]. Available: https://www.sdxcentral.com/resources/sdn/north-bound-interfaces-api/

[11] B. Lantz, B. Heller, N. Handigol, V. Jeyakumar, and B. O'Connor, "Mininet - an instant virtual network on your laptop (or other pc)," March 2015. [Online]. Available: http://mininet.org/

[12] B. Heller, "Reproducible network research with high-fidelity emulation," Ph.D. dissertation, Stanford University, 2013.

[13] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 253–264.

[14] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.

[15] Open Networking Foundation, "Openflow switch specification," Tech. Rep., December 2009. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf

[16] E. W. Biederman, "ip-netns - process network namespace management," March 2015. [Online]. Available: http://man7.org/linux/man-pages/man8/ip-netns.8.html

[17] T.-Y. Huang, V. Jeyakumar, B. O. Bob Lantz, N. Feamster, K. Winstein, and A. Sivaraman, *Teaching Computer Networking with Mininet*, ACM SIGCOMM 2014, August 2014. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2014/doc/slides/mininet-intro.pdf

[18] Open Networking Foundation. (2013, October) Open networking foundation introduces northbound interface working group.

[19] J. Skodzik, P. Danielis, V. Altmann, and D. Timmermann, "Hartkad: A hard real-time kademlia approach," in *11th IEEE Consumer Communications & Networking Conference (CCNC)*, 2014, pp. 566–571.

[20] "Noxrepo.org," 2008. [Online]. Available: http://www.noxrepo.org/

[21] A. Al-Shabibi, "Pox wiki," March 2015. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki

[22] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[23] B. Heller. (2012, March) Ripcord-lite on github. [Online]. Available: https://github.com/brandonheller/ripl.

[24] ——. (2013, February) Ripcord-lite for pox on github. [Online]. Available: https://github.com/brandonheller/riplpox

[25] B. Hubert, "Linux advanced routing & traffic control howto," March 2015. [Online]. Available: http://www.lartc.org/howto/

[26] M. A. Brown, "Traffic control howto," 2006. [Online]. Available: http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/

[27] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "Estinet openflow network simulator and emulator," *Communications Magazine, IEEE*, vol. 51, no. 9, pp. 110–117, September 2013.

[28] S.-Y. Wang, "Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet," in *Computers and Communication (ISCC), 2014 IEEE Symposium on*, June 2014, pp. 1–6.

[29] K. Suzuki, "Real-time virtual nic on kvm for real-time network with openflow," LinuxCon Japan, Tech. Rep., 2013. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/lcjpcojp13_ksuzuki.pdf